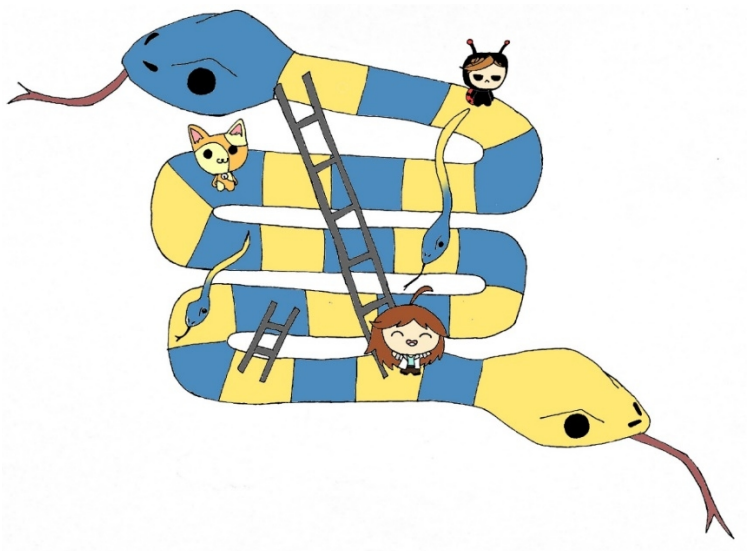
Kodea ondo sartzek



Programazioa ikasten Pythonekin

Pello Xabier Altadill

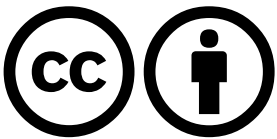
June Altadill

http://pello.io/programazioa

Kodea ondo sartzek: programazioa ikasten,

Egilea: Pello Xabier Altadill Izura

Marrazkiak: June Altadill



Hitzaurrea

Liburu hau programatzen ikasteko gida bat da. Horretarako, ahalik eta tresnarik sinpleenak erabiltzen ditu, Python bezalako hizkuntza erraz eta erabilgarria eta programak lehen kapitulutik idazten hasteko aukera. Errazenetik hasi eta pixkanaka teknika berriak sartzen dira.

Baina zerbait argi izan behar duzu. Irakurtzera mugatzen bazara, baliteke ideiaren batekin geratzea, baina ez duzu benetan programatzen ikasiko. Pro bihurtzeko, ordenagailua aurrean izan behar duzu eta hemen azaltzen dizuguna praktikatu. Proposatutako ariketak egiten, idazten, probatzen eta zure gustura hobetzen saiatu behar duzu. Programatzen programatuz ikasten delako.

Liburu hau, ebatzitako ariketak eta web orri bat jartzen ditugu zure eskura. Bertan sartu besterik ez duzu egin behar, eta proposatzen dizkizugun praktikekin jolastu ahal izango duzu:

http://pello.io/programazioa

Edukien aurkibidea

1 Programazioa ikasten

Ordenagailuak azkarrak dira eta memoria izugarria dute: segundoko milioika jarraibide exekutatzeko eta datu kopuru imajinaezinak kudeatzeko gai diren makinak dira. Baina horrek ez du esan nahi bizkorrak direnik. Berez ez dakite ezer egiten.

Ordenagailu batek, tablet batek edo mugikor batek zerbait zehatza egin dezaten, programa bat exekutatu behar dute. Eta programa bat programatzaileek idatzitako agindu-multzo bat da.

Programazioaren alderdi interesgarriena da makinaren kontrola har dezakegula, hark guk nahi duguna egin dezan: instrukzio sinple batetik hasi eta nabigatzaile edo joko oso konplexuak programatzeraino. Programazioa lotuta dago, nahi duzuna sor dezakezulako, zure irudimena eta trebetasunak beste mugarik gabe.

Baina zein hizkuntzatan komunikatu gaitezke ordenagailuekin? Horietako asko daude. Barnean, ordenagailuak hizkuntza bitarra erabiltzen du, hau da, zerbait egiteko interpretatzeko gai den zeroen sekuentzia. Hala ere, programazio-lana errazteko, hizkuntza sinpleagoak daude, pertsonen hizkuntzaren antz handia dutenak.

Python lengoaia horietako bat da, eta bertute asko ditu: erraza da, ikasteko oso erraza, edozer egiteko aukera ematen du, liburu denda ugari ditu eta, gainera, modu profesionalean erabiltzen da. Liburu honetan Python erabiliz programatzen ikasiko duzu. Pixkanaka, hizkuntzak eskaintzen dizkizun tresna berriak ezagutuko dituzu, programa konplexuagoak sortzen ikasteko.

Zer behar duzu oraintxe bertan hasteko? Nabigatzaile bat. Lehenengo kapitulutik programatzen hasiko gara. Programatzen ikasteko modurik onena... programatzea da! Ez da beldurrik izan

behar, eta, gainera, dibertigarriena da. Beraz, nahikoa da. Pasa hurrengo kapitulura zure lehen programa idazteko!

Oharra 

Agian entzuna duzu ordenagailuek adimen artifiziala izan dezaketela edo etsaiak oso azkarrak diruditen jokoei aurre egin diezula. Egia esan, ordenagailuak jokabide adimendunak imitatzen dituzten programak exekutatzen ditu.

Oharra 

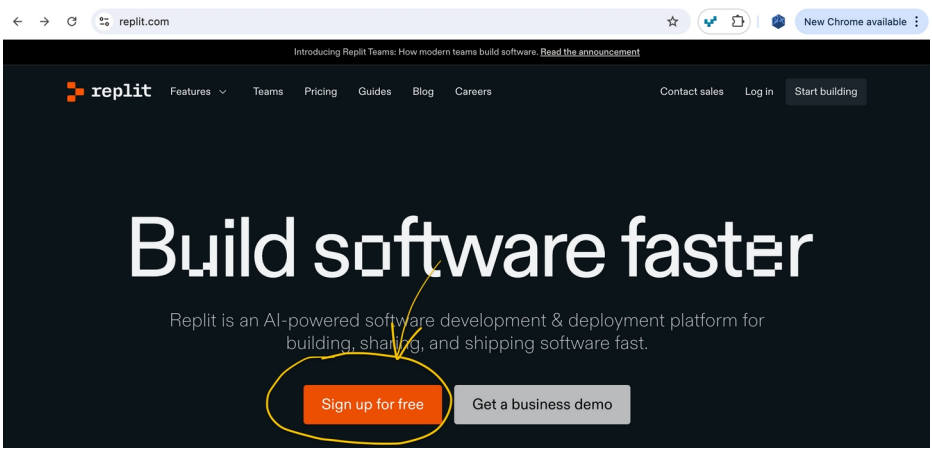
Ohiko programazio-ingurune bat instalatu nahi baduzu, zoaz 17. kapitulura.

2 Lan ingurunea

Interneterako konexioa duen edozein ordenagailutan sartu:

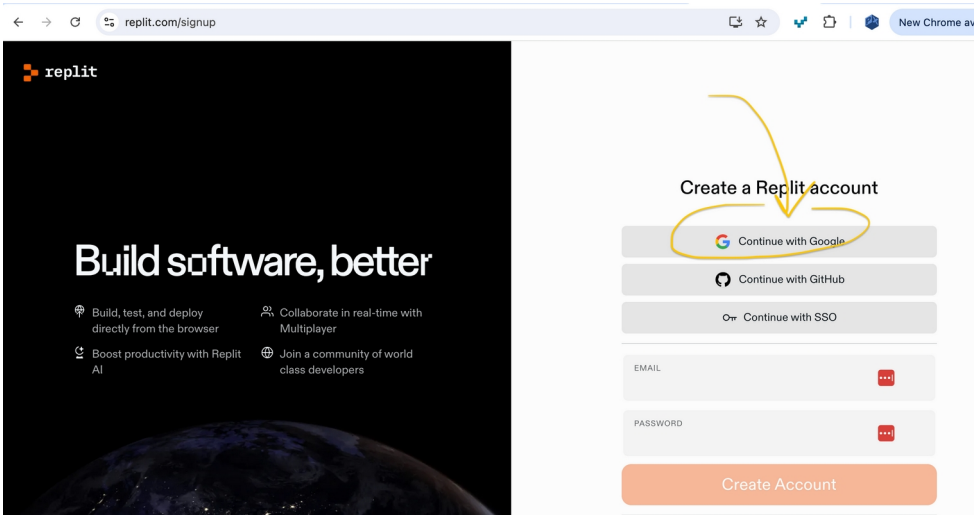
https://repl.it/

Hortik, `Sign up for free` sakatu



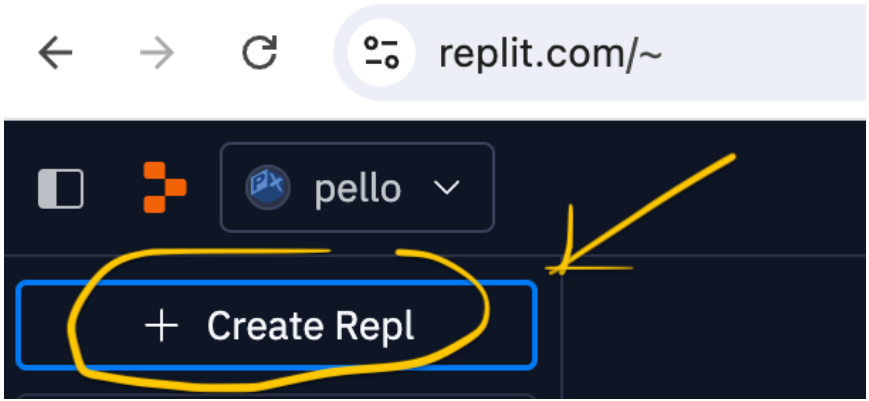
replt.it gunea

Kontu berria sortu edo Google kontuarekin konektatu:



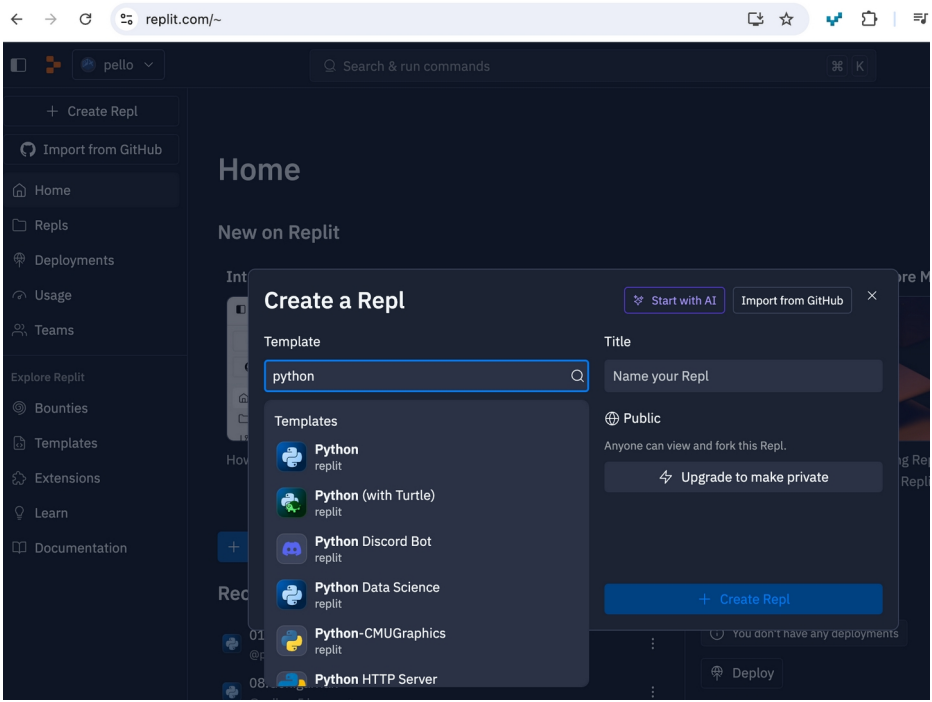
Google bitartez izena ematen.

Behien repl.it barruan zaudela, `new repl` botoia sakatu:



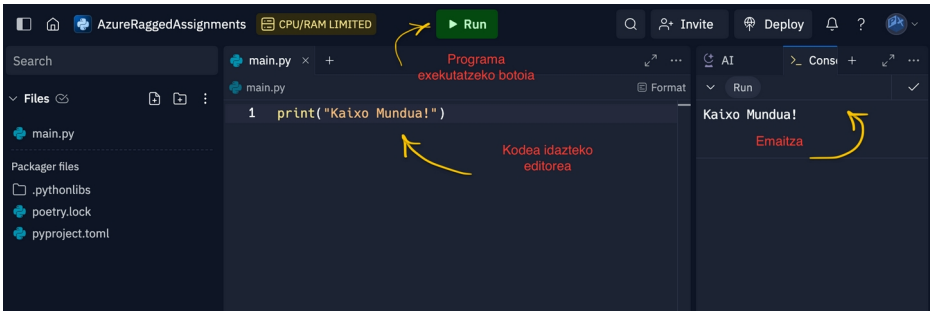
New Repl botoia sakatu.

Hurrengo pantaila ikusiko duzu. Bilatu Python eta hasieratu ingurunea:



replt.iten ingurune mota aukeratzen.

Behin hori eginda, programazio ingurunea kargatuko da. Hiru gune nagusi dira hemen:



Ezkerrean, fitxategi eta direktoriak dauzkazu.

Erdian, programa idazteko editorea duzu.

Eskuinean, kontsola, non ikusiko duzun emaitza programa exekutatzen duzunean.

Goiko aldean, Run botoia, programa nahi beste aldiz exekutatzeko.

Repl.it. gunean izen ematea gomendatzen dizugu. Horrela, egiten dituzun programa guztiak gordeta eta lokalizatuta izango dituzu.

Beste online aukerak:

- https://paiza.io/es
- https://www.programiz.com/python-programming/online-compiler/

3 Kaixo mundua

Programatzaileek idatzi ohi duten lehen programa mezu bat pantailatik ateratzea da. Eta mezu hori munduari egindako agurra izan ohi da: "Kaixo mundua!" Honela egiten da:

```
print ("Kaixo mundua!")
```

Hori exekutatu edo egikaritzen baduzu (Run), pantailatik horrelako zerbait ikusi beharko zenuke:

```
Kaixo mundua!
```

`print` Python lengoaiaren funtzio bat da, mezuak pantailan erakusteko aukera ematen diguna, eta askotan erabiliko dugu mezuak, emaitzak eta abar erakusteko.

KONTUZ! 

Kapitulu honetakoak bezalako programa oso sinpleetan, saiatu ez jartzen tarterik programaren aginduen aurretik, edo Pythonek errore bat emango du:

```
    print("Kaixo mundua!")
```

Oker legoke:

```
print ("Kaixo mundua!")
^

IndentationError: unexpected indent
```

Python-en espazioak edo tabulazioak gehitzen dira kodea beste bloke batzuen barruan dagoela adierazteko, pixkanaka ikusiko duzun bezala. Oraingoz, kapitulu honetarako, hasi zure kodea lerroaren hasieratik.

Zure txanda da!

0.0 Ariketa 

Idatzi zure izena pantaila bidez erakusten duen programa bat.

```
print("Kaixo, Ada naiz")
```

Emaitza:

```
Kaixo, Ada naiz.
```

3.1 Iruzkinak

Programa batean, iruzkinak edo komentarioak jar daitezke. Exekutatzen ez den testua da, ordenagailuarentzat existituko ez balira bezala. Zertarako erabiltzen da? Oro har, iruzkinak programaren zati jakin batzuk azaltzeko erabiltzen dira.

```
#Programa honek Kaixo dio

print ("Kaixo")
```

Pythonek ez dio jaramonik egiten iruzkinari, eta kasu honeta pantailatik `"Kaixo"` erakutsiko du, besterik ez. Hainbat lerroko iruzkinak ere egin daitezke:

```
""
```

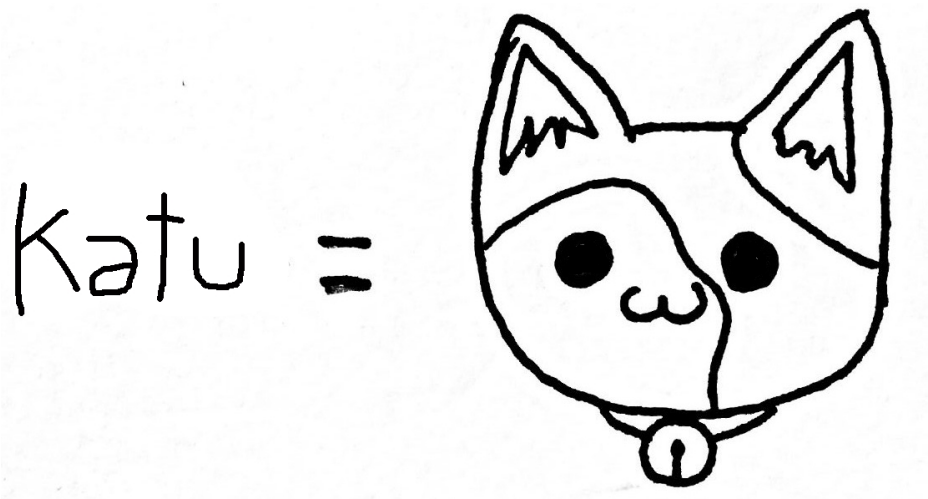
```
Python programa bat da.

ADAk sortua

eta Nekok berrikusia

""
```

Batzuetan, iruzkinak aldi baterako erabiltzen dira exekutatzea nahi ez dugun kodearen zati bat "desaktibatzeko".

Oharra 

Oro har, iruzkinak saihestu behar dituzu. Programatzaile on batek programa ulerterrazak idazten saiatu behar du, inolako iruzkinik edo azalpenik behar ez dutenak.

4 Aldagaiak



Aldagaiek datuak gordetzeko balio dute. Programek, funtsean, arazo bat konpontzeko eta emaitza bat emateko datuak maneiatzen dituzte. Prozesu guztian zehar datuak gordetzea beharrezkoa da, eta horretarako aldagaiak erabiltzen dira. Aldagaiak datu edukiontziak bezalakoak dira. Sukaldaritzan erabiltzen diren edalontziak eta platerak bezalakoak dira nolabait: zerbait dute, horrekin lan egiten da, nahastu egiten da, prozesatu egiten da eta

emaitza bat lortzen da: zorte pixka batekin zerbait goxoa lortzen da.

Python aldagai bat definitzeko, nahikoa da bere izena adieraztea eta balioren bat ematea. Adibidez:

```
izena = "Ada"
```

`"Ada"` balioa duen aldagai bat sortu berri dugu. `"Ada"` datu bat da, eta testu motakoa da. Orain, aldagai horren balioa bistara dezakegu pantailan:

```
print(izena)
```

Emaitza pantailan:

```
Ada
```

Edozein unetan alda dezakegu aldagai horren balioa:

```
izena = "Ada"
print (izena)
izena = "Neko"
print (izena)
```

Pantailan hurrengoa ikusiko dugu:

```
Ada

Neko
```

Oharra 

Aldagai baten edukia mezuaren zati gisa ere erakuts daiteke. Horretarako hainbat aukera daude, jarraian ikusiko dugun moduan.

4.1 Komen bidez bereizi

Nahikoa da aldagaiak eta testua komekin tartekatzea:

```
izena = "Juan"

adina = 34

print("Zure izena da", izena, ",", adina,
"urte dituzu")

# print("Zure izena da %s, %d urte dituzu" %
(izena, adina))
```

Emaitza pantailan:

```
Kaixo, Bug dut izena.
```

```
Bug naiz, 10 urte ditut.
```

Orain zu!

0.1 Ariketa 

Sortu bi aldagai izena eta adina, eta erakutsi haien balioa pantailan.

```
izena = "Ada"

adina = 14

print ("Zure izena", izena, "da", adina, "urte
dituzu")
```

Emaitza:

```
Zure izena Ada da eta 14 urte dituzu.
```

4.2 Formatodun testua

Aldagai bat baino gehiago erakusteko beste modu bat mezu bat da. Aurretik, `f` letra eta giltzen arteko aldagaiak daude:

```
izena = "Bug"

adina = 10

print (f"Kaixo, {izena} dut izena")
```

```
print (f"{izena} naiz, {adina} urte ditut")
```

Emaitza pantailan:

```
Kaixo, Bug dut izena.
Bug naiz, 10 urte ditut.
```

KONTUZ 

Aukera hau Python 3.6tik bakarrik dago eskuragarri

4.3 Ehunekoa ordeztea

Aldagaiak pantaila bidez erakusteko beste aukera bat. `%s` elementuak aldagaiekin ordezkatzen dituen mezu bat sortzen da.

```
print ("Kaixo, %s dut izena" % izena)
```

Emaitza pantailan:

```
Kaixo, Ada dut izena.
```

Gauza bera egin dezakezu hainbat aldagairekin

```
izena = "Neko"
```

```
adina = 5

print ("Kaixo, %s dut izena eta %d urte ditut"
% (izena, adina))
```

Emaitza pantailan:

```
Kaixo, Neko dut izena eta 5 urte ditut.
```

5 Datu-motak

Datuak!

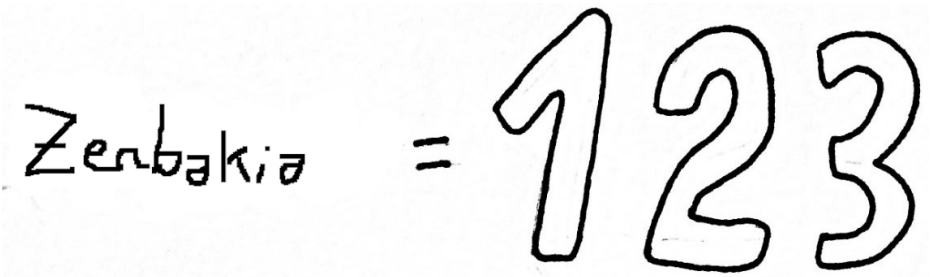
Programek lantzen duten osagaiak dira. Gure programek eraldatzen dituzten elementuak dira. Programa batek datuak jaso, eraldatu eta emaitza gisa itzultzen ditu. Datuak desberdinak izan daitezke, gure programak egin behar duenaren arabera. Zenbakiak izan daitezke, hitzak edo testuak izan daitezke, baliogabeak edo hutsak ere izan daitezke. Datuak gordetzeko, normalean aldagaiak erabiltzen ditugu.

Programazioa sukaldaritzarekin alderatzen badugu, orduan azukrea, irina eta arraultzak datuak izango lirateke, ontziak aldagaiak izango lirateke: tarta beste datu bat izango litzateke, emaitza eta errezeta programa izango litzateke.

Nola daki Pythonek zein datu mota darabilen? Ez dago esan beharrik, beste lengoai batzuetan bezala. Horregatik Python lengoai sinpleagoa eta malguagoa da. Baina, kontuz! ezin izango dugu nahi duguna egin datuekin.

Jarraian, oinarrizko datu motak ikusiko ditugu.

5.1 Zenbakiak



Mota guztietako zenbakiak dira:

5.1.1 Osoak: 1, 2, 3, 4,...

```
kontagailua = 10

adina = 12
```

5.1.2 Hamartarrekin:

Zenbaki hamartarretan, zati osoa eta hamartarra (4.5 edo 3.1415) bereizteko, `.` erabiltzen da. Baliteke eskolako matematika klasean koma bat erabiltzea hamartarrak bereizteko, baina programazioan ingelesezko formatua erabiltzen da eta `.` erabili behar dugu.

```
pisua = 34.67

prezioa = 242.9943
```

5.1.3 Negatiboak:

0 zenbaki txikiagoak gidoia edo marratxo batekin adierazten dira: -4, -5, -3.1415,...

```
nota = -5

tenperaturaMarten = -50.676
```

5.2 Testua

Testua, kateak edo string ere deituak, komatxo bikoitz edo sinpleen arteko edozein hizki multzoa da.

```
izena = "Ada"

esaldia = "

hitzak = 'Lanera noa'

neska = 'Lagunak izatea besterik ez dut nahi'
```

Testuaren kasuan, zenbait karaktere berezi sar ditzakegu, ondorio interesgarriak izateko. Karaktere horiek kontra-barra batekin edo backslash batekin idazten dira aurretik.

5.2.1 Lerro jauzia

Horrek lerro jauzi bat gehitzen dio testuari, pantaila bidez erakusten bada:

```
esaldia = "Kaixo,\n zer moduz"
```

Honela erakutsiko da:

```
Kaixo,
zer moduz
```

Hainbat lerrotako testu bat ere defini daiteke:

```
"""Igande bat zen.
arratsaldean
autoetara joan nintzen
talka"""
```

Tabulazioak Horrek tabulazio bat (hainbat espazio) gehitzen dio testuari, pantaila bidez erakusten bada:

```
esaldia = "Izena\tAbizena\tAdina"
```

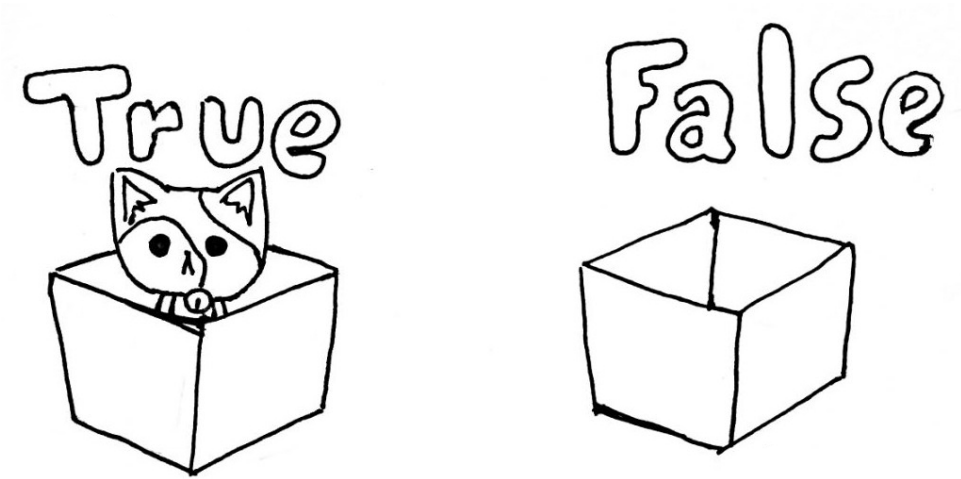
Honela erakutsiko da:

```
Izena Abizena Adina
```

Beste karaktere berezi batzuk:

- `\\` Testu batean kontrabarra erakusteko.
- `\"` Komatxo bikoitza erakusteko testu batean.
- `\'` Komatxo sinple bat erakusteko testu batean.
- `\a` Pitidoa jotzeko.

5.3 Boolearrak



Boolear motak bi balio posible bakarrik izan ditzake: `True` ala `False`: egia ala gezurra. Programazioan funtsezko datua da, erabakiak hartzeko erabiltzen baita.

```
amaituta = False

handiagoaDa = True

pythonItzelaDa = True
```

5.4 Zerrendak

Zerrendak datu multzoak dira, eta honela definitzen dira:

```
lagunak = ["Ada", "Miranda", "Ruby"]
```

Edozein motatakoak izan daitezke, baina normalena da zerrenda bateko elementu guztiak mota berekoak izatea:

```
hutsa = []

zenbakiak = [12, 16, 30, 0, 22, 1, 1, 12]

egiak = [True, False, False, True]
```

Zerrendako balio zehatz bat adierazteko, interesatzen zaigun zerrendako elementuaren posizioa adierazi behar dugu, 0tik hasita:

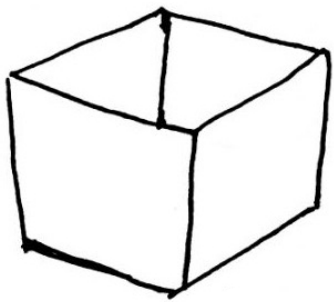
```
izenak = ["Ada", "Neko", "Bug"]

print (izenak [0]) # "Ada"
```

izenak zerrendaren kasuan, posizio posibleak 0, 1 eta 2 izango dira. Baina kontuz! Posizio handiegia pasatzen baduzu, programa akats batekin amaituko da:

```
izenak = ["Ada", "Neko", "Bug"]

print (izenak [4]) #ERROREA!
```

Aurrerago itzuliko gara zerrendetara eta beste egitura batzuetara.

5.5 None



Arraroa dirudien arren, programetan batzuetan hutsa edo ezereza irudikatzen duen zerbaitekin hitz egin behar izaten da. Badago hitz bat Python-en hutsa irudikatzeko aukera ematen diguna, eta hori da: `None`

```
hasierakoBalioa = None

datua = None
```

Egia esan, ez da aldagaiak sortzeko erabiltzen. `None` balio bat adierazten du egoera berezietan. Adibidez, fitxategi huts bat irakurtzen denean edo erabiltzaileak datu bat ematen ez digunean.

0.2 Ariketa

Idatzi hemen ikusitako mota bakoitzaren aldagai bat definituko duen programa bat, eta kontsola bidez erakutsi:

```
izena = "Ada"

adina = 42

pisua = 101.54

bizirik = True

dirua = None

lagunak = ["Ada", "Ruby", "Miranda"]

print (izena)

print (adina)

print (pisua)

print (bizirik)

print (dirua)

print (lagunak)
```

Emaitza:

```
Ada
```

```
42
101.54
True
null
["Ada", "Ruby", "Miranda"]
```

6 Datuak irakurtzen

Programa batek zerbait egin ahal izateko, askotan erabiltzaileak datu bat sartu behar du. Adibidez, programa batek gure izenak zenbat letra dituen edo gure urtebetetzerako zenbat falta den esatea nahi badugu, programak lehenik eta behin datu bat eskatu beharko du.

Oraingoz ikusten ari garenak bezalako oinarrizko programek kontsola erabiltzen dute exekutatzeko. Pantaila beltz horietan idatzitako aginduak jartzen dira :)

Erabiltzaileari datu bat eskatu eta aldagai batean gordetzeko, honako input funtzio hau erabiltzen da:

```
izena = input("Sartu zure izena: ")

print("Kaixo, zer moduz zaude ", izena)

# print("Kaixo, zer moduz zaude %s" % izena)
```

Pantailan honako hau ikusiko zenuke:

```
Sartu zure izena:
```

input funtzioaren ondorioz, pantailan agertzen da `Sartu zure izena:` mezua. Era berean, programa gelditu egiten da, erabiltzaileak zerbait idatzi arte. Erabiltzaileak `Rosa` idazten badu, honela ikusiko da:

```
Sartu zure izena: Rosa

Kaixo, zer moduz zaude Rosa
```

0.3 Ariketa

Idatzi erabiltzaileari izen bat eskatu eta aldagai batean gordeko duen programa bat. Jarraian, kontsola bidezko agurra egin behar duzu.

```
izena = input ("Sartu zure izena:")
print ("Kaixo, zer moduz zaude", izena)
#Alternatiba:
#print ("Kaixo, zer moduz zaude % s" % izena)
```

Emaitza:

```
Sartu zure izena: Juan

Kaixo, zer moduz zaude Juan
```

6.1 Kontuz datuekin

Erabiltzaileak zerbait idazteko, input funtzioa erabiltzen duzun bakoitzean testu gisa gordeko da. Nahiz eta zenbaki bat idatzi:

```
balioa = input ("Eman zenbaki bat:")

bikoitza = balioa + balioa

print(bikoitza)
```

Erabiltzaileak `"4"` bezalako zenbaki bat sartuz gero, emaitza hau izango litzateke:

```
Eman zenbaki bat: 4

44
```

`4 + 4` gehitu eta `8` erakutsi beharrean, `4` eta `4` elkartu egin ditu, izan ere, `4` irakurri duenean, `4` testua da: `"4"`.

Hori saihesteko, beste funtzio bat erabili behar dugu datu hori zenbaki oso bihurtzeko: `int()`

```
balioa = input ("Eman zenbaki bat:")

bikoitza = int(balioa) + int(balioa)

print(bikoitza)
```

Edo lehenago ere bihur dezakegu:

```
balioa = input ("Eman zenbaki bat:")
balioa = int(balioa)
bikoitza = balioa + balioa
print(bikoitza)
```

Edo, are gehiago, `input` egiten den momentuan bihurtu dezakegu:

```
balioa = int(input("Eman zenbaki bat:")
bikoitza = balioa + balioa
print(bikoitza)
```

Orain bai:

```
Eman zenbaki bat: 4
8
```

0.4 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat eta gehitu `10`. Jarraian, emaitza kontsola bidez erakutsi behar duzu. Gogoratu sartutako balioa zenbaki oso bihurtzeko sartzea komeni dela.

```
balioa = input("Sartu zenbaki bat: ")

emaitza = int(balioa) + 10

print("Batura hau da:", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 32

Batura hau da: 42
```

6.2 Beste bihurketa batzuk

Python lengoaian aldagai motak adierazi behar ez baditugu ere, motak (testua, zenbakia, etab.) kontuan hartzen dira programa exekutatzerakoan. Adibidez, zenbaki bat duen aldagai bat badugu eta testu batean kateatu nahi badugu, errore bat jasoko genuke:

```
balioa = 66

testua = "Nire adina da" + balioa
```

Errorea horrelako zerbait izango litzateke:

```
TypeError: can only concatenate str (not
"int") to str
```

Hori saihesteko, testu mota behartu behar dugu `str()` erabiliz:

```
balioa = 66

testua = "Nire adina da " + str(balioa)
```

Beraz, batzuetan balio bat mota jakin batean bihurtu beharko dugu. Hauek dira bihurtzeko funtzioak:

- `str()`: balio bat testu bihurtzen du. · `"5"` itzuliko luke.
- `int()`: balio bat zenbaki oso bihurtzen du.
- `float()`: balio bat zenbakihamartarra bihurtzen du.
- `bool()`: balio bat boolear batera bihurtzen du.

KONTUZ! 

Bateragarria ez den balio bat bihurtzen saiatzen bazara, programak huts egingo du eta bat batean amaituko da.

7 Eragileak

Programek zenbakiekin kalkuluak egin behar dituzte, baita datuak prozesatu eta balioen arabera erabakiak hartu ere. Horretarako operadore edo eragileak behar ditugu.

7.1 Eragile aritmetikoak

Batuketak, kenketak eta oinarrizko kalkulu guztiak balioekin eta aldagaietan gordetzen denarekin egiteko aukera emango dizuten guztiak dira; adibidez, batuketa:

```
txikleak = 4

txikleak = txikleak + 2
```

Programaren kalkuluen arabera, 4 txikle edukitzetik 6 izatera pasa zara. Hauek dira programazioko oinarrizko eragiketak:

Batuketa: +

Kenketa: -

Biderketa: *

Zatiketa: /

Adibidez, egun batek zenbat segundo dituen kalkulatzeko:

```
minutuak = 60

segunduak = 60

orduak = 24

segunduak = segunduak * minutuak * orduak
```

Behar bezain eragiketa konplexuak egin ditzakezu. Irakurtzeko errazagoak izan daitezen, parentesiak erabil daitezke, matematikan egiten den bezala:

```
ada = 14

bug = 10

neko = 2

batezbestekoa = (ada + bug + neko)/3

print(batezbestekoa)
```

0.5 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatu eta 5 kentzen dion programa bat. Jarraian, kontsola bidez erakutsi behar duzu emaitza.

```
balioa = input("Sartu zenbaki bat: ")
```

```
emaitza = int(balioa) - 5
print("Kenketa da:", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 30
Kenketa hau da: 25
```

7.2 Modulua eta berreketa

Programazioan oso garrantzitsua den eragiketa bat dago, agian mateetan hain ohikoa ez dena: modulua da. Zatiketaren emaitzaren ordez hondakina itzultzen duen zatiketa da:

```
balioa = 8
emaitza = balioa % 3
```

Emaitza hauxe izango da:

```
2
```

Berreketa zenbaki bat bere buruarekin hainbat aldiz biderkatzearen emaitza da. Pythonen operadore honekin egin daiteke eragiketa hori:

```
balioa = 2

emaitza = balioa ** 3 # baliokidea: 2 * 2 * 2
```

Emaitza `8` izango litzateke.

7.3 Zeinu aldaketa

Ondo dakizunez, zenbaki batzuk zero baino txikiagoak dira, eta negatibo esaten zaie. Zenbaki horiek aurretik `-` batekin adierazten dira:

```
-5, -248, -1.87,...
```

Zenbaki baten zeinua aldatu nahi badugu, aurretik `-` bat jar dezakegu:

```
tenperatura = -11

kontua = 200

tenperatura = -tenperatura # 11

kontua = -kontua # -200
```

7.4 Eragile laburtuak

Askotan, aldagai baten gainean jardun beharko duzu, eta emaitza aldagaian bertan gorde:

```
kontagailua = 0

kontagailua = kontagailua + 2
```

Horrelako egoeretan, eragile laburtu bat erabil dezakezu, eragiketa egin eta aldi berean esleitzen duena. Hori aurreko kodearen baliokidea izango litzateke:

```
kontagailua = 0

kontagailua += 2
```

Gauza bera egin daiteke operadore guztiekin:

Eragiketa	Horixe bera Laburtua
`a = a + 1`	`a + = 1`
`a = a – 1`	`a - = 1`
`a = a* 1`	`a* = 1`
`a = a/1`	`a/= 1`
`a = a % 1`	`a % = 1`

```
a = a** 1        a** = 1
```

0.6 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatu eta inkrementatu (`+1`) egingo duen programa bat. Jarraian, kontsola bidez erakutsi behar duzu emaitza. Ondoren, balioa dekrementatu (`-1`) behar du eta emaitza kontsola bidez erakutsi. Operadore laburtuak erabili!

```
balioa = int(input("Sartu zenbaki bat: "))
balioa += 1
print("Inkrementua da", balioa)
balioa -= 1
print("Dekrementua da", balioa)
```

Emaitza:

```
Sartu zenbaki bat: 6
Gehikuntza 7 da
Beherakada 6 da
```

7.5 Konparazio eragileak

Balio bat bestearekin alderatzeko aukera ematen diguten eragileak dira. Normalean zenbakiekin erabiltzen da, eta eragiketa horien emaitza `True` ala `False` da.

Adibidez, balio bat beste baten berdina den egiaztatzeko, `==` operadorea erabiliko dugu.

```
balioa = 5

emaitza = balioa == 5
```

Emaitza hauxe izango litzateke: `True`.

Hona hemen konparazio-operadoreak:

Berdin: `==`

Desberdina: `!=`

Handiagoa: `>`

Txikiagoa: `<`

Handiagoa edo berdina: `>=`

Txikiago edo berdina: `<=`

Testudun operadore hau ere erabil daiteke berdintasuna egiaztatzeko:

```
izena = "Ada"

Emaitza = izena == "Bug"
```

Emaitza hau izango litzateke: `False`. Era berean, testu bat ordena alfabetikoan handiagoa edo txikiagoa den alderatzeko aukera ematen digu:

```
izena = "Ada"

emaitza = "Ada" < "Bug"
```

Emaitza hauxe izango litzateke: `True`.

0.7 Ariketa

Idatzi erabiltzaileari bi zenbaki eskatzen dizkion programa bat. Gero, bere desberdintasuna alderatu behar du eta emaitza kontsolaren bidez erakutsi behar du.

```
balio1 = input("Sartu zenbaki bat: ")

balio2 = input("Sartu beste zenbaki bat: ")

emaitza = balio1 != balio2

print("Ezberdinak al dira?", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 42
Sartu beste zenbaki bat: 42
Desberdinak dira? False
```

7.6 Eragile boolearrak

Eragile boolearrek `True` ala `False` balio boolearrekin eragiketak egiteko aukera ematen digute.

7.6.1 and

Operadore horrek `True` itzuliko du, baldin eta bi operadoreak ere `True` badira:

```
balioa = 5
emaitza = (balioa == 5) and True;
```

Emaitza hauxe izango litzateke: `True`.

Aukera guztiak laburbiltzeko, egiaren taula delakoa honako hau izango litzateke:

a	and	b	emaitza
False	and	False	False
False	and	True	False
True	and	False	False
True	and	True	True

0.8 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat. Ondoren, zenbaki hori 0 baino handiagoa den eta, gainera, bikoitia den alderatu behar duzu.

```
balioa = input("Zenbaki bat sartu: ")
balioa = int(balioa)
emaitza = (balioa >= 0) and (balioa % 2 == 0)
print("Bikoitza eta positiboa da?", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 14
Bikoitza eta positiboa da? True
```

7.6.2 or

Operadore honek `True` itzuliko du, baldin eta gutxienez bat `True` bada:

```
balioa = 5
emaitza = (balioa == 5) or False;
```

Emaitza hauxe izango litzateke: True.

Aukera guztiak laburbiltzeko, hauxe izango litzateke `or` operadorearen egiaren taula.

False	or	False	False
False	or	True	True
True	or	False	True

```
False   or   False   False
```

```
True   or   True   True
```

0.9 Ariketa

Idatzi programa bat erabiltzaileari bi zenbaki eskatzeko eta bietako bat positiboa den egiaztatzeko. Jarraian, kontsola bidez erakutsi behar duzu emaitza.

```
balio1 = input("Sartu zenbaki bat: ")

balio2 = input("Sartu beste zenbaki bat: ")

emaitza = (int(balio1) >= 0) or (int(balio2)
>= 0)

print("Zenbaki batek ala besteek positiboa
dute?", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: -4

Sartu beste zenbaki bat: 6

Zenbaki batek ala besteek positiboa dute? True
```

7.6.3 not

Operadore horrek kontrako balioa itzultzen du eragiketan. `True` balio bati aplikatzen bazaio `False` itzuliko du eta alderantziz.

```
balioa = True

emaitza = not balioa
```

Emaitza hau izango litzateke: `False`. Aukera guztiak laburbiltzeko, hauxe izango litzateke operadorearen egiaren taula.

	a	emaitza
not	True	False
not	False	True

0.10 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat, eta egiaztatu ez dela ez positiboa ez bikoitia.

```
balioa = input("Sartu zenbaki bat: ")

balioa = int(balioa)
```

```
positiboaEtaBikoitia = (balioa >= 0) and
(balioa % 2 == 0)

emaitza = not positiboaEtaBikoitia

print("Bikoitia eta positiboa da?", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: -4

Bikoitia eta positiboa da? True
```

7.7 Operadoreak nahasten

Operadoreak behar adina konbina ditzakegu:

```
jubilazioAdina = 65

adina = 42

if adina > 17 and adina < (jubilazioAdina +
1):

 print ("Lan egin dezakezu")
```

Oro har, konparaketa eta boolear eragileak baldintzapeko blokeetan, begiztetan eta abarren baldintzen barruan erabiltzen dira. Aurrerago ikusiko dugu.

7.8 Proposatutako ariketak

0.0 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatu eta 7 biderkatuko dion programa bat. Jarraian, kontsola bidez erakutsi behar duzu emaitza.

```
balioa = input("Sartu zenbaki bat: ")
emaitza = int(balioa) * 7
print("Biderketa honakoa da:", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 3
Biderketa honakoa da: 21
```

0.1 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatu eta bitan zatikatuko duen programa bat. Jarraian, kontsola bidez erakutsi behar duzu emaitza.

```
balioa = input ("Sartu zenbaki bat:")
emaitza = int (balioa)/2
print ("Zatiketa da:", emaitza)
```

Emaitza:

```
Sartu zenbaki bat:60
Zatiketa da: 30.0
```

0.2 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatu eta 3rekin modulua (%) egiten duen programa bat. Jarraian, kontsola bidez erakutsi behar duzu emaitza.

```
balioa = input("Sartu zenbaki bat: ")
emaitza = int(balioa) % 3
print("Modulua honako hau da:", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 7
Modulua honako hau da: 1
```

0.3 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatu eta ber 2 (2ko esponentziala) aplikatuko dion programa bat. Jarraian, kontsola bidez erakutsi behar duzu emaitza.

```
balioa = input("Sartu zenbaki bat: ")
emaitza = int(balioa) ** 2
print("Esponentzialaren emaitza:", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 4
Esponentzialaren emaitza: 16
```

0.4 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatu eta 5 kentzen dion programa bat. Jarraian, zeinua aldatu behar diozu eta emaitza kontsolarekin erakutsi.

```
balioa = input("Sartu zenbakia: ")
kenketa = int(balioa) - 5
emaitza = -kenketa
print("Kenketa da:", emaitza)
```

Emaitza:

```
Sartu zenbakia: 4
```

```
Kenketa da: 1
```

0.5 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat. Ondoren, `% 2` eragiketa 0ren berdina den egiaztatu behar duzu, eta emaitza erakutsi. Zenbaki bat 2rekin zatitzen bada eta ondarra 0 bada, zenbaki hori bikoitia dela esan nahi du.

```
balioa = input("Sartu zenbaki bat: ")

moduloa = int(balioa) % 2

emaitza = moduloa == 0

print("Balioa bikoitia da?", emaitza)
```

Emaitza

```
Sartu zenbaki bat: 8

Balioa bikoitia da? True
```

0.6 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat. Ondoren, zenbaki hori 0 edo handiagoa den egiaztatu behar duzu, hau da, positiboa den.

```
balioa = input("Sartu zenbaki bat: ")
emaitza = int(balioa) >= 0
print("Positiboa da?", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 6
Positiboa da? True
```

0.7 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat. Ondoren, 0 baino txikiagoa den alderatu behar duzu, eta emaitza kontsolaren bidez erakutsi. Zenbakia negatiboa den antzematen ariko ginateke.

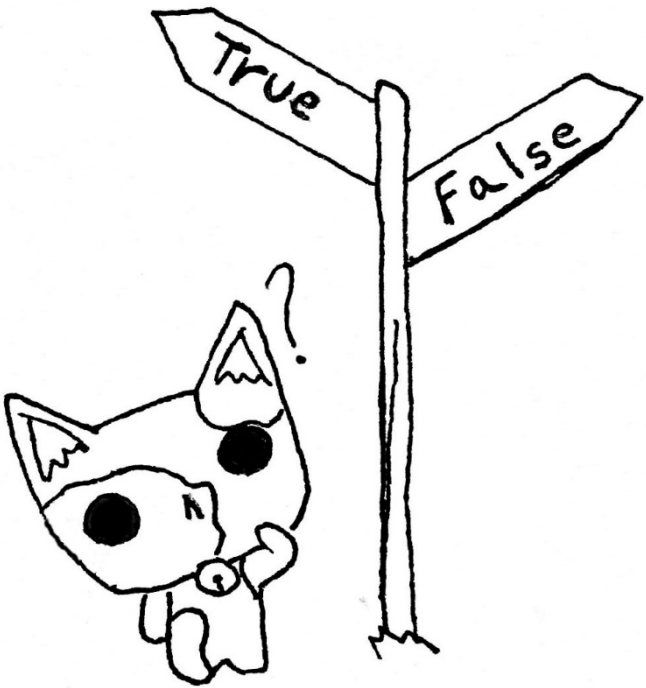
```
balioa = input("Sartu zenbaki bat: ")
emaitza = int(balioa) < 0
print("Negatiboa da?", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: -3
```

```
Negatiboa da? True
```

8 Baldintzak



Askotan, programek gauza bat edo beste egin behar dute, baldintza baten arabera. Adibidez, erabiltzaile batek datu oker bat sartzen badu, programa amaitu egingo da. Datu batek balio jakin bat badu, modu batera prozesatzen da, eta bestela, beste batera. Nola lortzen da portaera hori? Baldintzen bidez.

Baldintzak programazio egiturak dira. Egitura hauek aukera ematen digute kode bat exekutatzea soilik baldintza batzuk betetzen direnean.

8.1 if

Baldintza bat egiteko egiturarik sinpleena `if` bat da, eta itxura hau du:

```
if *baldintza*:
 *eragiketak*
 *eragiketak*
 *...*
```

Ikus dezakezunez, baldintza batekin hasten da. Baldintza boolear batek itzultzen duen edozein adierazpen izan daiteke, hau da, `True` ala `False`, egiazkoa ala faltsua. Hala bada, bete egingo dira jarraibideak, eta, bestela, salto egingo da.

Adibidez, programa honek, balioa negatiboa bada, "Balioa 0 baino txikiagoa da" pantailaratuko du:

```
balioa = -2
if balioa < 0:
 print ("Balioa 0 baino txikiagoa da")
print ("Programaren amaiera")
```

Honela ikusiko litzateke:

```
Balioa 0 baino txikiagoa da
Programaren amaiera
```

Aldiz:

```
balioa = 5
if balioa < 0:
 print ("Balioa 0 baino txikiagoa da")
print ("Programaren amaiera")
```

Honela ikusiko litzateke:

```
Programaren amaiera
```

Oharra 

Oso garrantzitsua den zerbait ere kontuan hartu behar duzu: `if` aren barruko aginduak espazio batzuen edo tabulazio baten atzean daude. Python programazio lengoaiaren berezitasun bat da hori: edozein bloketan, hala nola baldintza batean, begizta batean, funtzio batean, haren edukiak **tabulatuta** egon behar du. Espazio horrek irakurketa errazten du eta programa baten egitura erraz ezagutzea ahalbidetzen du beste programatzaile batzuentzat. Baita zuretzat ere, zure programa bada.

Programak konplexuagoak egiten diren heinean, baldintzak eta beste edozein kontrol egiturak kabiatuko ditugu, eta espazio horiekin programaren egitura jarraitzea erreztuko dugu.

1.0 Ariketa 

Idatzi erabiltzaileari zenbaki bat eskatzen dion eta negatiboa den egiaztatzen duen programa bat. Beraz "negatiboa da", mezu bat erakutsi behar duzu kontsola bidez.

```
balioa = input("Idatzi zenbaki bat: ")
if int(balioa) < 0:
 print("Negatiboa da")
```

Emaitza:

```
Idatzi zenbaki bat: -5
Negatiboa da
```

8.2 if else

`if`arekin bloke bat sor dezakegu, baldintza bat betetzen bada bakarrik exekutatzen dena. Baina zer gertatzen da programak baldintza baten arabera gauza bat ala bestea egitea nahi badugu? "Beste" aukera sartu ahal izateko, `if-else` egitura erabiltzen dugu:

```
if *baldintza*:
 *eragiketak*
else:
 *eragiketak*
```

Adibidez:

```
izena = input ("Esadazu zure izena:")
if izena != "":
 print ("Kaixo", izena)
```

```
else:
 print ("Ez duzu ezer sartu!")
```

Horrelako zerbait ikus liteke, erabiltzaileak sartzen duenaren arabera:

```
Esadazu zure izena: Ada
Kaixo Ada
```

Baina erabiltzaileak ezer idatzi gabe enter tekla sakatu besterik ez badu egiten, hurrengoa ikusiko genuke:

```
Esadazu zure izena:
Ez duzu ezer sartu!
```

1.1 Ariketa

Idatzi erabiltzaileari testu bat eskatzen dion programa bat. Testua `"agurra"` bada, agur bat adierazi behar duzu; bestela, `"Ez dut ulertzen"` dioen mezu bat erakutsi behar duzu.

```
testua = input("Sartu testua: ")
if testua == "agurra":
 print("Kaixo!")
```

```
else:
 print("Ez dut ulertzen.")
```

Emaitza:

```
Sartu testua: ez dakit
Ez dut ulertzen.
```

8.3 if elif else

Bada beste egitura bat hainbat baldintza egiaztatu behar ditugunean. Horretarako, `if-elif-else` egitura dago:

```
if *baldintza1*:
 *eragiketak*
elif: *baldintza2*:
 *eragiketak*
elif: *baldintza3*
 *eragiketak*
else:
 *eragiketak*
```

Demagun hizkuntza desberdinetan agurtzeko gai den programa bat nahi dugula. Honelako programa bat sor dezakegu:

```
hizkuntza = input ("Zer hizkuntzan hitz egiten duzu?")

if hizkuntza == "Espainera":

 print ("Kaixo")

elif hizkuntza == "Ingelera":

 print ("Hello")

elif hizkuntza == "Frantsesa":

 print ("Salut")

else:

 print ("Ez dut hizkuntza hori ezagutzen")
```

Behar adina elif izan ditzakegu.

1.2 Ariketa

Idatzi erabiltzaileari testu bat eskatzen dion programa bat. Testua `"goiza"` bada, `"Egun on"` mezua erakutsi behar duzu, testua `"arratsalde"` bada, `"Arratsalde on"` mezua erakutsi behar duzu, testua `"gau"` bada `"Gabon"` mezua erakutsi behar duzu eta, bestela `"Ez dut ulertzen"`.

```
testua = input("Idatzi testua: ")
if testua == "goiza":
 print("Egun on.")
elif testua == "arratsalde":
 print("Arratsalde on.")
elif testua == "gau":
 print("Gabon.")
else:
 print("Ez dut ulertzen.")
```

Emaitza:

```
Idatzi testu bat: arratsalde
```

```
Arratsalde on.
```

1.3 Ariketa 

Sortu programa bat erabiltzaileari bi balio oso eskatzeko, alderatzeko eta pantaila bidez erakusteko bat bestea baino handiagoa den edo berdinak diren.

```
balio1 = int(input("Sartu zenbaki bat: "))

balio2 = int(input("Sartu beste zenbaki bat:
"))

if balio1 > balio2:

 print(balio1, " ", balio2, " baino handiagoa
da")

elif balio1 < balio2:

 print(balio1, " ", balio2, " baino txikiagoa
da")

else:

 print(balio1, " ", balio2, "berdinak dira")
```

Emaitza:

```
Sartu zenbaki bat: 5
Sartu beste zenbaki bat: 10
5 10 baino txikiagoa da
```

8.4 Proposatutako ariketak

1.0 Ariketa

Sortu programa bat erabiltzaileari bi zenbaki oso eskatuko dizkiona. Gero pantaila bidez erakutsi ea lehenengo zenbakia bigarrenaren multiploa den. Zenbaki bat bestearen multiploa den jakiteko, moduluaren eragiketa egin behar duzu (%) haien artean: 0 bada, multiploa izango da.

```
balio1 = input("Sartu zenbaki bat: ")

balio2 = input("Sartu beste zenbaki bat: ")

ondarra = int(balio1) % int(balio2)

if ondarra == 0:

 print(balio1, " zenbakia ", balio2, "
zenbakiaren multiploa da.") else:

 print(balio1, " zenbakia ez da ", balio2, "
zenbakiaren multiploa.")
```

Emaitza:

```
Sartu zenbaki bat: 40
```

```
Sartu beste zenbaki bat: 4

40 zenbakia 4 zenbakiaren multiploa da.
```

1.1 Ariketa

Sortu erabiltzaileari zenbaki oso bat eskatzen dion programa bat eta ondorengoa egin: lehenik eta behin pantailan erakutsi behar du zenbakia negatiboa edo positiboa den. Gero, zenbakia positiboa bada, negatibo bihurtu behar du, eta negatiboa bada, positibo.

```
zenbakia = input("Sartu zenbaki bat: ")
zenbakia = int(zenbakia)
if zenbakia >= 0:
 print(zenbakia, " positiboa da")
else:
 print(zenbakia, " negatiboa da")
zenbakia = -zenbakia
print("Bihurketa: ", zenbakia)
```

Emaitza:

```
Sartu zenbaki bat: -6
-6 negatiboa da
Bihurketa: 6
```

1.2 Ariketa

Idatzi urteko hilabete baten izena eskatzen duen programa bat eta erakutsi zenbat egun dituen. Hilabete ezezagun bat sartuz gero, `"Ez dakigu"` mezua erakutsi.

```
hilabetea = input("Sartu urteko hilabete bat:
")
if hilabetea == "Urtarrila":
 print("31 egun ditu")
elif hilabetea == "Otsaila":
 print("28/29 egun ditu")
elif hilabetea == "Martxoa":
 print("31 egun ditu")
elif hilabetea == "Apirila":
 print("30 egun ditu")
```

```
elif hilabetea == "Maiatza":
 print("31 egun ditu")
elif hilabetea == "Ekaina":
 print("30 egun ditu")
elif hilabetea == "Uztaila":
 print("31 egun ditu")
elif hilabetea == "Abuztua":
 print("31 egun ditu")
elif hilabetea == "Iraila":
 print("30 egun ditu")
elif hilabetea == "Urria":
 print("31 egun ditu")
elif hilabetea == "Azaroa":
 print("30 egun ditu")
elif hilabetea == "Abendua":
 print("31 egun ditu")
```

```
else:
 print("Hilabete ezezaguna")
```

Emaitza:

```
Sartu urteko hilabete bat: Ekaina
30 egun ditu
```

Hobespena: saiatu programa motzago egiten or eragilea erabiliz.

1.3 Ariketa

Sortu programa bat erabiltzaileari zenbaki oso bat eskatzen diona eta pantailan zenbaki hori bikoitia eta positiboa den ala ez erakusten duena. Bestela, negatiboa edo bakoitia den adierazi behar duzu.

```
zenbaki = input("Sartu zenbaki bat: ")
zenbaki = int(zenbaki)
if zenbaki >= 0 and zenbaki % 2 == 0:
 print(zenbaki, " bikotia eta positiboa da")
elif zenbaki < 0 and zenbaki % 2 != 0:
 print(zenbaki, " bakoitia eta negatiboa da")
```

```
elif zenbaki < 0:
 print(zenbaki, " negatiboa da")
else:
 print(zenbaki, " bakoitia da")
```

Emaitza:

```
Sartu zenbaki bat: -9
- 9 bakoitia eta negatiboa da
```

1.4 Ariketa

Sortu programa bat erabiltzaileari kilotan duen pisua eta zentimetrotan duen altuera eskatzeko eta GMI kalkulatzeko (pisua/(altuera ber 2)); emaitza erakutsi behar du eta gero mezu bat erakutsi:

- GMI 16 baino txikiagoa bada, `"Beharrezkoa da gehiago jatea"` mezua agertuko da.
- GMI (>=) 16 eta 25 (<) artean badago, `"Ondo zaude"` mezua agertuko da.
- GMI (>=) 25 eta 30 artean badago (<), `"Gorputz gordina duzu"` mezua agertuko da.

- GMI 30 baino handiagoa bada, mezu hau agertuko da: "Obesitate arazoa duzu".

```
pisua = input("Sartu zure pisua: ")
altuera = input("Sartu zure altuera: ")
pisua = int(pisua)
altuera = int(altuera)
emaitza = pisua / (altuera * altuera)
gmi = (emaitza * 10000)
print("Zure gmi: ", gmi)
if gmi < 16 :
 print("Beharrezkoa da gehiago jatea")
elif gmi >= 16 and gmi < 25:
 print("Ondo zaude")
elif gmi >= 25 and gmi < 30:
 print("Gorputz gordina duzu")
else:
```

```
    print("Obesitate arazoa daukazu")
```

Emaitza:

```
Sartu zure pisua: 70
Sartu zure altuera: 172
Zure gmi: 23.66143861546782
Ondo zaude
```

1.5 Ariketa

Sortu erabiltzaileari jokalari dortsal bat eskatzen dion programa bat eta ondorengoa egin: egiaztatu zenbaki hori 0 eta 99 artean dagoela. Ez badago, programa errore mezu batekin amaitu behar da. Zenbakia 0 eta 99 artekoa bada, programak testu bat erakutsi behar du dortsal bakoitzari dagokion posizioarekin:

- Erabiltzaileak 1 tekleatu badu, testua `"Atezaina"` izango da
- Erabiltzaileak 2 eta 5 artean tekleatu badu, testua `"Defentsa"` izango da.

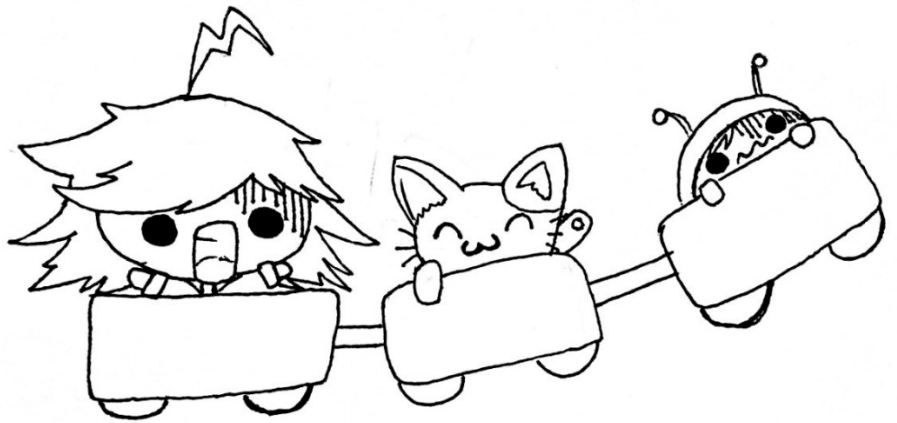
- Erabiltzaileak 6 eta 8 artean edo 11 tekleatu badu, testua `"Erdilaria"` izango da. Erabiltzaileak 9 tekleatu badu, testua `"Aurrelaria"` izango da.
- Beste edozein aukeratarako, testua `"Edozein"` izango da.

```
zenbakia = input("Sartu zenbakia: ")

zenbakia = int(zenbakia)

if zenbakia >= 0 and zenbakia <= 99:

 if zenbakia == 1:

 print("Atezaina")

 elif zenbakia >= 1 and zenbakia <= 5:

 print("Defentsa")

 elif zenbakia >= 6 and zenbakia <= 8 or
zenbakia == 11:  print("Erdilari")

 elif zenbakia == 9:

 print("Aurrelari")

 else:

 print("Edozein")
```

```
  else:
   print("Errorea, zenbakia ez dago 0 eta 99
  artean") Emaitza:
```

```
Dortsala sartu: 11

Erdilari
```

9 Begiztak



Hasieran esan bezala, ordenagailuak oso tentelak dira. Esaten zaiena egiten dute. Baina, aldiz, gaitasun izugarriak eta pazientzia amaigabea daukate. Ez zaie batere inporta behar adina aldiz edozer gauza egitea.

Ordenagailu baten zeregin ohikoenetako bat agindu bat errepikatzea da. Hori begizta egituren bidez lor daitekeen zerbait da. Begizta ekintza errepikatu bat da. Oro har, begizta batek

baldintza bat betetzen du exekutatzeko: baldintza horiek betetzen badira, orduan begizta horrek dituen aginduak exekutatuko dira.

Begiztak errusiar mendiak bezalakoak dira, non zirkuitu itxi batean buelta batzuk ematen dituzun. Jarraian, begizta mota desberdinak ikusiko ditugu.

9.1 while begizta

Baldintza bat betetzen den bitartean, `while` begizta exekutatu egiten da. Egitura oso sinplea du:

```
while *baldintza*:
 *eragiketak*
```

Adibidez, begizta bat exekutatuko dugu aldagai baten balioa 0 baino handiagoa den bitartean.

```
kontagailua = 4
while kontagailua > 0:
 print ("Begiztaren barruan nago")
 kontagailua = kontagailua - 1
```

Pantailako emaitza hau izango litzateke:

```
Begiztaren barruan nago
Begiztaren barruan nago
Begiztaren barruan nago
Begiztaren barruan nago
```

Oharra 

Kontuz! Konturatu al zara begiztaren barruan kontagailuari balio bat kentzen ari garela? Kontuz ez bagabiltza eta hori egitea ahazten badugu, kontagailuaren balioa ez litzateke inoiz aldatuko eta **begizta amaigabea** sortuko genuke. Programa ez litzateke inoiz amaituko eta betiko trabatuta geratuko litzateke!

2.0 Ariketa 

Sortu 0n hasitako kontagailu aldagai bat definituko duen programa. Ondoren, idatzi while begizta bat. Kontagailua 5 baino txikiagoa den bitartean, erakutsi mezu hau: `"Begizta barruan nago"`.

```
kontagailua = 0
while kontagailua < 5:
 print("Begizta barruan nago")
```

```
    kontagailua = kontagailua + 1
```

Emaitza:

```
Begizta barruan nago
Begizta barruan nago
Begizta barruan nago
Begizta barruan nago
Begizta barruan nago
```

Ikus dezagun beste adibide bat. Hurrengo programak datu bat eskatzen dio erabiltzaileari begizta batean. Programa ez da begiztatik aterako erabiltzaileak datu hutsa sartzen duen bitartean:

```
izena = ""
while izena == "":
 izena = input ("Nola duzu izena?")
print ("Kaixo", izena)
```

2.1 Ariketa

Idatzi erabiltzaileari bi zenbaki eskatzen dizkion programa bat. Lehenak bigarrena baino txikiagoa izan behar du. Begiztak bi

zenbaki horien arteko tartean dauden zenbakiak erakutsi behar ditu.

```
min = input("Sartu minimo bat: ")
min = int(min)
max = input("Sartu maximo bat: ")
max = int(max)
while min < max:
 print(min)
 min = min + 1
```

Emaitza:

```
Sartu minimo bat: 3
Sartu maximo bat: 8
3
4
5
6
```

```
7
```

9.2 For begizta

`for` begizta ekintza bat errepikatzeko ere erabiltzen dugu. Baldintza bat baino gehiago, kontagailu moduko bat erabiltzen du:

```
for *aldagaia* in *tartea*:
 *eragiketak*
```

Adibidez, honako begizta honek `"Kaixo"` mezua 4 aldiz erakutsiko du:

```
for i in range (4):
 print ("Kaixo")
```

Hona hemen emaitza:

```
Kaixo
Kaixo
Kaixo
Kaixo
```

`for` begiztetan oso interesgarria den zerbait `i` aldagaiak dira, nahi dugun izena izan lezakeenak, begiztaren buelta bakoitzari dagokion balioa izango duena. Hobeto ikusteko, nahikoa da aurreko programa pixka bat aldatzea:

```
for i in range (4):
 print ("Kaixo", i)
```

Eta emaitzan ikusiko dugu nola aldatzen den `i` hori:

```
Kaixo 0
Kaixo 1
Kaixo 2
Kaixo 3
```

Hori baliagarria izan daiteke programa askotan.

2.2 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat, 0tik 10era biderkatzeko taula erakusten duena.

```
balioa = input("Sartu zenbaki bat: ")
balioa = int(balioa)
```

```
for i in range(11):
 print(balioa,"x",i,"=",(balioa*i))
 # Beste Aukerak:
 # print("%d x %d = %d" % (balioa, i, balioa *
i))
```

Emaitza:

```
Sartu zenbaki bat: 3
3 x 0 = 0
3 x 1 = 3
3 x 2 = 6
...etab.
```

2.3 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat. 0 edo txikiagoa bada, zerbait handiagoa sartzeko adierazi behar duzu, eta bestela, `"Python ondo dago!"` mezua erakutsi behar duzu pantailan, zenbakiak adierazten duen adina aldiz:

```
zenbakia = input("Sartu zenbaki bat: ")

zenbakia = int(zenbakia)

if zenbakia <= 0:

 print("0 baino handiagoa den zenbaki bat
sartu behar duzu") else:

 for i in range(0, zenbakia):

 print("Python ondo dago!")
```

Emaitza:

```
Sartu zenbaki bat: 3

Python ondo dago!

Python ondo dago!

Python ondo dago!
```

9.2.1 Tartea aldatu

Lehenetsita, range (4) 0tik 3ra bitarteko zerrenda itzultzen ari da, hau da: 0, 1, 2, 3. Lau elementu dira guztira eta, beraz, begiztak lau buelta emango ditu.

Jakina, edozein maila mota sor daiteke. Ezer adierazten ez bada, tartea 0an hasten da. Baina bi zenbakiren arteko tartea adieraz daiteke:

```
range(0, 4) # 0, 1, 2, 3
range(2, 6) # 2, 3, 4, 5
```

Adibidez:

```
for i in range (5, 9):
 print ("Kaixo", i)
```

Hona hemen emaitza:

```
Kaixo 5
Kaixo 6
Kaixo 7
Kaixo 8
```

Hirugarren parametro bat ere adieraz daiteke, balio batetik bestera nola egiten den jauzi adierazteko. Adibidez, 1tik 11ra, salto bikoitza emanez:

```
range (1, 11, 2) # 1, 3, 5, 7, 9
```

Hurrengo adibidean, begizta zenbaki bikoitiekin egingo litzateke.

```
for i in range (0, 6, 2):
 print ("Kaixo", i)
```

Hona hemen emaitza:

```
Kaixo 0
Kaixo 2
Kaixo 4
```

9.2.2 Atzerantz

Begizta atzerantz ere ibil daiteke, jauzi negatiboa eginez:

```
print("Atzerako kontaketa hasten:")
for i in range (5, 0, -1):
 print (i)
```

Hona hemen emaitza:

```
5
4
```

```
3
2
1
```

2.4 Ariketa

Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat. Ondoren, erabiltzaileak sartu duen zenbakira arteko zenbaki bikoiti guztiak erakutsi behar ditu, 0tik hasieta Erabili for begizta eta egin salto binaka.

```
balioa = input("Sartu zenbaki bat: ")
balioa = int(balioa)
for i in range(0, balioa, 2):
 print(i)
```

Emaitza:

```
Sartu zenbaki bat: 5
0
2
```

```
4
```

9.3 Zerrenden gaineko begiztak

Begizta bereziki erabilgarriak dira zerrenda bateko elementu guztiak aztertu nahi ditugunean, erakusteko, prozesatzeko edo dena delakoa. Forma oso erraza da:

```
objektuak = ["izarra", "perretxikoa", "lorea"]

for objektua in objektuak:

 print (objektua)
```

Begiztaren buelta bakoitzean, objektu aldagaiak balio bat hartuko du objektuak izeneko zerrendatik, eta, beraz, emaitza hau izango da:

```
izarra

perretxikoa

lorea
```

9.4 Begiztatik irteten

Batzuetan, baliteke begizta batetik irten eta beste ezer prozesatzen ez jarraitzea. Demagun zerrenda baten barruan izen bat bilatzeko programa bat dugula:

```
izenak = ["Mia", "Jon", "Arya", "Ane", "Bug",
"Ada", "Lisp"]

for izena in izenak:

 if izena == "Ane":

 print ("Ane zerrendan dago")
```

Programa honek arazo bat dauka: nahiz eta `Ane` aurkitu, begiztak zerrendaren amaierara arte jarraituko du. Zerrenda hori oso handia bada, gure programa ez da eraginkorra izango! Hau da, denbora galtzen ibiliko da. Hasieran esan bezala, ordenagailuak oso tentelak dira. Gelditzeko esaten ez badiegu, aurrera jarraituko dute.

Zorionez, begiztetan break agindua erabil dezakegu, begizta bat-batean bukatzea lortuko duena:

```
izenak = ["Mia", "Jon", "Arya", "Ane", "Bug",
"Ada", "Lisp"]

for izena in izenak:
```

```
if izena == "Ane":
print ("Ane zerrendan dago")
break
```

2.5 Ariketa

Idatzi 3 zenbakidun zerrenda bat definitzen duen programa bat. Ondoren, sortu for begizta bat, zerrenda zeharkatu eta zenbaki bakoitza bezain beste aldiz pantailatik `"Python"` hitza ateratzen duena.

```
errepikatu = [3, 6, 2]
for aldiz in errepikatu:
 for i in range(aldiz):
 print("Python")
```

Emaitza:

```
Python
Python
Python
Python
```

```
...
```

9.5 Noiz erabili while ala for?

Biekin gauza bera egin zenezakeen arren, bakoitzak erabilera berezia du.

`for` begizta buelta kopuru zehatza exekutatu nahi denean erabiltzen da, ez gehiago, ez gutxiago. Edo, aurrerago ikusiko dugun bezala, datu-egiturak hasieratik amaierarainoko zerrendek kudeatu nahi direnean.

`while` begiztak ordez, baldintzak oso zehatzak ez direnean erabil daitezke. Adibidez, erabiltzaileak datu bat sartzea nahi badugu, begizta batean sar dezakegu. Begizta ez da amaituko erabiltzaileak datu on bat sartu arte (hori litzateke baldintza).

9.6 Proposatutako ariketak

2.0 Ariketa

Idatzi programa bat while begizta batekin, erabiltzaileari izen bat eskatzeko (adibidez, `"Ada"`) eta `"Kaixo Ada"` izen hori agurtzeko. `"irten"` testua sartuz gero, begizta amaitu egin behar da.

```
izena = ""
while izena != "irten":
 izena = input("Sartu izen bat: ")
 print("Kaixo", izena)
print("Amaiera.")
```

Emaitza:

```
Sartu izen bat: Ada
Kaixo Ada
Sartu izen bat: Neko
Kaixo Neko
```

```
Sartu izen bat: irten

Kaixo irten

Amaiera.
```

2.1 Ariketa

Idatzi programa bat, erabiltzaileari zenbaki bat eskatzen diona eta zenbakia 0 ez den bitartean amaitzen ez duen `while` begizta duena. Zenbakia sartu ondoren, agur bat erakutsi behar da zenbakiak adierazten duen adina aldiz. Zenbakia 0 baino txikiagoa bada, begizta break batekin amaitu behar da;

```
balioa = ""
while balioa != 0:
 balioa = input("Sartu zenbaki bat: ")
 balioa = int(balioa)
 if balioa < 0:
   break
for i in range(balioa):
 print("Kaixo")
```

Emaitza:

```
Sartu zenbaki bat: 3
Kaixo
Kaixo
Kaixo
Sartu zenbaki bat: -1
```

2.2 Ariketa

Sortu erabiltzaileari balio oso bat eskatzen dion programa bat, egiaztatu 0 baino handiagoa dela eta, gainera, bikoitia dela. Hala bada, bistaratu pantailan `*` karakterea zenbakiaren balioa adina aldiz. Erabili `print("*")`. Adibidez, 8 sartuz gero, hau erakutsiko du:

```
*******
```

Sartutako balioak baldintzak betetzen ez baditu, erabiltzailea ohartarazteko mezu bat erakutsi behar diozu eta programa amaitu.

```
zenbakia = input("Sartu zenbaki bat: ")
zenbakia = int(zenbakia)
```

```
if zenbakia <= 0 or zenbakia % 2 != 0:

 print("0 baino handiagoa den zenbaki bikoitia
sartu behar duzu") else:

 izarrak = ""

 while zenbakia > 0:

   izarrak = izarrak + "*"

   zenbakia = zenbakia - 1

   print(izarrak)
```

Emaitza:

```
Sartu zenbaki bat: 6

*****
```

2.3 Ariketa

Sortu aurrekoaren antzeko programa bat, zenbaki bikoiti eta positiboak bakarrik onartu behar dituena. Sartutako zenbakiarekin `*` – testua pantailan agertu behar da, `*`rekin amaituta. Adibidez:

8 zenbakia sartzen bada, hauxe erakutsiko da:

```
*-*-*-*
```

Adibidez, 4 zenbakia sartzen bada:

```
*-*-*
```

Programa:

```
zenbakia = input("Sartu zenbaki bat: ")
zenbakia = int(zenbakia)
if zenbakia <= 0 or zenbakia % 2 != 0:
 print("0 baino handiagoa den zenbaki bikoitia sartu behar duzu") else:
 sekuentzia = ""
 zenbakia = zenbakia / 2
 while zenbakia > 0:
   sekuentzia = sekuentzia + "*-"
   zenbakia = zenbakia - 1
   sekuentzia = sekuentzia + "*"
   print(sekuentzia)
```

Emaitza:

```
Sartu zenbaki bat: 10
*-*-*-*-*-*
```

2.4 Ariketa

Erabiltzaileari zenbaki oso bat eskatzen dion programa bat sortu, eta balio hori erabiliz, lauki bat marraztu behar du pantailan `*` karakterea erabiliz.

Adibidez, 4 sartuz gero, hau erakutsiko da:

```
****
****
****
****
```

```
zenbakia = input("Sartu zenbaki bat: ")
zenbakia = int(zenbakia)
if zenbakia <= 0:
```

```
 print("0 baino handiagoa den zenbaki bat
sartu behar duzu") else:
 izarrak = "\n"
 for i in range(zenbakia):
   for j in range(zenbakia):
        izarrak = izarrak + "*"
        izarrak = izarrak + "\n"
        print(izarrak)
```

Emaitza:

```
Sartu zenbaki bat: 2
**
**
```

2.5 Ariketa

Sortu programa bat erabiltzaileari zenbaki oso bat eskatu eta haren faktoria kalkulatzeko. Adibidez, 5 zenbakiaren faktoria 5 x 4 x 3 x 2 x 1 = 120 izango litzateke.

```
zenbakia = input("Sartu zenbaki bat: ")
```

```
zenbakia = int(zenbakia)

if zenbakia <= 0:

 print("0 baino handiagoa den zenbaki bat
sartu behar duzu") else:

 faktoriala = zenbakia

 while zenbakia > 1:

   zenbakia = zenbakia - 1

   faktoriala = faktoriala * zenbakia

   print("Emaitza: ", faktoriala)
```

Emaitza:

```
Sartu zenbaki bat: 4

Emaitza: 24
```

2.6 Ariketa

Sortu erabiltzaileari zenbaki oso bat eskatzen dion programa bat eta egiaztatu zenbaki hori lehena (primoa) den ala ez, hau da, bakarrik bere buruarekin edo 1egatik zatigarria dena. Adibidez: 2, 3, 5, 7, 11, 13, 17,...

```
zenbakia = input("Sartu zenbaki bat: ")
zenbakia = int(zenbakia)
if zenbakia <= 0:
 print("0 baino handiagoa den zenbakia sartu behar duzu")
else:
 zatigarria = False
 hasierakoa = zenbakia
 zenbakia = zenbakia - 1
 while zenbakia > 1 and not zatigarria:
   if hasierakoa % zenbakia == 0:
        zatigarria = True
        zenbakia = zenbakia - 1
   if not zatigarria:
        print(hasierakoa, " zenbakia lehena da.")
   else:
```

```
        print(hasierakoa, " lehen ez da.")
```

Emaitza:

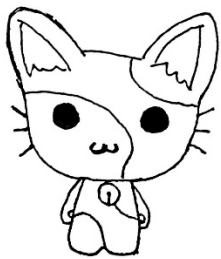
```
Sartu zenbaki bat: 5
5 zenbakia lehena da.
```

2.7 Ariketa

Sortu 0tik 10era biderketa taula guztiak erakusten dituen programa bat.

```
for i in range(11):
 for j in range(11):
 print(i,"x",j,"=",i*j)
# Honek ere berdin egingo du
for i in range(11):
 for j in range(11):
 print(" %d x %d = %d" % (i, j, i*j))
```

10 Datu egiturak


Orain arte datu sinpleak erabili ditugu. Zenbaki bat, testu bat eta abar duten aldagaiekin jolasten aritu gara. Baina badira datu konplexuagoak sortzeko aukera ematen diguten beste datu mota batzuk ere: datu bakar bat baino zerbait gehiago eduki dezakete.

Ordenagailu programek oso gauza zailak egin ditzakete, baina, funtsean, datuak prozesatzen dituzte. Eta askotan, datu horiek sekuentzia luzeetan datoz. Jarraian, datu mota horietako batzuk ikusiko ditugu.

10.1 Zerrendak

Zerrendak zenbakiz indexatutako datu-multzo bat dira. Hori definizio oso formala da, baina izenak berak esaten dizu zer diren: zerrenda bat! Datu motei buruzko kapituluan zerrendak aurkeztu genituen eta nola sortzen diren ikusi genuen:

```
hizkuntzak = ["Ingelesa", "Gaztelania",
"Frantsesa"]
```

Gogoratu, zenbaki edo indize baten bidez eskura ditzakezula elementuak:

```
hizkuntzak = ["Ingelesa", "Gaztelania",
"Frantsesa"]

print (hizkuntzak [2]) #Frantsesa
```

Zerrenda honela errepresentatu daiteke:

```
0 1 2

"Ingelesa" "Gaztelania" "Frantsesa"
```

3.0 ariketa

Definitu izenen zerrenda bat eta erakutsi pantaila bidez:

```
izenak = ["Ada", "Bug", "Neko"]
```

```
print (izenak) # ["Ada", "Bug", "Neko"]
```

Emaitza:

```
["Ada", "Bug", "Neko"]
```

10.1.1 Zerrenden luzera

Zerrenda baten luzera jakiteko, `len()` funtzioa erabil daiteke:

```
hizkuntzak = ["Ingelesa", "Gaztelania",
"Frantsesa"]

print (len (hizkuntzak)) # 3
```

3.1 Ariketa

Sortu programa bat 5 zenbaki hamartarrekin definitzeko. Gero, sortu begizta bat zenbaki guztien batez bestekoa kalkulatzeko.

```
zenbakiak = [3.4, 2.7, 4.3, 6.6, 8.3]

batuketa = 0.0

for zenbakia in zenbakiak:

 batuketa = batuketa + zenbakia

batazbestekoa = batuketa / len(zenbakiak)
```

```
print("Batazbestekoa da: ", batazbestekoa)
```

Emaitza:

```
Batazbestekoa da: 5.0600000000000005
```

10.1.2 Zerrendatik zatiak atera

Zenbakizko indizea erabiliz, zerrenda baten zatiak atera daitezke, zerrenda horren azpi-zerrenda bat sortuz. Adibidez, "zerrendako lehen hiru balioak nahi ditut", "4. zenbakitik 6.era" edo "azken biak nahi ditut". Horretarako, nahikoa da indize-tarte bat adieraztea:

```
zenbakiak = [1, 2, 3, 4, 5, 6, 7, 8, 9]

zenbakiak [0:4] # [1, 2, 3, 4]

zenbakiak [5:8] # [6, 7, 8]
```

Lehen elementuak ere atera daitezke:

```
zenbakiak = [1, 2, 3, 4, 5, 6, 7, 8, 9]

zenbakiak[: 6] # [1, 2, 3, 4, 5, 6]
```

Edo azken balioak

```
zenbakiak = [1, 2, 3, 4, 5, 6, 7, 8, 9]

zenbakiak[-4:] # [6, 7, 8, 9]
```

Edo, besterik gabe, azkena:

```
zenbakiak = [1, 2, 3, 4, 5, 6, 7, 8, 9]

zenbakiak[-1] # [9]
```

10.1.3 Elementuak gehitu eta ezabatu

Zerrenda bati elementu bat gehitu nahi badiogu, nahikoa da `append` funtzioa erabiltzea:

```
hizkuntzak = ["Ingelesa", "Gaztelania",
"Frantsesa"]

hizkuntzak.append ("Italiera")

print (hizkuntzak) # ["Ingelesa",
"Gaztelania", "Frantsesa", "Italiera"]
```

Eta elementu bat zerrendatik kendu nahi badugu, `del` agindua erabil daiteke:

```
hizkuntzak = ["Ingelesa", "Gaztelania",
"Frantsesa"]

del hizkuntzak[1]

print (hizkuntzak) # ["Ingelesa", "Frantsesa"]
```

Azken elementua kentzeko, `pop` funtzioa ere erabil daiteke. Honek elementua kentzen du eta itzultzen du:

```
hizkuntzak = ["Ingelesa", "Gaztelania",
"Frantsesa"]

azkena = hizkuntzak.pop()

print(hizkuntzak) # ["Ingelesa", "Gaztelania"]

print(azkena) # "Frantsesa"
```

Eta zerrendako elementu baten balioa ere alda daiteke:

```
hizkuntzak = ["Ingelesa", "Gaztelania",
"Frantsesa"]

hizkuntzak[2] = "Italiera"
```

Eta gogoratu, zerrendan zehar joateko, begizta hau erabil dezakegu:

```
hizkuntzak = ["Ingelesa", "Gaztelania",
"Frantsesa"]

for hizkuntza in hizkuntzak:

 print (hizkuntza)
```

Indizea erabiliz ere zeharkatu daiteke zerrenda. Horretarako, `range` funtzioa erabili beharko dugu, eta zerrendaren luzera eman beharko diogu, len erabiliz:

```
hizkuntzak = ["Ingelesa", "Gaztelania",
"Frantsesa"]

for i in range (len (hizkuntzak)):

 print (hizkuntzak [i])
```

Dena den, indizea begiztaren barruan behar ez bada, hobe da zerrenda indizerik gabe egitea, aurreko adibidean egiten den bezala.

Beste lengoai batzuetan, zerrendei *array* esaten zaie. Testuan zehar hitz desberdinak erabiliko ditugu.

3.2 Ariketa

Definitu izenen zerrenda bat eta erakutsi pantaila bidez. Gehitu elementu bat eta erakutsi zerrenda osoa pantailan. Ondoren, zerrendatik elementu bat ezabatu eta berriro pantailan erakutsi zerrenda.

```
izenak = ["Ada", "Bug", "Neko"]

print(izenak) # ["Ada", "Bug", "Neko"]
```

```
izenak.append("Miranda")

print(izenak) # ["Ada", "Bug", "Neko",
"Miranda"]

del izenak[1]

print(izenak) # ["Ada", "Neko", "Miranda"]
```

Emaitza:

```
["Ada", "Bug", "Neko"]

["Ada", "Bug", "Neko", "Miranda"]

["Ada", "Neko", "Miranda"]
```

10.2 Hiztegiak

Hiztegiak datu-multzoak dira, non elementu bakoitzak gako bat eta balio bat ditu. Beste era batera esanda, zerrenda bat bezalakoak dira, baina 0, 1, 2... zenbaki-indizea izan beharrean, zuk nahi duzun balioa dute.

Adibidez, hiztegi bat defini dezakegu, hainbat pertsonaren adinak biltzen dituena, non izena gakoa den eta adina, balioa:

```
adinak = {"Ada": 14, "Bug": 10, "Neko": 2}
```

```
print(adinak ["Ada"]) # 14
```

Hiztegia honela adieraz daiteke:

```
"Ada" "Bug" "Neko"
14 10 2
```

3.3 Ariketa

Defini ezazu telefonoak izeneko hiztegi bat, non lagun pare baten telefonoak gordeko dituzun. Gakoa lagunaren izena izango da, eta balioa, berriz, telefono zenbakia.

```
telefonoak = {"Ada": 666555333, "Bug":
111000111 }

print(telefonoak)

for izena in telefonoak.keys():

 print(izena, telefonoak[izena])
```

Emaitza:

```
{'Ada': 666555333, 'Bug': 111000111}
Ada 666555333
```

```
Bug 111000111
```

Elementu berriak gehitzeko, balio berri bat esleitu daiteke. Ezabatzeko, `del` agindua erabiltzen da:

```
adinak = {"Ada": 14, "Bug": 10, "Neko": 2}

print (adinak) # {'Ada': 14, 'Bug': 10,
'Neko': 2}

adinak["Miranda"] = 23;

print(adinak) # {'Ada': 14, 'Bug': 10, 'Neko':
2, 'Miranda': 23} del adinak["Bug"]

print(adinak) # {'Ada': 14, 'Neko': 2,
'Miranda': 23}

for izena in adinak.keys ():

 print (izena, adinak[izena])
```

3.4 Ariketa

Erabili aurreko ataleko telefonoen hiztegia. Eskatu erabiltzaileari datuak hiztegian beste sarrera bat sartzeko: `izena` eta `telefonoa` eskatu beharko dituzu, eta gero hiztegira gehitu.

Amaieran, hiztegiko elementu guztiak erakutsi.

```
telefonoak = {"Ada": 666555333, "Bug":
111000111}

izena = input("Sartu izena: ")

telefonoa = input("Sartu telefono zenbakia: ")

telefonoak[izena] = int(telefonoa)

for izena in telefonoak.keys():

 print(izena, telefonoak[izena])
```

Emaitza:

```
Sartu izen bat: Neko

Sartu zenbaki bat: 333222000

Ada 666555333

Bug 111000111

Neko 333222000
```

Hiztegiko balioak ezabatu ditzakegu, honako funtzio honekin:

```
adinak = {"Ada": 14, "Bug": 10, "Neko": 2}

print (adinak) # {'Ada': 14, 'Bug': 10,
'Neko': 2}
```

```
del adinak["Bug"]
print (adinak) # {'Ada': 14, 'Neko': 2}
```

Eta hiztegiko balio guztiak erakutsi nahi baditugu? Ez dago arazorik, baina hiztegiaren gako guztiak itzuliko dizkigun funtzio bat erabili beharko dugu: `keys()`. Honela litzateke:

```
adinak = {"Ada": 14, "Bug": 10, "Neko": 2}
for izena in adinak.keys ():
 print (izena, adinak [izena])
```

Emaitza pantailan:

```
Ada 14
Bug 10
Neko 2
```

3.5 Ariketa

Erabili aurreko ataleko telefonoen hiztegia. Eskatu erabiltzaileari izen bat, eta gero kendu elementu hori hiztegitik. Amaieran, hiztegiko elementu guztiak erakutsi.

```
telefonoak = {"Ada": 666555333, "Bug":
111000111 }

izena = input("Sartu izena: ")

del telefonoak[izena]

for izena in telefonoak.keys():

 print(izena, telefonoak[izena])
```

Emaitza:

```
Sartu izen bat: Bug

Ada 66555333
```

Oharra 

Beste hizkuntza batzuetan, hiztegiei *hash*, *hashtable* edo "mapak" esaten zaie.

10.3 Datu kabiatuen egiturak

Oinarrizko egiturek, hala nola zerrendek eta hiztegiek, zerrendak eta hiztegiak eduki ditzakete kabiatuta.

Datu egitura habiatuak sor daitezke, behar bezain konplexuak. Demagun lagun baten datuak hiztegi honekin irudikatu nahi dituzula:

```
laguna = {"izena": "Neko", "adina": 2}
```

Eta hainbat lagun izango dituen datu egitura bat sortu nahi baduzu? Kasu horretan, hiztegien zerrenda egin dezakezu:

```
lagunak = [
 {"izena": "Neko", "adina": 2},
 {"izena": "Bug", "adina": 10},
 {"izena": "Ada", "adina": 14}
]
print (lagunak [1]) # {"izena": "Bug",
"adina": 10}
print (lagunak [2] ["izena"]) #Ada
```

3.6 Ariketa

Idatzi bezeroak izeneko hiztegi-zerrenda bat definitzen duen programa bat. Bezeroaren gakoak hauek lirateke: `izena`, `email` eta `adina`. Programak zerrenda egin behar du eta bezero bakoitzaren izena eta adina erakutsi.

```
bezeroak = [
 {
 "izena": "Juan",
 "email": "jj@terra.com",
 "adina": 39
 },
 {
 "izena": "Pedro",
 "email": "pp@ozu.es",
 "adina": 42
 },
 {
 "izena": "Ana",
 "email": "ana@ole.com",
 "adina": 37
 }
```

```
]

for bezeroa in bezeroak:

 print(f"{bezeroa['izena']}
{bezeroa['adina']}")
```

Emaitza:

```
Juan 39

Pedro 42

Ana 37
```

10.3.1 Hiztegien hiztegia

Eta joko baten egoera, pertsonaiak eta objektuak bilduko dituen datu-egitura bat nahi baduzu? Hiztegi habiatu bat egin nezake, non gakoa pertsonaiaren izena den:

```
pantaila = {

 "Mario": {"bizitza": 10, "objektuak":
["perretxikoa", "izarra"]},

"Luigi": {"bizitza": 7, "objektuak": []},

 "Toad": {"bizitza": 15, "objektuak":
["perretxiko"]},
```

```
}

print (pantaila["Luigi"]["bizitza"]) # 7
```

Baina nola dakit zer egitura mota diseinatu behar dudan? Emango diozun erabileraren arabera. Batzuetan denak ibili beharko dituzu, beste batzuetan elementu jakin bat eskuratu beharko duzu... zure programak eskatzen duenaren arabera, egitura jakin bat diseinatu beharko duzu, zure programa azkarrago joan dadin. Orohar, egitura azkarrenak hiztegiak dira, batez ere elementu zehatzak atzitu nahi dugunean.

10.4 Testuak

Testu datu mota, katea edo string ere deitua, funtsezkoa da programetan. Horregatik, funtzio laguntzaile asko ditu datu mota horiek errazago erabiltzeko. Jarraian, testuetarako baliagarriak diren zenbait funtzio ikusiko ditugu, baina, aurretik, testuari buruzko zerbait azaltzea komeni da:

10.4.1 Testuak zerrendak dira

Izan ere, ordenagailuarentzat, testu bat zerrenda bat bezalakoa da. Letra zerrenda (edo katea) honela erabil dezakegu:

```
testua = "Kaixo naiz Ada"

testua[1] # "a"
```

Testua zerrenda bat balitz bezala ere ibil dezakegu:

```
testua = "Ada"
for karaktere in testua:
 print (karaktere)
```

Irteera hau izango litzateke:

```
A
d
a
```

Testu baten luzera ere jakin dezakegu, `len()` funtzioarekin:

```
testua = "Neko miauka"
len(testua) # 11
```

Baina zalantzarik izanez gero, interesgarriena da testutik nahi dugun zatia atera dezakegula, hasiera eta amaiera adierazita:

```
testua = "Python gora"
testua [0:6] # "Python"
testua [7:12] # "gora"
```

Lehen karaktereak ere atera daitezke:

```
testua = "Python gora"
testua[:6] # "Python"
```

Edo azken karaktereak:

```
testua = "Python gora"
testua[-4:] # "gora"
```

Edo, besterik gabe, azkena:

```
testua = "Python gora"
testua[-1] # "a" testua
```

10.4.2 Letra larriak/Letra xeheak

Zenbait funtzio ditugu testua letra larriz edo xehez idazteko:

```
testua = "Irakasle Ada"
testua.upper() #IRAKASLE ADA
testua.lower() #Irakasle Ada
```

`title()` izeneko funtzio bat ere badugu, hitz bakoitza testu baten barruan aldatzen duena, lehen letra larriz jarriz.

```
testua = "hau esaldi bat da"

texto.title() #Hau Esaldi bat da
```

10.4.3 split: testutik zerrendara

`split` funtzio interesgarria da testu bat zatika banatzeko eta zerrendan bihurtzeko:

```
testua = "zatoz nirekin, bizi nahi baduzu"

hitzak = testua.split() # ["zatoz", "nirekin",
"bai", "nahi duzu", "bizi"]
```

Besterik ezean, `split` operazioa testuaren hutsuneetan mozten da. Baina beste edozein bereizle adieraz daiteke, adibidez:

```
testua = "Ada; Neko; Bug"

izenak = testua.split(";") # [Ada "," Neko ","
Bug "]
```

10.4.4 Bilaketak

Batzuetan, testu baten barruan hitz bat bilatu behar dugu. Horretarako, `find` funtzioa erabil dezakegu. Hitza aurkituz gero, hitz hori zein posiziotan hasten den erakutsiko du. Aurkitzen ez badu, `-1` itzuliko du.

```
hitzak = "Irakaslerik onena Ada da,
zalantzarik gabe"

aurkitu = hitzak.find ("hobeto") # 3

aurkitu = hitzak.find ("Ada") # 22

EzAurkitua = hitzak.find ("xxx") # -1
```

Testu bat nolabait hasten den jakin nahi badugu, `startswith()` erabil daiteke.

```
hitzak = "Python lengoaia ona da"

hasi = hitzak.startswith ("Py") #True

hasi = hitzak.startswith ("es") #False
```

Testu bat modu batean amaitzen den jakin nahi badugu, berriz, `endswith()` erabil daiteke:

```
hitzak = "Python lengoaia ona da"

acaba = hitzak.endswith ("da") #True

acaba = hitzak.startswith ("da") #False
```

10.4.5 Soberakinak ezabatu

Testuak zuriuneekin edo ezabatu nahi ditugun beste karaktere batzuekin has edo buka daitezke, esaterako, hutsune edo lerro jauziekin. Testu batetik soberan dauden zati horiek kentzeko, honako funtzio hauek erabil daitezke.

`lstrip`ekin, testuaren hasieran espazioak ezabatzen dira:

```
testua = "Espazioak ditut"

garbia = testua.lstrip() # "Espazioak ditut"
```

`rstrip` delakoarekin, testuaren hasieran espazioak ezabatzen dira:

```
testua = "Espazioak ditut"

garbia = testua.rstrip() # "Espazioak ditut"
```

Eta `strip` funtzioarekin bi aldeetako espazioak kenduko ditugu:

```
testua = "Espazioak ditut"

garbia = testua.strip() # "Espazioak ditut"
```

Besterik ezean, espazioak kentzen dira, baina kendu nahi dugun edozein testu adieraz dezakegu:

```
testua = "--Testua-gidoiarekin---"
```

```
garbia = testua.lstrip ("-") # "Testua-
gidoiarekin---"
```

Fitxategi batetik edo kontsolatik testua irakurtzen dugunean ere, lerro jauziak kentzeko `rstrip` funtzioa erabil daiteke:

```
testua = "Honek lerro-jauzi bat du\n"
garbia = testua.rstrip("\n")
```

10.5 Proposatutako ariketak

3.0 Ariketa

Idatz ezazu 5 elementuko zerrenda bat hasieratzen duen programa bat:

- 5 zenbaki zerrenda (`0` balioarekin).
- 5 izenen zerrenda.
- 5 boolear balioko beste bat (`True` balioarekin hasitakoak).

```
izenak = ["Frodo", "Sam", "Merrin", "Pippin",
"Bilbo"]
logikoak = [True] * 5
```

```
zenbakiak = [0] * 5
print(izenak)
print(zenbakiak)
print(logikoak)
```

Emaitza:

```
[Frodo, Sam, Merrin, Pippin, Bilbo]
[0, 0, 0, 0, 0]
[True, True, True, True, True]
```

3.1 Ariketa

Idatzi 10 zenbakiko zerrenda bat definitzen duen programa bat. Ondoren, begizta bat sortu behar duzu, posizio bikoitietan 0 bat sartzeko.

```
zenbakiak = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in range(len(zenbakiak)):
 if i % 2 == 0:
    zenbakiak[i] = 0
```

```
print(zenbakiak)
```

Emaitza:

```
[0, 2, 0, 4, 0, 6, 0, 8, 0, 10]
```

3.2 Ariketa

Idatzi zerrenda bat kudeatzeko programa bat, erabiltzaileari 4 aukera dituen menu batekin:

1. Elementua sartu.
2. Ezabatu.
3. Erakutsi.
4. Irten.

Menua erabiltzaileak 4. aukera sartzen ez duen bitartean erakutsi behar da. 1 aukera aukeratuz gero, edozein balioko `append` egingo da, 2 aukeratuz gero, `pop` egingo da, eta 3 aukeratuz gero, zerrendako edukia erakutsiko da.

```
zenbakiak = []
aukera = -1
while aukera != "4":
```

```
print("Aukeratu")
print("1. Elementua sartu.")
print("2. Elementua kendu.")
print("3. Arraia erakutsi.")
print("4. Irten.")
aukera = input("Sartu aukera: ")
if aukera == "1":
  berria = int(input("Sartu zenbakia: "))
  zenbakiak.append(berria)
elif aukera == "2":
  zenbakiak.pop()
elif aukera == "3":
  print(zenbakiak)
elif aukera == "4":
  print("Agurra")
else:
```

```
    print("Aukera ezezaguna")
```

Emaitza:

```
Aukeratu
1. Elementua sartu.
2. Elementua ateratzea.
3. Erakutsi zerrenda.
4. Irten.
Aukera ezazu: 3
[]
```

3.3 Ariketa

Idatzi erabiltzaileari hitzak eskatu eta esaldi bat eraikitzeko hitzak kateatzen dituen programa bat, erabiltzaileak puntu bat idatzi arte (`.`). Azkenean programak sortutako esaldia erakutsi beharko du. Erabiltzaileak ez badu ezer idazten, ez da ezer kateatu behar.

```
izenburua = ""
hitza = ""
while hitza != ".":
```

```
    hitza = input("Idatzi hitz bat: ")
    if hitza != "." or hitza != "":
        izenburua = izenburua + " " + hitza
print("Sortutako esaldi:", izenburua)
```

Emaitza:

```
Idatzi hitz bat: Kaixo
Idatzi hitz bat: zer
Idatzi hitz bat: moduz?
Idatzi hitz bat:.
Kaixo, zer moduz? .
```

3.4 Ariketa

Idatzi programa bat erabiltzaileari bere izena, jaioterria eta jaioturtea eskatzeko. Ondoren, esaldi bat erakutsi behar duzu informazio guztiarekin, interpolazioa edo kate txantiloiak erabiliz.

```
izen = input("Idatzi zure izena: ")
jaiotegia = input("Idatzi jaiotze lekua: ")
```

```
urtea = input("Idatzi jaiotze urtea: ")

mezua = f"{izen} duzu zinen, {jaiotegia}n jaio
zinen {urtea} urtean." print(mezua)
```

Emaitza:

```
Idatzi zure izena: Ada

Idatzi zure jaioterria: Teverga

Idatzi zure jaioturtea: 2006

Ada duzu izena, Tevergan jaio zinen 2006
urtean.
```

3.5 Ariketa

Idatzi erabiltzaileari esaldi bat eskatzen dion programa bat. Gero esaldi horretako hitz bat eskatu behar du. Azkenkik, programak esaldi bera itzuliko du baina aukeratutako hitza letra larriz agertu beharko da:

```
esaldia = input("Idatzi esaldi bat: ")

hitza = input("Idatzi esalditik hitz bat: ")

posizioa = esaldia.index(hitza)

if posizioa != -1:
```

```
    hasiera = esaldia[0:posizioa]
    bukaera = esaldia[posizioa +
len(hitza):len(esaldia)]
    emaitza = f"{hasiera}{hitza.upper()}
{bukaera}"
    print(emaitza)
else:
    print(hitza, "ez dago esaldian.")
```

Emaitza:

```
Idatzi esaldi bat: Zein ona den Ada andereñoa
Idatzi esalditik hitz bat: ona
Zein ONA den Ada andereñoa
```

3.6 Ariketa

Sortu 5 izeneko zerrenda bat definitzen duen programa bat eta, ondoren, erabili begizta bat izenak banan banan erakusteko.

```
izenak = ["Frodo", "Merrin", "Sam", "Pip",
"Bilbo"]
```

```
for izena in izenak:
  print(izena)
# alda-eredu bat:
for i in range(len(izenak)):
 print(izenak[i])
```

Emaitza:

```
Frodo
Merrin
Sam
Pip
Bilbo
```

3.7 Ariketa

Sortu 10 zenbaki osoko zerrenda bat definitzen duen programa bat. Gero, beste begizta bat sortu, elementu bakoitza inkrementatu (`+1`) eta erakusteko.

```
zenbakiak = [3, 5, -4, 2, 1, 4, 0, 6, 9, 8, 3]
```

```
for zenbakia in zenbakiak:
 print(zenbakia)
for i in range(len(zenbakiak)):
 zenbakiak[i] = zenbakiak[i] + 1
for zenbakia in zenbakiak:
 print(zenbakia)
# Gehiketarako alternatiba:
# zenbakiakGehitu = zenbakiak.map( zenbakia =>
zenbakia + 1 )
```

Emaitza:

```
3
5
-4
2
1
4
```

```
0
6
9
8
3
```

3.8 Ariketa

Sortu 10 zenbaki osoko zerrenda bat definitzen duen programa bat. Ondoren, sortu begizta bat zerrendan elementuren bat errepikatuta dagoen bilatzeko. Errepikatutako bat aurkitzea nahikoa da.

```
numeruak = [3, 5, -4, 2, 1, 4, 0, 6, 9, 8, 3]
errepikatua = False
i = 0
j = 0
while i < len(numeruak) and not errepikatua:
 while j < len(numeruak):
   if numeruak[i] == numeruak[j]:
```

```
            errepikatua = True
            break
        j = j + 1
    i = i + 1
if errepikatua:
    print("Zenbaki bat errepikatuta dago")
else:
    print("Ez dago zenbaki errepikaturik")
```

Emaitza:

```
Zenbaki bat errepikatuta dago
```

3.9 Ariketa

Sortu 10 zenbaki osorekin hasitako zerrenda bat definituko duen programa. Gero, beste begizta bat sortu, zenbaki positiboak, negatiboak eta 0 direnak zenbatuko dituena.

```
zenbakiak = [3, 5, -4, 2, 1, 4, 0, 6, -9, 8,
3]

positiboak = 0
```

```
negatiboak = 0
hutsak = 0
for zenbakia in zenbakiak:
 if zenbakia > 0:
   positiboak = positiboak + 1
 elif zenbakia < 0:
   negatiboak = negatiboak + 1
 else:
   hutsak = hutsak + 1
print("Positiboak: ", positiboak)
print("Negatiboak: ", negatiboak)
print("Hutsak: ", hutsak)
```

Emaitza:

```
Positiboak: 8
Negatiboak: 2
Zeroak: 1
```

3.10 Ariketa

Sortu 5x10 elementuko bi dimentsioko zerrenda bat definituko duen programa bat. Zerrendako balioak ausazko (*aleatorio* edo *random*) zenbakiak erabiliz hasten dituen begizta bat sortzen du. Zenbaki aleatorioak sortzeko, `random` eta `random.randint()` funtzioa erabiltzen du liburutegiak, hemen erakusten den bezala:

```
import random

random.randint(0, 30); # 0 eta 30 arteko
ausazko zenbakia
```

Horren ondoren, sortu beste begizta bat, elementuren batean 15 zenbakia aurkituz gero begizta eten eta zer posiziotan dagoen erakutsiko duena.

```
import random

matrizea = [([0] * 10)] * 5

print(matrizea)

for i in range(len(matrizea)):

 random.seed()

 for j in range(len(matrizea[i])):

   matrizea[i][j] = random.randint(0, 30)
```

```
print(matrizea)
for i in range(len(matrizea)):
 for j in range(len(matrizea[i])):
   if matrizea[i][j] == 15:
        print("15 aurkitu da ", i, j)
```

3.11 Ariketa

Sortu 10 zenbaki osorekin hasitako zerrenda bat definitzen duen programa. Begizta batean, pantaila bidez erakusten ditu elementu guztiak. Gero, beste begizta bat sortzen du, eta, bertan, elementuak lekuz aldatzen aldatzen ditu, indizeetan random funtzioa erabiliz. Gero emaitza erakusten du.

```
import random
zenbakiak = [4, 7, -3, 7, 1, 11, 9, 0, 5, 8]
print(zenbakiak)
for i in range(len(zenbakiak)):
 ausazko_indizea = random.randint(0,
len(zenbakiak) - 1)  aurrekoa = zenbakiak[i]
```

```
    zenbakiak[i] = zenbakiak[ausazko_indizea]
    zenbakiak[ausazko_indizea] = aurrekoa

print(zenbakiak)
```

Emaitza:

```
[5, 4, 11, 7, 1, -3, 0, 9, 7, 8]
```

11 Funtzioak

miau()

lo()

Funtzioak programen barruan dauden programa txikiak dira. Adibidez, funtzio honek pantailan agur bat besterik ez du ateratzen:

```
def agurra ():
 print "Kaixo"
```

Ikus daitekeenez, funtzio bat definitzeko `def`, hitza erabiltzen da, eta, ondoren, funtzioaren izena `agurra` eta `()` parametroen zerrenda, kasu honetan hutsik dagoena. Funtzioaren barruan, nahi ditugun aginduak jar ditzakegu.

Eta Pythonen estiloarekin jarraituz, funtzioaren barruan dagoen kodearen aurretik tabulazioa edo espazioak daudela kontutan hartu behar da.

Funtzio hori definitu ondoren, funtzio hori erabiltzen (edo deitzen) dugun bakoitzean, funtzio horretan dagoen kodea exekutatuko da:

```
agurra()
```

Horrek erakutsiko du:

```
Kaixo
```

4.0 Ariketa

Idatzi programa bat hiru funtzioekin: egunak, arratsaldeak eta gauak. Bakoitzak agur desberdina egin behar du: `"Egun on"`, `"Arratsalde on"` eta `"Gabon"`, hurrenez hurren. Gehitu hiru funtzioetarako deiak.

```
def egunak ():
 print("Egun on")
```

```
def arratsaldeak ():
 print("Arratsalde on")
def gauekoak ():
 print("Gabon")
egunak()
arratsaldeak()
gauekoak()
```

Emaitza:

```
Egun on
Arratsalde on
Gabon
```

11.1 Parametroak

Funtzioek parametroak jaso ditzakete. Horiek aldagai bihurtzen dira funtzioaren barruan, eta funtzioak gertatzen zaienaren arabera gauza desberdinak egitea ahalbidetzen digute.

Adibidez, norbait agurtzen duen funtzio bat sor dezakegu:

```
def agurtu (pertsona):
 print ("Kaixo", pertsona)
```

Funtzioaren barruan, pertsonen balioa desberdina izango da deian ematen zaionaren arabera. Honela erabil liteke:

```
agurtu("Neko") #pertsona aldagaia "Neko"
izango da

agurtu("Ada") #pertsona aldagaia "Ada" izango
da
```

Hona hemen horren emaitza:

```
Kaixo Neko

Kaixo Ada
```

4.1 Ariketa

Idatz ezazu programa bat karratua izeneko funtzio batekin, parametro bat izango duena. Funtzioak parametroa bere buruarekin biderkatu (karratua) eta pantaila bakoitzean erakutsi behar du.

```
def karratua(a):
 print(a * a)
```

```
karratua(3)
```

Emaitza:

```
9
```

11.2 Lehenetsitako balioa duen parametroa

Funtzio baten parametroek lehenetsitako balioa izan dezakete. Hau da, balio jakin bat esleitu ahal zaie, eta deian parametro hori pasatzen ez bada, parametroak balio hori lehenetsiko du.

```
def agurtu(pertsona, aldiz = 3):
 for i in range (aldiz):
 print ("Kaixo", pertsona)
```

Honela deitzen bazaio:

```
agurtu("Neko", 2)
```

Pantailan ikusiko litzateke:

```
Kaixo Neko
Kaixo Neko
```

Aldiz, bigarren parametroa pasatzen ez badugu,aldizek lehenetsitako balioa (`3`) hartuko du: `agurtu("Bug")`

Hau ikusiko litzateke:

```
Kaixo Bug
Kaixo Bug
Kaixo Bug
```

11.3 Parametro infinituak

Python-en funtzioei esker, parametro kopuru zehaztugabea pasatzeko aukera daukazu. Arraro samarra dirudi, baina baliagarria izan daiteke. Imajinatu funtzio bat sortu nahi duzula, pasatzen dizkiozun parametro guztiak batuko dituena:

```
def batu (*balioak):
 emaitza = 0
 for balioa in balioak:
 emaitza = emaitza + balioa
 return emaitza
```

Erreparatzen baduzu, balioak parametroa zehaztu dugu, aurretik * bat jarrita. Horrekin esan nahi dugu ez dela parametro bakarra, edozein luzera izan dezakeen zerrenda baizik.

Beraz, funtzio horri honela deitu al diogu:

```
batu (3, 4, 5) # 12

batu (2, 45) # 47
```

4.2 Ariketa

Idatz ezazu programa bat parametro mugagabeak edo dinamikoak jasotzen dituen `hitzak_batu` izeneko funtzio batekin. Hitzak batzeko funtzioak parametroak hartu behar ditu, eta horiekin guztiekin testu bat itzuli, esaldi bat osatuz.

```
def hitzak_batu(*hitza):
 esaldia = ""
 for h in hitza:
 esaldia = esaldia + " " + h
 return esaldia
print(hitzak_batu("kaixo", "zer", "moduz"))
```

Emaitza:

11.4 Balioak itzultzen: `return`

Funtzioek beharrezkoak diren eragiketa guztiak egin ditzakete, baina emaitza bat itzultzen dutenean erabilgarriagoak dira. Horretarako, `return` agindua erabiltzen da, eta horrek datu jakin bat edo datu egitura bat baino ezin du itzuli.

Adibidez, bi balioren batura kalkulatzen duen funtzio bat:

```
def batu(a, b):
    emaitza = a + b
    return emaitza
```

Pixka bat laburtu daiteke zuzenean:

```
def batu(a, b):
    return a + b
```

Honela erabil liteke funtzioa:

```
print(batu(40, 2)) # 42
```

Gauza pare bat hartu behar dituzu kontuan:

1. Behin `return` egiten denean, programa funtziotik ateratzen da.

2. Funtzioan `return` bat baino gehiago izan dezakezu, baina horietako bat bakarrik exekutatuko da.

4.3 Ariketa

Idatzi programa bat zatitu izeneko funtzio batekin, eta jaso itzazu parametro hauek: `a` eta `b`. Funtzioak bi zenbakien arteko zatiketa itzuli behar du.

```
def zatitu (a, b):
 return a/b
print(zatitu(32, 4))
```

Emaitza:

```
8.0
```

Beste adibide bat, balio bat hainbat aldiz biderkatzen duen funtzio bat. Kopurua 1 baino txikiagoa bada, 0 bat itzuliko du:

```
def berredura (kopurua, aldiz):
 if (aldiz > 0):
```

```
    guztira = 1
    for i in range (aldiz):
        guztira = guztira * kopurua
    return guztira
  else:
    return 0
berredura (2, 3) # 8
```

Oharra 

aurreko funtzioa argiago geratzen da horrela.

```
def berredura (kopurua, aldiz):
 if (batzuetan < 1):
   return 0
 guztira = 1
 for i in range (batzuetan):
```

```
    guztira = guztira * kopurua
  return guztira # TODO CHECK
```

11.5 Dei kabituak

Ez izan funtzio dei kabiatuak egiteko beldurrik. Oso gauza naturala da programazioan. Imajinatu erabiltzaileari zenbaki bat eskatzeaz arduratzen den funtzio hau dugula eta sartutako zenbaki itzultzen duela:

```
def irakur_zenbakia ():
 zenbakia = input("Idatzi zenbakia:")
 return int(zenbakia)
```

Demagun beste funtzio bat ere badugula zenbakiak kentzeko:

```
def kenketa (a, b):
 return a - b
```

Erabiltzaileari bi zenbaki eskatu eta kentzen dizkion programa bat egin nahi badugu, horrela egin genezake.

```
print (kenketa(irakur_zenbakia(),
irakur_zenbakia()))
```

Zenbaki edo aldagai bat parametro gisa jarri beharrean, dei bat jar diezaiokegu funtzio bati.

Funtzio horiek `irakur_zenbakia()`, balio batzuk itzuliko dituzte. Lehenengo `irakur_zenbakia()` deitu bezain laster, honela izango da:

```
print (kenketa(5, irakur_zenbakia()))
```

Eta gero, bigarrenari `irakur_zenbakia()` deitu ondoren:

```
print (kenketa(5, 2))
```

Gero, kenketa egingo da eta balio bat itzuliko du:

```
print(3)
```

Eta, azkenik, print deituko da eta emaitza ikusiko dugu:

```
3
```

4.4 Ariketa

Idatzi bi funtzio. zenbaki bat jaso eta zenbaki bera positiboan itzultzen duena. `positibo(balioa)`

Eta bi zenbaki jaso eta horien artekopotentzia kalkulatzen duen beste funtzio bat. `berredura(balio1, balio2)`

Bigarren funtzioari deitu behar diozu, parametro hauek kontuan hartuta: `berredura (positiboa (-5), positiboa (4))`

Oharra: saiatu ez erabiltzen aurretik zeuden funtzioak.

```
def positibo (balioa):
 if balioa < 0:
   return -balioa
 return balioa
def berredura (balioa1, balioa2):
 return balioa1 ** balioa2


print(positibo(-1))
print(berredura(2, 3))
print(berredura(positibo(2), positibo(4)))
berredura(positibo(-5), positibo(4))
```

Emaitza:

```
1
```

```
8

16
```

11.6 Zergatik erabili funtzioak?

Zertarako sortu behar ditugu funtzioak? Bada, egia esan... funtsezkoak dira.

Orain arte, agindu-sekuentzia soil bat diren programak ikusi ditugu, hasieratik bukaeraraino exekutatzen direnak.

Hori ondo dago programak sinpleak direnean eta gauza gutxi egin behar dituztenean, baina programak zerbait konplexuagoa egin behar duenean, litekeena da funtzioak erabili behar izatea, hainbat arrazoirengatik.

11.6.1 0 arrazoia: zatitu eta irabaziko duzu

Programak arazoak konpontzen saiatzen dira, soluzio bat eskainiz. Batzuetan arazoak oso konplexuak izan daitezke. Funtzioek arazo horiei zatika heltzeko aukera ematen dizute. Funtzio bakoitzak arazoaren zati baterako konponbidea eman diezazuke. Beraz, arazo handia arazo txiki askotan bana dezakezu eta arazo bakoitza funtzio batekin konpon dezakezu. Programak funtzioetan banatzea programa konplexuagoak diseinatzeko lehen urratsa da.

11.6.2 1. arrazoia: kodea ez errepikatzea

Zure programak zerbait egin behar badu hainbat aldiz, kodea behar beste aldiz errepikatu beharko zenuke. Imajinatu erabiltzailearen hainbat datu jaso behar dituzula, eta hori egiten duzun bakoitzean datua hutsik ez dagoela egiaztatu behar duzula:

```
datua = ""
while datua == "":
 datua = input("Mesedez, sartu datu bat:")
 if datua == "":
 print ("Datua hutsik dago!")
```

Erabiltzaileari 3 datu eskatzen badizkiozu, kode hori 3 aldiz errepikatu beharko zenuke! Aldiz, kode bera duen funtzio bat sortzen baduzu, behin bakarrik idatzi beharko duzu, eta gero behar adina aldiz erabili ahal izango duzu.

Oharra 

kodea ez errepikatzea da programatzaile on orok jarraitu beharreko araurik garrantzitsuenetako bat. *"3aren araua"* ere

aplika dezakezu. Zerbait errepikatu behar duzun hirugarren aldian, automatizatu edo funtzio biurtu egin behar duzu.

11.6.3 2. arrazoia: berrerabili kodea

Kodea ez errepikatzeaz gain, funtzio batek aukera ematen digu kode berak hainbat datu-motatarako balio dezan. Horretarako erabiltzen dira parametroak!

```
def agurtu (agurra, batzuetan):
 for i in range (batzuetan):
 print (agurra)
```

Balio desberdinekin dei dakioke:

```
agurtu ("Holi", 3)
agurtu ("Aupa", 1)
```

Emaitza hau litzateke:

```
Holi
Holi
Holi
Aupa
```

11.6.4 3. arrazoia: mantentze lanak erraztea

Kodea leku bakar batean badago, errazagoa da zuzentzea, aldatzea, hobetzea eta orokorrean mantentzea. Programa bat zerotik sortzea oso polita da, baina programatzaileen benetako lana kodea denboran zehar mantentzea da. Gure funtzioak ondo diseinatuak baditugu, lana *PIIIIILA* aurreztuko dugu.

11.6.5 4. arrazoia: testak egiteko aukera ematen du

Agian gazte samarra zara honetarako, baina profesional onek programak probatzen dituzte. Zer esan nahi du horrek? Beraien programak ongi daudela egiaztatzen duten programak idazten dituztela. Zure kodeak funtzioak baditu, testak idatzi ahal izango dituzu funtzio horiek behar dutena egiten dutela egiaztatzeko.

Berez, aditua zarenean, zure eginkizuna bera baino lehenago... test programa idatzi beharko duzu!

11.7 Nola egin funtzio onak

Edonork idatz ditzake funtzioak eta kodea zati txikitan antolatu. Baina pro gisa funtzioak idatzi nahi badituzu, honako hau lortu behar duzu:

- Funtzio batek **gauza bakarra** egin behar du. Hobe da funtzio txiki asko izatea, gauza asko egiten dituzten funtzio gutxi

izatea baino. Zure funtzioa pantailan sartzen ez bada edo 24 lerro gainditzen baditu, agian zati txikitan banatu beharko duzu.

- Funtzio batek ez luke bere **kanpoan dagoen ezer** aldatu behar. Ezustekorik izan nahi ez baduzu, funtzio batek ez luke programaren barruan ezustekorik eragin behar.
- Funtzio batek **zerbait itzuli** beharko luke, eta horrek beti berdina izan beharko luke parametro jakin batzuetarako.

11.8 Proposatutako ariketak

4.0 Ariketa

Idatzi bi zenbaki jasotzen dituen funtzio bat, detektatu zein den handiena eta erakutsi horien arteko aldea.

```
def diferentzia (balio1, balio2):
 kenketa = 0
 if balio1 > balio2:
   kenketa = balio1 - balio2
 else:
   kenketa = balio2 - balio1
 print("Aldea hau da: ", kenketa)
diferentzia(10, 5)
diferentzia(4, 12)
```

Emaitza:

```
Aldea hau da: 5
```

```
Aldea hau da: 8
```

4.1 Ariketa

Idatzi bi zenbaki eta eragiketa zeinu aritmetiko bat jasotzen dituen programa bat: +, -,*, /. Funtzioak eragiketa horren emaitza itzuli behar du bi zenbakien artean.

```
def kalkulatu (balioa1, balioa2, eragiketa):

 if eragiketa == "+":
   return balioa1 + balioa2

 elif eragiketa == "-":
   return balioa1 - balioa2

 elif eragiketa == "*":
   return balioa1 * balioa2

 elif eragiketa == "/":
   return balioa1 / balioa2

 else:
   return "Eragiketa ezezaguna"


emaitza = kalkulatu(4, 6, "*")

print(emaitza)
```

```
emaitza = kalkulatu(5, 5, "-")
print(emaitza)
emaitza = kalkulatu(4, 3, "@")
print(emaitza)
```

Emaitza:

```
24
0
Eragiketa ezezaguna
```

4.2 Ariketa

```
def agurtu (momentua)
```

Funtzio honek parametro gisa eguneko une bat hartzen du: `"goizean"`, `"arratsaldean"` edo `"gauean"`, eta dagokion agurra itzuli behar du: `"Egun on"`, `"Arratsalde on"`, eta `"Gabon"`, hurrenez hurren.

```
def agurtu(momentua):
 if momentua == "goizean":
   return "Egun on"
```

```
    elif momentua == "arratsaldean":
      return "Arratsalde on"
    elif momentua == "gauean":
      return "Gabon"
    else:
      return ""
  print(agurtu("arratsaldean"))
  # beste bertsio bat
  def agurtu2(momentua):
    return {
    "goizean": "Egun on",
    "arratsaldean": "Arratsalde on",
    "gauero": "Gabon"
    }[momentua]
  print(agurtu2("arratsaldean"))
```

Emaitza:

```
Arratsalde on.

Arratsalde on.
```

4.3 Ariketa

```
hasiZenbakiarekin (luzera, zenbakia)
```

Funtzio honek zerrenda bat sortu behar du, bi parametroak kontutan hartuz: luzera zenbaki kopuruarekin, eta zenbaki guztiak zenbakia balioarekin bete behar dira.

```
def hasieratuZenbakiarekin(luzera, zenbakia):

 zenbakiak = []

 for i in range(luzera):

 zenbakiak.append(zenbakia)

 return zenbakiak

def hasieratuZenbakiarekin2(luzera, zenbakia):
    return [zenbakia] * luzera

print(hasieratuZenbakiarekin(10, 3))

print(hasieratuZenbakiarekin(10, 3))
```

Emaitza:

```
[3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
[3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

4.4 Ariketa

```
def ausazkoa (max)
```

Funtzio horrek 0 eta parametro gisa pasatzen den gehieneko balioaren arteko ausazko (*random*) zenbaki bat itzuli behar du.

```
import random
def ausazkoa (max):
 return random.randint(0, max)
print(ausazkoa(5))
```

Emaitza:

```
3
```

4.5 Ariketa

```
izenaSortu(silabak)
```

Silaba kopuru bat dela eta, ausazko izen bat sortzen duen funtzio bat idatzi (kontsonanteak eta bokalak txandakatu). Aurreko ariketako funtzioa erabil dezakezu.

```
import random

def ausazkoa(max):

 return random.randint(0, max - 1)

def izenaSortu(silabak):

 bokalak = ["a", "e", "i", "o", "u"]

 konsonanteak =

[
"b","c","d","f","g","h","j","k","l","m","n","ñ
","p","q","r","s","t","v"," w","x","y","z"]

 izena = ""

 for i in range(silabak):

   konsonantea =
konsonanteak[ausazkoa(len(konsonanteak))]
bokala = bokalak[ausazkoa(len(bokalak))]

   izena = izena + konsonantea +  bokala
```

```
    return izena

print(izenaSortu(3))
```

Emaitza:

```
xamozu
```

4.6 Ariketa

```
def pasahitzaSortu(luzera)
```

Luzera bat emanda, zorizko karaktereak dituen string bat sortzen duen funtzioa. Karakteredun string zerrenda bat erabil dezakezu, eta ausazko karaktereak atera zerrendatik, izen bat sortzeko. Ausazko zenbakiak sortzeko `random.randint` funtzioa erabil dezakezu.

```
import random

def ausazkoa (max):

 return random.randint(0, max - 1)

def pasahitzaSortu (luzera):

 hizkiak =
[
"a","b","c","d","e","f","g","h","i","j","k","l
```

```
",
"m
","n","ñ","o","p","q","r","s","t","u","v","w",
"x","y","z",
"0
","1","2","3","4","5","6","7","8","9",".","-",
"_","!","$"]
pasahitza = ""
 for i in range(luzera):
   hizkia = hizkiak[ausazkoa(len(hizkiak))]
   pasahitza = pasahitza + hizkia
 return pasahitza
print(pasahitzaSortu(8))
```

Emaitza:

```
_! 5_flg $
```

4.7 Ariketa

Sortu `faktura` izeneko funtzioa (`produktuak`, `kantitateak`, `prezioak`), tamaina bereko hiru array jasotzen dituena, honako eduki hauekin:

- `produktuak`: produktuen izenak.
- `kantitateak`: zenbaki osoak, kantitatea adierazteko.
- `prezioak`: zenbaki hamartarrak, produktu bakoitzaren prezioa adierazteko.

Funtzioak produktu bakoitza aztertu eta prezio osoa kalkulatu behar du, kantitatearen eta prezioaren arabera. Produktu bakoitzaren prezio osoa erakutsi behar da, eta programaren amaieran, batura osoaren prezioa erakutsi behar da.

```
def faktura(produktuak, kantitateak,
prezioak):

 faktura = "FAKTURA\n-------------------\n"

 totala = 0

 for i in range(len(produktuak)):

   faktura = faktura + produktuak[i]

   faktura = faktura + " x " +
str(kantitateak[i])

   faktura = faktura + " : " +
str(prezioak[i]) + "\n"  totala = totala +
(kantitateak[i] * prezioak[i])
```

```
    faktura = faktura + "\
n----------------------\n"

    faktura = faktura + "Totala: " +
str(totala)

 return faktura

# Adibidea

fakturaGuztira =
faktura(["Ogia","Arraultza","Irina"],[1,6,2],
[1.2, 0.2, 0.8])

print(fakturaGuztira)
```

Emaitza:

```
FAKTURA

- ------------------

Ogia x 1: 1.2

Arraultzak x 6: 0.2

Irina x 2: 0.8

- ----------------------
```

```
Guztira: 4.0
```

4.8 ariketa

Hau zailagoa izango da. RPG jokuetako pertsonaiak sortzeko funtzio bat duen programa bat sortzen du.

```
def atributuakSortu (mailaKonpentsazioa)
```

Funtzio horrek hiru aldagai definitu behar ditu: `indarra`, `abiadura` eta `adimena`. Programak 20 puntu banatu behar ditu hiru aldagaien artean. Bestela esanda, hiru aldagaien artean 20 batu behar dira.

`mailaKonpentsazioa` parametroak puntu oso bereiziak edo berdinduak banatzen diren adierazteko balio behar du, balio desorekatuena zenbat eta handiagoa izan, hau da, atributuen arteko aldea handiagoa da; hori nola lortu programatzailearen kontua da.

Azkenean, programak esleitutako puntuen laburpena erakutsi behar du.

```
import random

def ausazkoa(maximoa):

 return random.randint(0, maximoa)
```

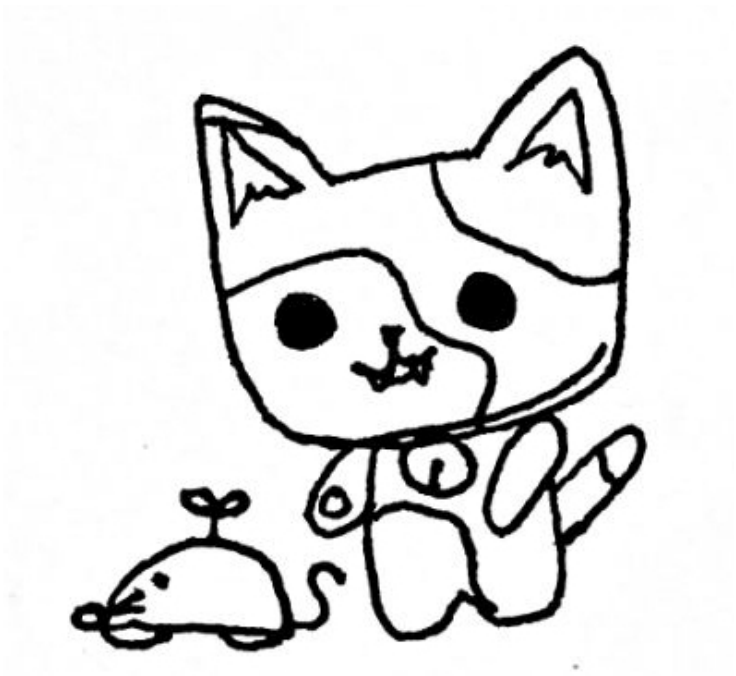
```
def atributuakSortu(konpentsaketaMaila):
 puntuakEman = 0
 adimena = 0
 indarra = 0
 abiadura = 0
 gainontzekoPuntuak = 20
 puntuak = 0
 while gainontzekoPuntuak > 0:
   if konpentsaketaMaila > gainontzekoPuntuak:
         puntuak = gainontzekoPuntuak
         gainontzekoPuntuak = 0
   else:
         puntuak = ausazkoa(konpentsaketaMaila
+ 1)
         gainontzekoPuntuak =
gainontzekoPuntuak - puntuak
         puntuakEman = ausazkoa(3)
```

```
    if puntuakEman == 0:
        adimena = adimena + puntuak
    elif puntuakEman == 1:
        indarra = indarra + puntuak
    elif puntuakEman == 2:
        abiadura = abiadura + puntuak
    print("\nBalioak konpentsazioaren arabera
adierazita: ", konpentsaketaMaila)
    print("Adimena: ", adimena)
    print("Indarra: ", indarra)
    print("Abiadura: ", abiadura)
atributuakSortu(3)
```

Emaitza:

```
Balioak konpentsazioaren arabera adierazita: 3
Adimena: 3

Indarra: 8

Abiadura: 5
```

12 Klaseak



Klaseek objektuei zuzendutako programazioa izeneko teknika aplikatzeko aukera ematen digute. Arazo konplexuak konpontzeko beste estrategia bat da. Funtzioekin, arazo bat arazo txikietan

banatzen dugu. Aldiz, objektuetara zuzendutako programazioarekin, arazoa klasetan banatzen saiatzen gara. Baina nola? Arazoaren zati den guztia irudikatuz, klaseak erabiliz. Hots, arazoan existitzen diren gauzak objektuak bilakatuz!

Demagun Mario Kart bezalako lasterketa joko baten programa egin behar dugula. Objektuetara zuzendutako programazioa erabiliz, jokoaren elementuak honako klase hauekin irudika ditzakegu:

Pertsonaia, bere izenarekin eta beste ezaugarri batzuekin.

Autoa, bere ezaugarri hauekin: abiadura, erresistentzia, azelerazio funtzioak, etab. Zirkuitua, luzerarekin, tunelekin, sariekin eta abarrekin.

12.1 Nola sortu klaseak

Klase bat programazio egitura bat da, gauza bat bere propietate eta metodoekin errepresentatzeko aukera ematen diguna. Hau da, klase batek:

- **Atributu** edo propietateak ditu: klasearen aldagaiak.
- Gauzak egiten ditu: **funtzioak**.

Adibidez, hurrengo klaseak katu oso sinple bat errepresentatzen du, miauka egiteko funtzio batekin:

```
class Katua:

 def miaukatu(self):

   print ("Miau")
```

Ikus daitekeenez, klasea definitzeko, `class` hitza erabiliko dugu, eta, ondoren, klasearen izena, lehenengo letra larriz. Bloke horren barruan doan guztia klasearen parte izango da.

Bestalde, kontuan izan behar duzu klase baten funtzio guztiek `self` parametro berbera izan behar ditutela, nahiz eta ez erabili. Parametro hori klaseari berari dagokio, eta haren propietateei eta funtzioei erreferentzia egiteko erabiltzen da, gero ikusiko dugun bezala.

12.2 Klasea vs instantzia

Klasea katuaren definizioa besterik ez da. Baina gure programan katu bat sortzeko, instantzia bat sortu behar dugu. Honela egiten da:

```
katu = Katua()

katu.miaukatu()
```

Pantailan honako hauek ikusiko ditugu:

```
Miau
```

Instantzia objektu zehatz bat da. Klasea definizioa baino ez da: katu batek izena eta miaukatu funtzioa ditu. Instantzia objektu zehatz bat da: katu.

5.0 Ariketa

Idatzi `Pertsona` izeneko klase bat, lo egiteko, jateko eta agurtzeko metodoak dituen funtzioak definituko duen programa bat. Funtzio edo metodo bakoitzaren barruan testuren bat atera behar duzu kontsola bidez. Sortu klasearen instantzia bat eta deitu metodo desberdinetara.

```
class Pertsona:

 def loEgin(self):

   print("ZZZZZ...")

 def jan(self):

   print("Ñam, Ñam...")

 def kaixo(self):

   print("Kaixo, zer moduz!")
```

```
pertsona = Pertsona()
pertsona.loEgin()
pertsona.jan()
pertsona.kaixo()
```

Emaitza:

```
ZZZZZ...
Ñam, Ñam...
Kaixo, zer moduz!
```

12.3 Eraikuntza funtzioa

Lehenago definitu dugun katu horri izen bat emango diogu. Horretarako, funtzio berezi bat sortuko dugu, `__init__` izenekoa. eraikuntza-funtzioa esaten zaio (*constructor* ingelesez). Funtzio hau klaseko objektu bat sortzen denean deitzen da, eta, beraz, klasearen propietateak hasieratzeko lekurik egokiena da:

```
class Katua:
 def __init__ (self, izena):
   self.izena = izena
```

```
    def miaukatu (self):
        print("Miau,", self.izena, "naiz.")
```

Orain, Katua klaseko objektuak sortzen ditugunean, izen bat emango diogu, eta hau `izena` atributuan geratuko da:

```
katua = Katua("Pixi")
katua.miaukatu()
besteKatua = Katua("Txeto")
besteKatua.miaukatu()
```

Eta kasu honetan ikusiko dugu:

```
Miau, Pixi naiz.
Miau, Cheto naiz.
```

5.1 Ariketa

Idatzi programa bat `Kaixo` izeneko klase bat definitzeko. Klaseak funtzio eraikitzaile bat izan behar du, `agurra` izeneko atributu batekin. Atributu hori `"kaixo"` hitzarekin hasiko da. Horrez gain, `esanKaixo` izeneko metodo bat erantsiko duzu, eta, pantailan, `agurra`ren edukia erakutsiko du.

```
class Kaixo:
 def __init__(self):
   self.agurra = "Kaixo"
 def esanKaixo (self):
   print(self.agurra)
kaixo = Kaixo()
kaixo.esanKaixo()
```

Emaitza:

```
Kaixo
```

12.4 Herentzia

Herentzia kodea berrerabiltzeko klaseek duten mekanismo bat da. Demagun `Katakume` bat definitzen duen klase bat egin nahi dugula. `Katu` klaseak egiten duen gauza bera egitea nahi dugu, baina, gainera, zurrunga egitea. `Katakumea` klaseak `Katua` klasetik heredatu ahal izango luke, honela:

```
class Katakumea (Katua):
 def purrustatu (self):
```

```
    print ("Purrrr")
```

Orain honako hau egin dezakegu. `Katakumea` objektu bat sortzea, `Katua` klasearen propietate berberekin. Automatikoki jasoko ditu `izena` atributua eta `miaukatu` metodoa:

```
katutxoa = Katakumea ("Lucifur")

katutxoa.purrustatu()

katutxoa.miaukatu()
```

Honela ikusiko litzateke:

```
Purrrr

Miau, Lucifur naiz.
```

12.5 super()

Azpiklase bat edo klase bat beste baten semea sortzen duzunean, heredatzen duzun klasetik `super()` funtzioa erabil dezakezu oinordetzan hartutako klasearen funtzioei deitzeko. Adibidez, aurreko kasuan, `Katakumea` azpimailatik berezko eraikitzaile bat gehi genezake, eta `Katua` superklasearen eraikitzaileari ere dei diezaiokegu:

```
class Katakumea (Katua):
```

```
    def __init__ (self, izena):
        super () .__init__ (izena)
        print ("Katakumea sortua")
    def purrustatu (self):
        print ("Purrrr")
```

5.2 Ariketa

Idatz ezazu `Janaria` klasea definituko duen programa bat, `izena` atributuarekin. Sor ezazu `Fruitua` izeneko azpiklase bat, janari `izena` eta `bitaminak` jasotzen dituen eraikitzaile batekin, eta `info` izeneko metodo bat, bere informazio guztia itzultzen duena. Sortu instantzia bat `Fruitua` klasea probatzeko.

```
class Janaria:
 def __init__(self, izena):
    self.izena = izena
class Fruitua(Janaria):
 def __init__(self, izena, bitaminak):
    super().__init__(izena)
```

```
        self.bitaminak = bitaminak
    def info(self):
        # return f"{self.izena} {self.bitaminak}";
        return self.izena + " " +
str(self.bitaminak)
postrea = Fruitua("Kiwi", ["A", "C"])
print(postrea.info())
```

Emaitza:

```
Kiwi ['A', 'C']
```

12.6 Enkapsulazioa

Katuaren adibidean, zuzenean sar daiteke `izena` propietatea. Horretarako, objektuetan nahikoa da horrelako zerbait jartzea:

```
objektua.AtributuIzena
```

Katuak izena izeneko propietate bat du.

```
nireKatua = Katua("Pixi")
print (nireKatua.izena) #Pixi
```

```
nireKatua.izena = "Pixel"

nireKatua.miaukatu () #Miau, Pixel naiz
```

Propietate bat hain zuzenean eskuratzea ez dago gaizki, baina programatzaile onak klasea **enkapsulatzen** saiatzen dira. Zer esan nahi du horrek? Ezin direla zuzenean haren propietateak irakurri edo aldatu. Klasea erabiltzen dutenentzat beharrezkoa dena bakarrik eskeini behar zaie. Bestela esanda, programatzaileek "kutxa beltzak" diruditen klaseak sortzen saiatu behar dute. Izena bezalako propietateen kasuan, Python-en metodo hauek gehitu daitezke:

Izenaren jabetzaren balioa itzultzeko metodo bat, *getter* ere esaten zaiona:

```
@property
def izena ():
  return self._izena
```

Izenaren balioa aldatu ahal izateko metodo bat, *setter* ere esaten zaiona:

```
@izena.setter
def izena (izena):
```

```
    if izena! = "":
        self._izena = izena
```

Klasea honela geratuko litzateke:

```
class Katua:
  def __init__(self, izena):
    self._izena = izena
  @property
  def izena(self):
    return self._izena
  @izena.setter
  def izena(self, izena):
    if izena != "":
        self._izena = izena
  def miaukatu(self):
    print("Miau, ni naiz ", self.izena)
```

Ikus ezazu propietatea izena dela orain. Jabetza hori "*pribatua*" dela adierazteko modu bat da, eta ez litzatekeela klasetik kanpo eskuratu behar. Orain, `Katua` izeneko klasea erabiltzen dugunean, funtzio berri horien bidez egingo da.

```
nireKatua = Katua ("Pixi")

print (nireKatua.izena) # deitu `def izena ()` metodora

nireKatua.izena = "Pixel" #deitu `def izena (izena) metodora nireKatua.miaukatu () # Miau, ni naiz Pixel
```

5.3 Ariketa

Idatz ezazu Ibilgailua klasea definituko duen programa bat, matrikula atributuarekin, *getter/setter* metodoekin eta abiarazi izeneko beste metodo batekin. Sortu `Kotxea` izeneko azpiklase bat, `Ibilgailua` klasea zabaltzeko, zein eraikitzaile batean `matrikula`, `modeloa` eta `kolorea` jasoko dituen, eta informazio guztia itzultzeko funtzio bat izango duena. Sortu instantzia bat `Kotxea` klasea probatzeko.

```
class Ibilgailua:

 def __init__(self, matrikula):
```

```
    self._matrikula = matrikula
  @property
  def matrikula (self):
    return self._matrikula
  @matrikula.setter
  def matrikula (self, matrikula):
    self._matrikula = matrikula
  def abiarazi (self):
    print("Abiarazten ", self._matrikula)


class Kotxea(Ibilgailua):
  def __init__(self, matrikula, modeloa,
kolorea):
    super().__init__(matrikula)
    self._modeloa = modeloa
    self._kolorea = kolorea
  def info (self):
```

```
        return f"{self.matrikula} {self._modeloa}
    {self._kolorea}"

        kotxea = Kotxea("0042ASI", "Opel Corsa",
    "Zuria")

    kotxea.abiarazi()

    print(kotxea.info())
```

Emaitza:

```
0042ASI Abiarazten

0042ASI Opel Corsa Zuria
```

Zer abantaila izan dezake enkapsulazioak?

Funtsean, "kanpotik" ezin da objektua kontrolik gabe manipulatu. Horregatik kutxa beltz bat bezalakoa da. Bideo-kontsola batekin konparatu daiteke. Joko bat jokatzeko eskuz ireki eta soldatu beharko bazenu, ziurrenik kontsola apurtuko zenuke. Horregatik, aparatuak kutxa beltz gisa diseinatzen dira, kanpotik manipulatzeko aukera batzuk baino ez dizute ematen. `Katua` klasearen kasuan, ez dugu uzten `izena` zuzenean aldatzen. *setter* funtzioaren bidez, esleitu nahi den `izena` zuzena dela kontrola dezakegu.

12.7 Klaseak klase barruan

Objektuetara zuzendutako programazioarekin, klaseen bidez mundu errealeko gauzak irudikatzen saiatzen gara. Eta klase horiek bata bestearekin lotuta egon daitezke.

Adibidez, demagun zure ikastola klaseen bidez errepresentatu nahi dugula. Ikastetxe batek gelak izan ditzake, gela batek ikasleak eta irakasleak izan ditzake, etab. Klaseek, beraz, beste mota batzuetako propietateak edo beste klase batzuetako zerrendak ere izan ditzakete.

```
class Ikaslea:
 def __init__ (self, izena):
 self.izena = izena
class Gela:
 def __init__ (self):
   self._ikasleak = []
 def sartuIkaslea (self, ikaslea):
   self._ikasleak.append (ikaslea)
 def listaPasa (self):
```

```
        for ikaslea in self._ikasleak:
            print (ikaslea.izena)
    ikasle1 = Ikaslea ("Gumball")
    ikasle2 = Ikaslea ("Darwin")
    gela = Gela ()
    gela.sartuIkaslea(ikasle1)
    gela.sartuIkaslea(ikasle2)
    gela.listaPasa()
```

Emaitza:

```
Gumball
Darwin
```

Diseinuak behar bezain konplexuak izan daitezke behar duguna irudikatzeko.

5.4 Ariketa

Idatz ezazu programa bat, `Pilotoa` eta *get/set* funtzioak dituena. Era berean, sor ezazu `Hegazkina` izeneko klase bat, `modeloa`, `pilotoa` eta `kopilotoa` atributuarekin, *get/set* funtzioekin, eta

`info` metodo batekin atributuen balioak pantailaratzeko. Sortu instantzia bat bi klaseak probatzeko.

```
class Pilotoa:
 def __init__(self, izena):
   self._izena = izena
 @property
 def izena (self):
   return self._izena
 @izena.setter
 def izena (self, izena):
   self._izena = izena


class Hegazkina:
 def __init__(self, modeloa, pilotoa,
koPilotoa):
   self._modeloa = modeloa
   self._pilotoa = pilotoa
```

```
        self._koPilotoa = koPilotoa
    @property
    def modeloa (self):
        return self._modeloa
    @modeloa.setter
    def modeloa (self, modeloa):
        self._modeloa = modeloa
    def info (self):
        return f"{self._modeloa} modeloa,
{self._pilotoa.izena} eta
{self._koPilotoa.izena}rekin hegaldia egiten"


pilotoa1 = Pilotoa("Han Solo")
pilotoa2 = Pilotoa("Murdock")
hegazkinTxikia = Hegazkina("AirBluff 727",
pilotoa1, pilotoa2)
print(hegazkinTxikia.info())
```

Emaitza:

```
AirBluff 727 modeloa, Han Solo eta Murdockekin
hegaldia egiten
```

12.8 Metodo estatikoak

Normalean, klase bat erabili ahal izateko, haren instantzia bat sortzen dugu beti, aurreko adibidean egiten genuen bezala:

```
ikasleBat = Ikaslea ("Gumball")
```

Batzuetan, agian, kopiarik egin nahi ez dugun klase bat sortu nahi dugu, zeregin zehatz bat egiteko bakarrik balio duena. Funtzio bat balitz bezala.

Adibidez, izen bat emanda, formatu zuzena ematen dion klase bat egin dezakegu, lehenengo letra larriarekin eta gainerako letra xeheekin:

```
class Formatua:

 @staticmethod

 def zuzendu (izena):

 return izena [0] .upper () + izena
[1:] .lower ()

print (Formatua.zuzendu("gUmBaLl"))
```

Emaitza:

```
Gumball
```

5.5 Ariketa

Idatzi programa bat, `Zenbakia` izeneko klase bat definituko duena, eta funtzio estatiko bat, `ausazkoa(max)` izenekoa. Funtzio honek zenbaki bat itzuli behar du 0 eta `max` tarteen barruan.

```
import random
class Zenbakia:
 @staticmethod
 def ausazkoa (max):
 return random.randint(0, max)
for i in range(5):
 print(Zenbakia.ausazkoa(10))
```

Emaitza:

```
4
3
```

```
0
9
1
```

ADI 

Zergatik egin klase bat horrelako funtzio batekin eta ez zuzenean funtzio batekin? klase batean egitea baliagarria izan daiteke leku berean funtzio estatiko bat edo gehiago sartu nahi ditugunean, eta ez ditugu instantzia desberdinak sortu nahi, funtzio zehatzak erabili baizik.

ADI 

Ez ahaztu klaseak eta objektuei zuzendutako programazioa diseinu estilo bat dela: kasu askotan, programaren erabilera errazteko eta kodea modu argiagoan banatzeko erabiltzen dira. Hasieran azaldu den bezala, arazoak konpontzeko bide bat eskeintzen digute.

12.9 Proposatutako ariketak

5.0 Ariketa

Idatzi `Instrumentua` izeneko klase bat definitzen duen programa bat. Eraikitzaileak parametro hauek izan behar ditu, hurrenez hurren: `izena` eta `mota`. Gainera, `jo` izeneko funtzio bat gehitu behar duzu, `mezu` bat aterako duena. Baita `info` funztioa, atributuen informazioa duen testu bat itzultzeko. Sortu klasearen instantzia bat eta deitu bere funtzioetara.

```
class Instrumentua:
 def __init__(self, izena, mota):
   self._izena = izena
   self._mota = mota
 def jo (self):
   print("Jotzen ", self._izena, "jotzen")
 def info (self):
   return f"{self._izena} {self._mota}"
```

```
nireGitarra = Instrumentua("Gitarra",
"klasikoa")

nireGitarra.jo()

print(nireGitarra.info())
```

Emaitza:

```
Gitarra jotzen

Hari-gitarra
```

5.1 Ariketa

Idatz ezazu programa bat, `IzenZuzena` izeneko klase bat definituko duena, eta, bertan, `__init__` eraikitzailea bat `izena` eta `abizena` atributuak hasieratzeko. Gainera, klaseak `zuzendu` izeneko metodo bat izan behar du, `izena` eta `abizena` atributuak zuzenduko dituena. `zuzendu` metodoak, lehenengo hizkiak larri eta gainerakoak xehe itzuliko ditu.

```
class IzenZuzena:

 def __init__(self, izena, abizena):

   self._izena = izena

   self._abizena = abizena
```

```
 def zuzendu (self):

   return self._zuzendu(self._izena) + " " +
self._zuzendu(self._abizena)

 def _zuzendu (self, hizkiak):

   return self._lehenengo(hizkiak) +
self._gainera(hizkiak)

 def _lehenengo (self, hizkiak):

   return hizkiak[0].upper()

 def _gainera (self, hizkiak):

   return hizkiak[1:len(hizkiak)].lower()


zuzentzailea = IzenZuzena("JUAN", "PÉREZ")

print(zuzentzailea.zuzendu())
```

Emaitza:

```
Juan Pérez
```

5.2 Ariketa

Idatz ezazu programa bat `Batuketa`, izeneko klase bat definitzeko, eta bi zenbakirekin hasieratzeko. `Batuketa(28, 14)`

Bientzako *get* eta *set* funtzioak barne hartzen ditu, eta balio negatiboa esleitzen saiatzen bazaizkie `0` esleitzea kontrolatu behar duzu. Gainera, bi zenbakien batura itzuliko duen `batu` funtzioa izango dute.

```
class Batuketa:
 def __init__(self, balio1, balio2):
   self.balio1 = balio1
   self.balio2 = balio2
 @property
 def balio1 (self):
   return self._balio1
 @balio1.setter
 def balio1 (self, balio1):
   if balio1 > 0:
         self._balio1 = balio1
```

```
    else:
    self._balio1 = 0
  @property
  def balio2 (self):
    return self._balio2
  @balio2.setter
  def balio2 (self, balio2):
    if balio2 > 0:
          self._balio2 = balio2
    else:
          self._balio2 = 0
  def batu (self):
    return self._balio1 + self._balio2

batuketa = Batuketa(28, 14)
print(batuketa.batu())
```

```
batuketa.balio1 = 600
batuketa.balio2 = 66
print(batuketa.batu())
```

Emaitza:

```
42
666
```

5.3 Ariketa

Sortu programa bat `Txanpona` izeneko klase batekin. Klaseak eraikitzaile huts bat izan behar du, eta funtzio bakar bat, `bota` izenekoa. Horren emaitza `"aurpegia"` edo `"gurutzea"` artean ausaz aukeratutako string bat izan behar da. Sortu klasearen instantzia bat probatzeko.

```
import random
def ausazkoa (max):
 return random.randint(0, max)
class Txanpona:
 def bota (self):
```

```
        aldeak = ["aurpegi", "gurutze"]
        zenbakia = ausazkoa(1)
        return aldeak[zenbakia]

txanpona = Txanpona()
for i in range(10):
 print(txanpona.bota())
```

Emaitza:

```
aurpegia
aurpegia
gurutzea
gurutzea
gurutzea
...
```

5.4 Ariketa

Sortu programa bat N aurpegiko dado baten portaera simulatzeko, `Dadoa` izeneko klase batekin. Sortu klaseko instantzia bat jaurtiketak probatzeko.

- `def __init__ (self, aldeak, zeroOnartu = False)`: aurpegi-kopurua gordetzen duen atributua eta `zeroOnartu` boolearra: dadoak 0 balioa itzul dezakeen esaten digun atributua. Lehenetsitako balioa `False` da.
- `def aldeak (self, aldeak)`: Parametrodun *setter* funtzioa. aldeak atributua ezartzen du.
- `def zeroOnartu (self)`: parametrodun *setter* funtzioa, boolear atributua ezartzen du.
- `def bota (self)`: funtzio honek dadoaren jaurtiketa simulatzen du eta emaitza bat itzultzen du. Kontuan hartu behar duzu `zeroOnartu` atributua.

Sor itzazu 6 aurpegiko dado bat, 10 aurpegiko dado bat eta zeroak ahalbidetzen dituen 20 aurpegiko dado bat sortzen duten instantziak, eta egizu bakoitzetik 100 jaurtiketa.

```
import random

def ausazkoa (maximoa):

 return random.randint(0, maximoa)
```

```
class Dadoa:
 def __init__(self, aldeak = 6, zeroOnartu =
False):
   self._aldeak = aldeak
   self._zeroOnartu = zeroOnartu
 @property
 def aldeak (self):
   return self._aldeak
 @aldeak.setter
 def aldeak (self, aldeak):
   self._aldeak = aldeak
 @property
 def zeroOnartu (self):
   return self._zeroOnartu
 @zeroOnartu.setter
 def zeroOnartu (self, zeroOnartu):
   self._zeroOnartu = zeroOnartu
```

```
    def bota (self):
        zenbaki = ausazkoa(self._aldeak)
        if not self._zeroOnartu:
            zenbaki = zenbaki + 1
        return zenbaki

dadoa = Dadoa()
for i in range(10):
    print(dadoa.bota())
```

Emaitza:

```
4
3
2
4
3
...
```

5.5 Ariketa

Sortu bi klase dauzkan programa bat:

1- `Jokalaria` klasea, `izena`, `posizioa` eta `zenbakia` propietateak dituena. Parametro horiek guztiak dituen eraikitzaile bat ere sortu, eta ezaugarri guztiak itzuliko dituen `txosten` izeneko funtzio bat.

2 - `Taldea` klasea, `izena`, `fundazioa`, `aurrekontua` propietateekin eta `jokalariak`: `Jokalaria` motako instantziak gordetzeko propietatea (zerrenda izango dena). Honako ezaugarri hauek dituen eraikitzaile bat izan behar du. Taldea klaseak beste bi funtzio izango ditu: `jokalariaFitxatu` eta `jokalariakErakutsi`:

- `def jokalariaFitxatu (self, jokalaria)`, zerrendara jokalariak gehitzeko.
- `def jokalariakErakutsi (self)`, jokalarien datu guztiak jasotzen dituen kate bat itzultzeko

Gainera, gehitu bi jokalari eta talde bat sortzeko beharrezkoa den kodea, eta horri jokalariak gehitu eta erakutsiko dituzu.

```
class Jokalaria:
```

```
 def __init__(self, izena, posizioa,
zenbakia):
    self._izena = izena
    self._posizioa = posizioa
    self._zenbakia = zenbakia
 def txosten (self):
    return f"{self._izena} {self._posizioa}
{self._zenbakia}"
class Taldea:
 def __init__ (self, izena, sortzea,
aurrekontua):
    self._izena = izena
    self._sortzea = sortzea
    self._aurrekontua = aurrekontua
    self._jokalariak = []
 def jokalariaFitxatu (self, jokalaria):
    self._jokalariak.append(jokalaria)
```

```
    def jokalariakErakutsi (self):
        for jokalaria in self._jokalariak:
            print(jokalaria.txosten())


jokalaria1 = Jokalaria("Maradona",
"Aurrelari", 10)
jokalaria2 = Jokalaria("Beckenbauer",
"Defentsa", 4)
print(jokalaria1.txosten())
ekipoa = Taldea("New Team", 1983, 39944.45)
ekipoa.jokalariaFitxatu(jokalaria1)
ekipoa.jokalariaFitxatu(jokalaria2)
ekipoa.jokalariakErakutsi()
```

Emaitza:

```
Maradona Aurrelaria 10
Beckenbauer Defentsa 4
```

5.6 Ariketa

Sortu ondorengo klaseak definitzen dituen programa bat:

1. `Dispositiboa` klasea: `izena` eta `prezioa` atributuak ditu. Eraikitzaile bat atributuekin, *set* eta *get* eta `toString` funtzioak gehitu (atributuak erakusteko)

2. `Mugikorra` klasea: `Dispositiboa` motako azpiklasea da, eta `zenbakia` atributua gehitu behar zaio. Sortu eraikitzailea eta `def toString (self)` metodoa, superklasekoak aprobetxatuz. Gehitu `def deitu (self, zenbakia)` funtzioa, pantailatik `"Deitzen [zenbakia]"` esaten duena.

3. `Ordenadorea` klasea: `Dispositiboa` azpiklasea da, prozesadorea atributua gehitu behar zaio. Sortu eraikitzailea eta `def toString (self)` funtzioa superklasekoak aprobetxatuz Gainera, mugikorra eta ordenagailua sortzeko behar den kodea idatzi.

```
class Dispositiboa:

 def __init__(self, izena, prezioa):

    self._izena = izena

    self._prezioa = prezioa
```

```
 def getIzena():
   return self._izena
 def setIzena(izena):
   self._izena = izena
 def getPrezioa():
   return self._prezioa
 def setPrezioa(prezioa):
   self._prezioa = prezioa
 def toString(self):
 return f"{self._izena} {self._prezioa}";
class Mugikorra(Dispositiboa):
 def __init__(self, izena, prezioa, zenbakia):
super().__init__(izena, prezioa)
   self._zenbakia = zenbakia
 @property
 def zenbakia(self):
   return self._zenbakia
```

```
 @zenbakia.setter
 def zenbakia(self, zenbakia):
   self._zenbakia = zenbakia
 def toString(self):
   return f"{super().toString()}
{self._zenbakia}"
 def deitu(zenbakia):
   print("Deitzen", zenbakia)
class Ordenagailua(Dispositiboa):
 def __init__(self, izena, prezioa,
prozesadorea):  super().__init__(izena,
prezioa)
   self._prozesadorea = prozesadorea
 @property
 def prozesadorea(self):
   return self._prozesadorea
 @prozesadorea.setter
```

```
    def prozesadorea(self, prozesadorea):
        self._prozesadorea = prozesadorea
    def toString(self):
        return f"{super().toString()}
{self._prozesadorea}"


ordenagailua = Ordenagailua("Dell", 4553.4,
"Lentium 4")
telefonoa = Mugikorra("Chanmhung", 434.4,
665745345)
print("Ordenagailua:",
ordenagailua.toString())
print("Telefonoa:", telefonoa.toString())
```

Emaitza:

```
Dell 4553.4 Lentium 4 ordenagailua
Chanmhung telefonoa 434.4 665745345
```

5.7 Ariketa

`Txanogorritxo` proiektua sortuko dugu, non protagonistak janari saski bat kudeatzen duen. Janari mota askotakoa izango da. Hauek dira egin beharreko klaseak:

1. `Janaria` klasea: `izena` eta `pisua` atributuak ditu. Eraikitzaile bat atributuak, *set* eta *get* eta `toString` funtzio bat erabiltzen, atributuak erakutsiz.

2. `Fruta` klasea: `Janaria` klasearen azpiklasea da, eta `bitamina` atributua gehitu behar zaio. Sortu eraikitzailea eta `toString` funtzioa superklaseko kodea berrerabiliz.

3. `Goxokia` klasea: `Janaria` klasearen azpiklasea da, eta `kaloria` atributua gehitu behar zaio. Sortu eraikitzailea eta `toString` funtzioa, superklaseko kodea berrerabiliz.

4. `Saskia` klasea, `janariak` izeneko atributua du, eta janari klaseko (edo azpiklaseko) elementu zerrenda bat da: `Futua` eta `Goxokia`. Eraikitzailean hasten da. Hiru funtzio ditu:

`def sartuJanaria(hau, janari)` saskian janari bat sartzen du.

`def pisuGuztira(hau)` saskiko janariaren guztizko pisua itzultzen du.

`def toString(self)` saskiko janari guztia erakusteko.

Gainera, erantsi behar den kodea instantzia hauek sortzeko: `Fruta` eta `Goxokia` klaseen instantziak sortu, eta erantsi `Saskia` motako instantzia bati.

```
class Janaria:
 def __init__(self, izena, pisua):
 self._izena = izena
 self._pisua = pisua
 @property
 def izena (self):
 return self._izena
 @izena.setter
 def izena (self, izena):
 self._izena = izena
 @property
 def pisua (self):
   return self._pisua
 @pisua.setter
```

```
 def pisua (self, pisua):
   self._pisua = pisua
 def toString (self):
   return f"{self._izena} {self._pisua}"
class Fruta(Janaria):
 def __init__(self, izena, pisua, bitamina):
   super().__init__(izena, pisua)
   self._bitamina = bitamina
 @property
 def bitamina (self):
 return self._bitamina
 @bitamina.setter
 def bitamina (self, bitamina):
   self._bitamina = bitamina
 def toString (self):
   return f'{super().toString()}
{self._bitamina}'
```

```
class Goxokia(Janaria):

 def __init__(self, izena, pisua, kaloria):
   super().__init__(izena, pisua)

   self._kaloria = kaloria

 @property

 def kaloria (self):

   return self._kaloria

 @kaloria.setter

 def kaloria (self, kaloria):

   self._kaloria = kaloria

 def toString (self):

   return f'{super().toString()}
{self._kaloria}'

class Saskia:

 def __init__(self):

   self._janariak = []

 def sartuJanaria (self, janari):
```

```
    self._janariak.append(janari)
  def pisuGuztira (self):
  guztira = 0
  for janaria in self._janariak:
  guztira += janaria.pisua
  return guztira
  def toString (self):
  informazioa = ""
  for janaria in self._janariak:
    informazioa = informazioa +
janaria.toString() + "\n"  return informazioa


txintxa = Goxokia("Bomer", 0.2, 100)
gominoa = Goxokia("Marrubia", 0.3, 210)
udarea = Fruta("Udarea", 0.1, "B")
sagarra = Fruta("Sagarra", 0.15, "A")
saskia = Saskia()
```

```
saskia.sartuJanaria(txintxa)
saskia.sartuJanaria(gominoa)
saskia.sartuJanaria(udarea)
saskia.sartuJanaria(sagarra)
print("Saskiaren edukia:", saskia.toString())
print("Pisua guztira:", saskia.pisuGuztira())
```

Emaitza:

```
Saskiaren edukia: Bomer 0.2 100
Marrubia 0.3 210
Udarea 0.1 B
Sagarra 0.15 A
Pisua guztira: 0.75
```

13 Salbuespenak

Baliteke programazioan bikaina izatea. Baina, hala ere, zure programek huts egin dezakete, gauza batzuk zure programaren kontroletik kanpo daudelako. Beraz, nahiz eta zure programa zuzena izan, gauzak oker joan daitezke.

Adibidez:

- Zure programak erabiltzaileari zenbaki bat idaztea eskatzen badio, baina erabiltzaileak letrak idazten baditu ala ezer idazten ez badu, zure programak huts egingo du.
- Zure programak fitxategi bat irakurri behar badu, baina fitxategi hori existitzen ez bada, zure programak huts egingo du.
- Zure programak sarera konektatu behar badu, baina zure ordenagailua konektatuta ez badago, zure programak huts egingo du.

Ikus dezakezunez, egoera batzuetan programak ezin du kontrolik izan. Zorionez, badugu mekanismo bat aukera ematen diguna gauza horietako bat gertatzen bada gure programak huts egin ez dezan eta besterik gabe amai dadin. Eta mekanismo hori salbuespenak dira.

13.1 Salbuespenak Python-en

Adibidez, demagun honako programa oso sinple bat dugula, erabiltzaileari zenbaki bat eskatu eta biderketa bat egiten duena:

```
balioa = input ("Sartu zenbaki bat:")
balioa = int(balioa)
```

```
karratua = balioa * balioa
print ("Karratua da:", karratua)
```

Erabiltzaileak behar ez duena sartzen badu, honako hau ikusiko dugu:

```
Sartu zenbaki bat: x
Traceback (most recent call last):
File "salbuespena.py", line 2, in < module>
balioa = int (balioa)
ValueError: invalid literal for int () with
base 10: 'x'
```

Salbuespen baten bidez, programak huts egitea ekidin dezakegu, eta gutxienez erabiltzaileari errore mezu bat erakutsi. Salbuespena lengoaiaren egitura bat gehiago da, eta forma hau du:

```
try:
 huts_egin_dezakeen_kodea
except:
 huts_egitean_gertatzen_dena
```

Ikus dezagun aurreko adibidea, salbuespen blokearen barruan huts egin dezakeen kodea babestuz:

```
balioa = input ("Sartu zenbaki bat:")
try:
 balioa = int (balioa)
 karratua = balioa* balioa
 print ("Karratua da:", karratua)
except:
 print ("Errorea datua bihurtzean!")
```

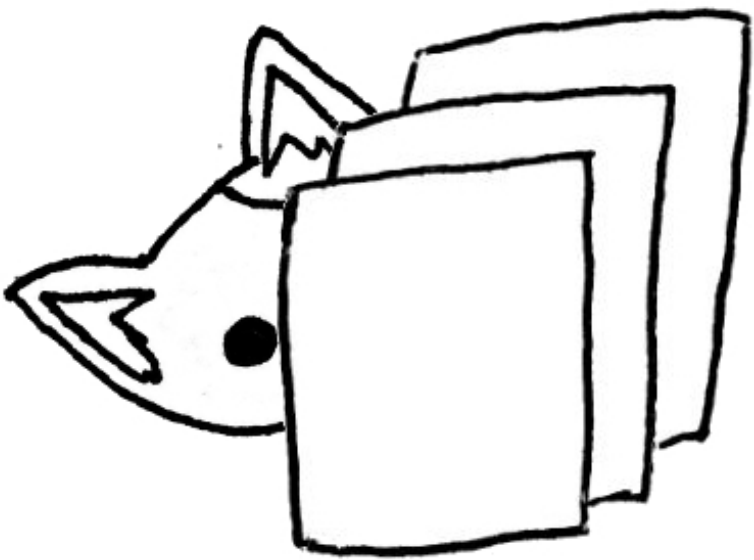
Orain, datu oker bat sartuz gero, honako hau ikusiko dugu:

```
Sartu zenbaki bat: x
Errorea datua bihurtzean!
```

Halaber, programa hobetu ahal izango litzateke, balioa berriro eskatzeko eta ez amaitzeko.

Errore motaren araberako salbuespen espezifikoak daude, eta errore-mezu zehatzagoa erakusteko erabil daitezke.

14 Fitxategien kudeaketa



Orain arte datu gutxi erabili ditugu, erabiltzaileak pantaila bidez idazten duena edo aldagaietan daukaguna. Baina datu kopuru handiagoak erabili nahi baditugu, fitxategietan irakurri eta idatz dezakegu. Era guztietako fitxategiak daude: testua, multimedia (musika, bideoa) eta fitxategi bitarrak. Horiek guztiak programa

batetik erabil daitezke. Sarrera gisa, ikus dezagun nola erabil ditzakegun testu fitxategiak.

14.1 Fitxategien irakurketa

Fitxategi bat irakurri ahal izateko, alde batetik fitxategi hori egon behar da, gero ireki eta irakurri ahal izango dugu. Kode honetan, programaren leku berean dagoen fitxategi bat irakurtzen da:

```
fitxategia = open("testua.txt", "r")

edukia = fitxategia.read()

print(edukia)

fitxategia.close()
```

Kontuan hartzekoak:

Fitxategia irakurtzeko, lehenik eta behin ireki egin behar da, honako honekin: `open("testua.txt", "r")`

Fitxategia irekitzean, haren izena adierazi behar da, eta, beste direktorio batean badago, fitxategiaren *path* edo "bidea". Adibidez `Testuak` izeneko karpetan egonez gero, hau izango litzateke bidea: `Testuak/testua.txt`.

`"r"` parametroak adierazten du fitxategia irakurketa moduan bakarrik irakurtzen dugula. Testu fitxategia horrelako zerbait izan liteke, eta programak hori bera erakutsiko luke pantailan.

```
Hau testu bat da

hainbat lerrokoa

Eta irakur daiteke

oso erraz
```

14.2 Lerroz-lerro irakurtzen?

Aurreko adibidean, bat-batean irakurri dugu fitxategiaren eduki osoa, testu aldagai batean gordeta. Baina batzuetan, baliteke fitxategia lerroz-lerro irakurri nahi izatea. Horretarako, honako funtzio hau erabili behar dugu:

```
fitxategia = open("testua.txt", "r")

lerroak = fitxategia.readlines()

for lerroa in lerroak:

 print (lerroa.strip()) # lerroaren amaieran
dagoen \n karakterea kendu

fitxategia.close()
```

14.3 JSON fitxategiak

Aurrekoa bezalako testu fitxategi sinpleek informazioa izan dezakete, baina ez dira programa baterako oso datu erabilgarriak. Programa batek erraz manipulatu ditzakeen datuak irakurri edo gorde nahi baditugu, formatu jakin bat erabiltzea komeni da. Programazioko formatu ezagunenetako bat JSON formatua da. Python-en hiztegi egituren antza duen formatua da. Python lengoaiaren zerrendak bezala ere irudikatzeko aukera badu.

Hurrengo edukia JSON formatuan dago. Hainbat objektu dituen zerrenda da. Fija zaitez, JSON motako objektuak Python-eko hiztegien berdinak dira!:

```
[
 {"id": 66, "izena": "Ada"},
 {"id": 2, "izena": "Neko"},
 {"id": 4, "izena": "Bug"}
]
```

Formatu horren alde ona da gure Python programara erraz eraman daitekeela, betiere zuzena bada, noski.

Eduki hori irakurri eta datu horiek hiztegi zerrenda bihurtzeko, `json` liburutegia erabiliko dugu. Fitxategi horren edukia automatikoki kargatu ahal izango dugu aldagai batean. Hortik aurrera, eduki hori guztia zerrenda gisa erabili ahal izango dugu, non elementu bakoitza hiztegi bat den!:

```
import json

fitxategiarea = open("testua.json", "r")

edukia = json.load(fitxategiarea)

for pertsonaia in edukia:

 print(pertsonaia["izena"])

fitxategiarea.close()
```

Pantailan, hau ikusiko dugu:

```
Ada

Neko

Bug
```

14.4 Fitxategien idazketa

Fitxategiak idazteko, prozesua antzekoa da, baina bi gauza egin behar ditugu:

- Fitxategia idazkera moduan irekitzea.
- `write` funtzioa erabili edukia idazteko.

Kode honekin, testu-lerro pare bat idatziko ditugu fitxategian:

```
fitxategia = open("testua.txt", "w")
fitxategia.write("Idatzi lerro bat\n")
fitxategia.write("Idatzi beste lerro bat\n")
fitxategia.close()
```

KONTUZ! 

Horrela idazten badugu, fitxategiaren edukia ezabatuko baitugu. Fitxategiaren edukia horrela geratuko litzateke:

```
Lerro bat idazten dut
Beste lerro bat idazten dut
```

Edukia fitxategiaren amaieran gehitu nahi badugu, fitxategia `"a"` (append) moduan ireki behar dugu:

```
fitxategia = open("testua.txt", "a")
fitxategia.write("Gehitu lerro bat\n")
fitxategia.write("Gehitu beste lerro bat\n")
fitxategia.close()
```

Orain, fitxategiaren edukia hau izango litzateke:

```
Lerro bat idazten dut
Beste lerro bat idazten dut
Gehitu lerro bat
Gehitu beste lerro bat
```

14.5 Fitxategi batean JSON idazten

JSON formatuko fitxategi baten kasuan, kontutan hartu behar da idazteko unean gure datuak testu bihurtu behar direla Zorionez, automatikoki egiten duen funtzio bat dago: `json.dumps()`

Hurrengo adibidean, json fitxategi baten edukia aldagai baten barruan kargatzen da. Gero elementu bat gehituko diogu zerrenda

horri. Fitxategia berriro irekiko dugu, idazkera moduan, eta `write` bat egingo dugu, `json.dumps` erabiliz edukia testu bihurtzeko:

```
import json

fitxategia = open("testua.json", "r")

edukia = json.load(fitxategia)

fitxategia.close()

pertsonaia = { "id": 666, "izena": "Gumball" }

edukia.append(pertsonaia)

fitxategia = open("testua.json", "w")

fitxategia.write(json.dumps(edukia))

fitxategia.close()
```

15 Liburutegiak

Programak gero eta konplexuagoak diren heinean, litekeena da funtzio asko definitu behar izatea, edo diseinua klaseetan bereizi

behar izatea, etab. Dena fitxategi berean eduki dezakegun arren, ez litzateke gure kodea antolatzeko modurik onena. Egokiena da klase bakoitza bere fitxategian bereiztea, eta funtzio edo funtzio-multzo bakoitza bere fitxategian biltzea.

Kodea fitxategietan eta karpetetan antolatu ondoren, beste fitxategi batzuetan berrerabil ditzakegu. Ikus dezagun adibide sinple bat. Funtzio hau definituko dugu `matematika.py` izeneko fitxategi batean:

```
Batuketa (a, b):

 def kendu(a, b):

   return a - b

 def batu(a):

   return a + 1
```

Orain, fitxategi hori beste programa batean sar dezakegu, `import` aginduaren bidez. Direktorio berean badaude, besterik gabe egin daiteke:

```
import matematika

balio1 = 5

balio2 = 10
```

```
emaitza = matematika.batu(balio1, balio2)

print(emaitza) # 15
```

6.0 Ariketa

Aurreko ariketa batean, pasahitzen sorgailu bat egitea proposatu zen. Erabili kode bera, baina sar ezazu fitxategi baten barruan. Sortu beste fitxategi bat kode hori inportatzeko eta erabiltzeko.

`sortu.py` fitxategia:

```
import random

def ausazkoa (max):

 return random.randint(0, max - 1)

def pasahitzaSortu (luzera):

 hizkiak =
[
"a","b","c","d","e","f","g","h","i","j","k","l
",
"m
","n","ñ","o","p","q","r","s","t","u","v","w",
"x","y","z",
"0
","1","2","3","4","5","6","7","8","9",".","-",
```

```
"_","!","$"]
pasahitza = ""
 for i in range(luzera):
 hizkia = hizkiak[ausazkoa(len(hizkiak))]
 pasahitza = pasahitza + hizkia
 return pasahitza
```

Eta fitxategian honela erabiliko genuke:

```
import sortu
pasahitza = sortu.pasahitzaSortu(8)
print(pasahitza)
```

Emaitza:

```
g3ep-ahx
```

Klaseekin gauza bera egin daiteke.

Demagun `PantailaIrakurgailua` izeneko klasea dugula `pantaila_irakurgailua.py` izeneko fitxategi batean. Kontsolatik datuak irakurtzeko aukera ematen digun klase bat da:

```
class PantailaIrakurgailua:
```

```
 def irakurZenbaki (self, mensaje = "Sartu
zenbaki bat:"):  numero = input (mezua)

   return int (zenbakia)

 def irakurTestua(self, mensaje = "Sartu
testua:")

   testua = input (mezua)

   return testua
```

Orain, klase hori beste fitxategi batean berrerabil dezakegu, gure fitxategiarekin batera.

```
import pantaila_irakurgailua

import matematika

irakurgailua.PantailaIrakurgailua()

balio1 = irakurgailua.irakurZenbaki()

print (matematika.batu(balio1))
```

Pantailan horrelako zerbait ikus daiteke:

```
Sartu zenbaki bat: 6

7
```

6.1 Ariketa

Definitu `Menu` izeneko klase bat, honako funtzio hauek dituena:

- `def __init__(self, aukerak)`: aukeren zerrenda bat jasotzen du parametro gisa.
- `def erakutsi(self)`: aurretik zenbaki bat duen aukerak erakusten ditu, print deituz.
- `def hautatu (self, zenbakia)` Itzuli `True`, baldin eta aukeratutako zenbakia menuan badago; bestela, itzuli `False`.

Gero klase hori `6.2.py` fitxategian inportatu eta erabili.

Fitxategia:

```
class Menu:
 def __init__ (self, aukerak):
   self._aukerak = aukerak
 def erakutsi (self):
   for i in range(len(self._aukerak)):
        print(f"{i+1} {self._aukerak[i]}")
```

```
    def hautatu (self, zenbakia):
        return zenbakia > 0 and zenbakia
<=len(self._aukerak)
```

6.2.py fitxategia, programa nagusia:

```
import menu

nireMenua = menu.Menu(["Erakutsi", "Kendu",
"Irten"])

nireMenua.erakutsi()

if nireMenua.hautatu(1):

 print("1. aukera badago menuan")

else:

 print("1. aukera ez dago menu honetan")
```

Emaitza:

```
1 Erakutsi

2 Ezabatu

3 Irten

Oraingo 1. aukera
```

15.1 Proposatutako ariketak

6.0 Ariketa

Sortu `Fitxategia` izeneko klase bat, honako funtzio hauekin:

- `def __init__ (self, fitxategiIzena)`: ireki beharreko fitxategi izena jasotzen du parametro gisa.
- `def irakurri (self)`: fitxategiaren edukia jasotzen duen kate bat itzultzen du.
- `def idatzi (self, edukia)`: fitxategian parametro gisa pasatzen zaion edukia idazten du. Gero, liburutegi bezala erabili behar duzu beste fitxategi batean inportatuz.

Fitxategia:

```
class Fitxategia:

 def __init__(self, fitxategiIzena):

   self._fitxategiIzena = fitxategiIzena

 def irakurri(self):

   fitxategia = open(self._fitxategiIzena,
"r")
```

```
        datuak = fitxategia.read()
        fitxategia.close()
        return datuak
    def idatzi(self, eduki):
        fitxategia = open(self._fitxategiIzena,
"w+")
        fitxategia.write(eduki)
        fitxategia.close()
```

6.0.py fitxategia, programa nagusia:

```
import fitxategia
from datetime import date
nireFitxategia =
fitxategia.Fitxategia("6.0.txt")
print("Aurreko edukia: ",
nireFitxategia.irakurri())
nireFitxategia.idatzi("Eduki aldatuta!!! " +
str(date.today())) print("Edukia:",
nireFitxategia.irakurri())
```

Testu-fitxategia:

```
Horixe da gaur egungo edukia.
```

Emaitza:

```
Aurreko edukia: Eduki aldatua!!! 2024-08-18
Edukia: Edukia aldatuta!!! 2024-08-23
```

6.1 Ariketa

Sor ezazu `Zerrenda` izeneko klase bat, honako funtzio hauekin:

- `def __init__ (self, fitxategiIzena)`: json fitxategi baten izena hartzen du parametrotzat, eta haren edukia zerrenda batean kargatu behar da. Zerrenda hori atributu gisa definituko da. Edukiak hiztegi-zerrenda bat izan behar du, formato honekin

  `[{"id": 1, "izena": "Juan"}, {...}]`

- `def existitzenDa(self, izena)`: `True` ala `False` itzuli parametro gisa pasatzen den izena zerrendan badago.

- `def xehetu (self)`: zerrendako izen guztiak letra xeheetara pasatu behar dira.

- `def posizioa (self, izena):` izen hori dagoen posizioa itzuli behar duzu.

Gero, klase hau erabili behar duzu beste fitxategi batean inportatuz.

`zerrenda.py` fitxategia:

```
import json
class Zerrenda:
 def __init__ (self, fitxategiIzena):
   edukia = open(fitxategiIzena, "r")
   self._datuak = json.load(edukia)
   edukia.close()
 def existitzenDa (self, izena):
   for datu in self._datuak:
        if datu["izena"] == izena:
             return True
   return False
 def xehetu (self):
```

```
 self._datuak = list(map(lambda datu: { "id":
datu["id"], "izena": datu["izena"].lower() },
self._datuak))

 def posizioa (self, izena):

   i = 0

   for datu in self._datuak:

        if datu["izena"] == izena:

             return i

        i += 1

   return -1

 def inprimatu (self):

   for datu in self._datuak:

        print(datu)
```

`6.1.py` fitxategia, programa nagusia:

```
import zerrenda

nire_zerrenda = zerrenda.Zerrenda("6.1.json")

badaude = nire_zerrenda.existitzenDa("eugene")
```

```
if badaude:
 print("Dago!")
nire_zerrenda.xehetu()
nire_zerrenda.inprimatu()
badaude = nire_zerrenda.existitzenDa("eugene")
if badaude:
 posizioa = nire_zerrenda.posizioa('eugene')
 print("Badago!")
 print(posizioa)
```

`6.1.json` fitxategia:

```
[
 {
 "id": 3,
 "izena": "Juan"
 },
 {
```

```
    "id": 5,
    "izena": "Eugene"
    },
    {
    "id": 10,
    "izena": "Paul"
    }
  ]
```

Emaitza:

```
{'id': 3, 'nombre': 'juan'}
{'id': 5, 'izena': 'eugene'}
{'id': 10, 'izena': 'paul'}
Badago!
1
```

6.2 Ariketa

Sor ezazu `Zereginak` klase bat, honako funtzio hauekin:

1. `def __init__ (self)`: `zereginak.json` izeneko fitxategia ireki behar duzu, eta honako formatu hau izango duten hiztegiak zerrenda batean kargatu: `{"id": 1, "zeregina"": "Ikasi zerbait"}`. Zerrenda hori atributua izango da.

2. `def sortu (self, zeregina)`: objektu berri bat sortu eta zerrendan gordetzen du.

3. `def ezabatu (self, id)`: `id` horrek duen zerrendako zeregina bat ezabatzen du.

4. `def gorde (self)`: gorde zerrenda `zeregina.json` fitxategian.

5. `def erakutsi (self)`: *string* batean itzultzen ditu zeregin guztiak.

Gero, klase hau erabili behar duzu beste fitxategi batean inportatuz.

`zereginak.py` fitxategia:

```
import json

class Zereginak:

 def __init__ (self):

    fitxategia = open("zereginak.json", "r")
```

```
    self._zereginak = json.load(fitxategia)
    fitxategia.close()
 def sortu (self, id, zeregina):
    berria = { "id": id, "zeregina":
zeregina };
    self._zereginak.append(berria)
 def gorde (self):
    fitxategia = open("zereginak.json", "w")


fitxategia.write(json.dumps(self._zereginak))
    fitxategia.close()
 def ezabatu(self, id):
    self._zereginak = list(filter(lambda datua:
datua["id"] != id, self._zereginak))
 def erakutsi (self):
    emaitza = ""
    for datua in self._zereginak:
```

```
            emaitza += json.dumps(datua) + "\n"
        return emaitza
```

`zereginak.json` fitxategia:

```
[
 {"id": 3, "zeregina": "Go ikasi"},
 {"id": 5, "zeregina": "Rust ikertu"},
 {"id": 10, "zeregina": "Lo egin"}
]
```

`6.2py` Fitxategia:

```
import zereginak
nireZereginak = zereginak.Zereginak()
print(nireZereginak.erakutsi(), "\n---")
nireZereginak.sortu(2, "Gehiago ikasi")
print(nireZereginak.erakutsi(), "\n---")
nireZereginak.ezabatu(2)
print(nireZereginak.erakutsi(), "\n---")
```

```
nireZereginak.sortu(66, "Irakurri")
print(nireZereginak.erakutsi(), "\n---")
nireZereginak.gorde()
```

Emaitza:

```
{"id": 3, "zeregina": "Go ikasi"}
{"id": 5, "zeregina": "Rust ikertu"}
{"id": 10, "zeregina": "Lo egin"}
---
{"id": 3, "zeregina": "Go ikasi"}
{"id": 5, "zeregina": "Rust ikertu"}
{"id": 10, "zeregina": "Lo egin"}
{"id": 2, "zeregina": "Gehiago ikasi"}
---
{"id": 3, "zeregina": "Go ikasi"}
{"id": 5, "zeregina": "Rust ikertu"}
{"id": 10, "zeregina": "Lo egin"}
```

```
---
{"id": 3, "zeregina": "Go ikasi"}
{"id": 5, "zeregina": "Rust ikertu"}
{"id": 10, "zeregina": "Lo egin"}
{"id": 66, "zeregina": "Irakurri"}
---
```

6.3 Ariketa

Sor ezazu `Jokalaria` izeneko klase bat, eduki hau izango duena:

- `def __init__ (self, izena, zenbakia)`: parametroak esleitzen dizkie `_izena` eta `_zenbakia`, atributuei.
- *get/set* metodoak izenarentzako eta zenbakiarentzako.
- `def info (self)`: string bat itzultzen du jokalariaren informazioarekin.

Sor ezazu `Taldea` izeneko klase bat, eduki honekin:

- `def karga(self)`: `jokalariak.json` izeneko fitxategi bat ireki behar duzu. Fitxategi horrek jokalari-hiztegien zerrenda bat izango du `[{"izena": "Pele",`

`"dortsala": 10}, {…}]` formatoarekin. Eta fitxategiaren objektu bakoitzeko, `Jokalaria` motako instantzia bat sortu behar duzu, eta `_jokalariak` izeneko zerrenda batean sartu.

- `def erakutsi (self)`: jokalarien zerrenda osoa erakutsi behar duzu.
- `def fitxaketa (self, izena, dortsala)`: jokalari berri bat sartu behar duzu zerrendan, `Jokalaria`ren instantzia bat sortuz.

`Taldea` klasean, `Jokalaria` klasea inportatu beharko duzu, erabili ahal izateko.

`jokalaria.py` fitxategia:

```
class Jokalaria:
 def __init__ (self, izena, zenbakia):
   self._izena = izena
   self._zenbakia = zenbakia
 @property
 def izena (self):
```

```
        return self._izena
    @izena.setter
    def izena (self, izena):
        self._izena = izena
    @property
    def zenbakia (self):
        return self.zenbakia
    @zenbakia.setter
    def zenbakia (self, zenbakia):
        self._zenbakia = zenbakia
    def toString (self):
        return self._izena + " " +
  str(self._zenbakia)
```

`taldea.py` taldearen fitxategia:

```
import json
import jokalaria
```

```
class Taldea:

 def karga(self):

    edukia = open("./jokalariak.json")

    jokalariak = json.load(edukia)

    print("Kargatuta:", jokalariak)

    self._jokalariak = []

    for j in jokalariak:

self._jokalariak.append(jokalaria.Jokalaria(j
["izena"], j["zenbakia"]))

 def fitxaketa(self, izena, dorsala):

    fitxategiBerria =
jokalaria.Jokalaria(izena, dorsala)
    self._jokalariak.append(fitxategiBerria)

 def inprimatu(self):

    for jokalaria in self._jokalariak:

         print(jokalaria.toString())
```

`jokalariak.json` jokalarien fitxategia:

```
[
 {
 "izena": "Maradona",
 "zenbakia": 10
 },
 {
 "izena": "Pele",
 "zenbakia": 8
 }
]
```

`6.3.py` fitxategia, programa nagusiarekin:

```
import taldea
nireTaldea = taldea.Taldea()
nireTaldea.karga()
nireTaldea.inprimatu()
nireTaldea.fitxaketa("Gento", 11)
```

```
  nireTaldea.inprimatu()
```

Emaitza:

```
Loaded: [{'zenbakia': 10, 'izena': 'Maradona'},
{'zenbakia': 8, 'izena': 'Pele'}]

Maradona 10

Pele 8

Maradona 10

Pele 8

Gento 11
```

16 Python

Zergatik aukeratu dugu Python? Bere abantaila ugariengatik. Lengoai interpretatua da, sintaxi oso erraza duena, eta ikasteko oso erraza da. Ez zara kezkatu behar (ez asko behintzat) datu motaz. Kontuan hartu behar dugun bakarra zera da: bloke bakoitzeko tabulazioak errespetatzea.

Hala ere, ez da ahaztu behar helburua ez dela programazio lengoai hau ikastea, baizik eta programazioaren funtsak ikastea; eta Pythonek zeregin hori errazten du.

Gainera, oso lengoai erabilgarria da, oso hedatua eta profesionalki erabilia. Hori gutxi balitz, programatzaile askok Python maite dute,

eta horrek kodea, liburutegi mordoa eta elkar laguntzen duten pertsonen komunitate izugarria dakar.

Pythonek bi bertsio berezi ditu, 2 eta 3. Liburu honetan 3. sintaxia eta estiloa erabili dugu, gaur egungoa delako eta 2. bertsioa desagertutzat ematen delako.

17 Python zure ordenadorean instalatzen

Web bidez landu daiteke. Baina zure ordenagailuan instalatu nahi baduzu, hemen duzu nola egin. Nahikoa da python.org gunera joatea eta 3. bertsioaren instalatzailea deskargatzea. Instalazioa pixka bat aldatzen da zure sistemaren arabera, baina, funtsean, honako hau litzateke:

- Windows: instalatzailea deskargatu, exekutatu eta urrats bakoitza berretsi.
- Mac: berdin-berdin.
- Linux: ziur aski berez instalatuta izango duzu, edo, beharbada, ez duzu azalpenik behar 😉

Behin Python instalatuta, programak idatzi eta horiek exekutatu behar dira kontsolan. Kontsolan, gure Python programa dagoen direktorio berdinean horrelako zerbait exekutatuko dugu:

```
python3 programa.py
```

edo

```
python programa.py
```

17.1 Kode editoreak

Koderako editore bat erabili nahi baduzu, aukera asko dituzu, baina honako hauek nabarmenduko ditugu:

- pycharm: (http://www.jetbrains.com/pycharm /) Editore profesionala eta doakoa. Pisutsua. [code]
- Visual Studio Code (https://vscode.io), Microsoft-ek garatutako editorea. Arina.
- pydev (http://pydev.org)
- sublime (http://www.sublimetext.com)

18 Test unitarioak

Test unitarioak kodea ondo eginda dagoela egiaztatzen duten programak dira. Funtsean, funtzioak emaitza zuzena dela egiaztatzen duten programak dira. Testak egiteko hainbat liburutegi daude, baina Python-en `unittest` lehenetsia dago, eta, beraz, ez dago ezer instalatu behar.

Demagun kalkulagailu bat errepresentatzen duen klase hau dugula:

```
class Kalkulagailua:

 def batu (self, a, b):

   return a + b

 def kendu (self, a, b):

   return a - b

 def biderkatu (self, a, b):

   return a * b

 def zatitu (self, a, b):

   return a / b
```

Mota horretako test unitarioak egiteko, nahikoa litzateke beste klase hau sortzea, eta hori `unittest` liburutegiaren testa-mota baten oinordekoa izango litzateke.

`Kalkulagailua` bera ere inportatu behar dugu, proban jarri behar baitugu. Klaseko funtzio bakoitzak kalkulagailuaren funtzio bakoitza egiaztatzen du. Nahi adina test egin daitezke, funtzioek behar dutena egiten dutela frogatzeko. Ikus dezakezunez, test bakoitzak funtzio bat exekutatzen du funtsean, eta emaitza esperotakoa dela egiaztatzen du, honako hau erabiliz:

```
  def test_batu (self):
   kal = kalkulagailua.Kalkulagailua()
   self.assertEqual(kal.batu(40, 2), 42)
```

`kalkulatzailea.test.py` fitxategi osoa honako hau izango litzateke:

```
  import unittest
  import kalkulagailua
  class TestStringMethods(unittest.TestCase):
   def test_batu(self):
     kal = kalkulagailua.Kalkulagailua()
```

```
        self.assertEqual(kal.batu(40, 2), 42)
    def test_kendu(self):
        kal = kalkulagailua.Kalkulagailua()
        self.assertEqual(kal.kendu(40, 2), 38)
    def test_biderkatu(self):
        kal = kalkulagailua.Kalkulagailua()
        self.assertEqual(kal.biderkatu(40, 2), 80)
    def test_zatitu(self):
        kal = kalkulagailua.Kalkulagailua()
        self.assertEqual(kal.zatitu(40, 2), 20)
if __name__ == '__main__':
    unittest.main()
```

Orain nahikoa da testen fitxategia exekutatzea, eta hau ikusiko dugu:

```
python3 kalkulagailua.test.py
....
```

```
-
----------------------------------------------
---------------------- Ran 4 test in 0.000s

OK
```

19 Python proiektu bati hasiera

`virtualenv` Python proiektu bat hasteko modu gomendagarri bat izango litzateke: gure sistema edozein dela ere funtziona dezakeen proiektu-karpeta bat sortzeko. Hori dela eta, proiektua malguagoa eta beste ordenagailu batzuk eramateko errazagoa da.

Komando hauek sistemaren kontsolan idatzi beharko dituzu.

```
pip3 install virtualenv

Defaulting to user installation because normal
site-packages is not writeable

Collecting virtualenv

Downloading virtualenv-20.0.31-py2.py3-none-
any.whl (4.9 MB) |. | 4.9 MB 447 kB/s

Collecting distlib < 1, > = 0.3.1

...
```

virtualenv proiektu berri bat sor dezakegu honela:

```
virtualenv proiektua
```

Eta horrek proiektua izeneko karpeta sortuko du. Jarraian, proiektuaren ingurune birtuala aktibatu behar dugu, honako hau exekutatuz:

```
source proiektua/bin/activate
```

Orain, kodea gehitu dezakegu edo dependentziak instalatu. Depententziak gure proiektuan erabiliko ditugun liburutegien zerrenda da. Horretarako, `requirements.txt` izeneko testu-fitxategi bat sortzea komeni da. Fitxategi horrek formatu hau izan behar du:

```
#Bertsio zehatz bat instalatzeko

#pakete_izena == bertsioa

#Bertsio berdina edo handiagoa instalatzeko

#pakete_izena >= bertsioa

#Bertsio berriena instalatzeko

#pakete_izena
```

Adibidez, `pygame` eta testing pakete bat instalatu nahi baditugu, honako hauek jar ditzakegu:

```
pygame = 1.9.6
```

```
unittest
```

Eta gero pakete hori eta beste batzuk instalatu genitzake, `pip3` (edo `pip`) pakete kudeatzailerarekin:

```
source bin/activate
pip3 install -r requirements.txt
```

Beste aukera bat da `pip3`ekin zuzenean liburutegiak instalatzea:

```
pip3 install pygame
```

Instalatuta dagoela egiaztatuko dugu:

```
pip3 list
Package Version
---------- -------
pip 20.2.2
pygame 1.9.6
setuptools 49.6.0
wheel 0.35.1
```

Eta `freeze` erbiliz, pakeeteak `requirements.txt` fitxategian gordeko dugu:

```
pip3 freeze > requirements.txt
```

Orain, `pygame` erabiliko duen fitxategi bat sor dezakegu:

```
import pygame

pygame.init()

# Set up the drawing window

screen = pygame.display.set_mode([500, 500])

# Run until the user asks to quit

running = True

while running:

 # Did the user click the window close button?
for event in pygame.event.get():

 if event.type == pygame.QUIT:

 running = False

 # Fill the background with white
```

```
    screen.fill((255, 255, 255))

    # Draw a solid blue circle in the center

    pygame.draw.circle(screen, (0, 0, 255), (250,
250), 75)

    # Flip the display

    pygame.display.flip()

# Done! Time to quit.

pygame.quit()
```

Honela jokatuko genuke:

```
python3 game.py
```

Eta Python-en ingurune birtuala amaitzeko, nahikoa litzateke:

```
deactivate
```

Oharra: `pygame` Python lengoaiarekin bideojokoak garatzeko liburutegia da.

Agurra

Liburua hemen amaitzen da, baina programatzen ikasteko eta proiektu berriak sortzeko abentura hasi berri da. Arazo ugari daude konpontzeko, eta tresna asko dauzkazu zure esku lanean ibiltzeko.

Gehiago ikastera animatzen zaitugu!

20 Lizentzia

Kodea ondo sartzek: programazioa ikasten, 2024

Pello Xabier Altadill Izura

CC BY 4.0 Lizentziapean

20.1 Honakoak egin ditzakezu:

•**Partekatu** — partekatu, kopiatu eta birbanatu edozein bitarteko edo formatutan edozein xedetarako, baita merkataritza-xedeetarako ere.

•**Moldatu** — nahasi, eraldatu eta horretan oinarrituz sortu edozein xedetarako, baita merkataritza-xedeetarako ere.

Lizentzia-emaileak ezin ditu askatasun horiek atzera bota, lizentziako baldintzak betetzen dituzun bitartean.

20.2 Honako baldintzen arabera:

•**Aitortu** — Aitortza egokia egin behar duzu , lizentziaren esteka eman behar duzu eta aldaketak egin diren adierazi behar duzu. Zentzuzko edozein modutan egin dezakezu hori, baina ez duzu aditzera eman behar lizentzia-emaileak zu edo zure erabilera onesten duenik.

•**Ez dago murrizketa gehigarririk** — Ezin duzu lege-baldintzarik edo neurri teknologikorik aplikatu, baldin eta lizentziak baimentzen duen zerbait legez murrizten badiete gainerakoei.

20.3 Oharrak:

Ez duzu lizentzia hau bete behar domeinu publikoan dagoen materialaren elementuen kasuan, ezta zuk egiten duzun erabilera salbuespen edo muga aplikagarri batek baimentzen duenean.

Ez da bermerik ematen. Lizentziak ez dizkizu nahitaez ematen zuk nahi duzun erabilerarako behar diren baimen guztiak. Adibidez,

publizitate-eskubideak, pribatutasun-eskubideak, eskubide moralek eta beste eskubide batzuek materiala nola erabiltzen duzun muga dezakete.