Programazioa Ikasi

Pello Xabier Altadill Izura

Beste mota batzuk dituzten klaseak
Metodo estatikoak
Proposatutako ariketak
Salbuespenak Python-en
Fitxategien irakurketa
#Lerroz lerro irakurtzen?
JSON fitxategiak
Fitxategien eskritura
Fitxategi batean idatzi json
Proposatutako ariketak
Kode editoreak
Test unitarioak
Python proiektu bati hasiera ematen tokian-tokian

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 ikasiko** benetan programatzen. Pro bihurtzeko, ordenagailua aurrean izan behar duzu eta hemen azaltzen dizuguna praktikatu. Proposatutako ariketak egiten, idazten, probatzen eta zure gustura hobetzen saiatu behar duzu. Izan ere,** programatzen ikasten da** programatuz.

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.

#Mukientzako programazioa 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 bat, edo* tableta* bat, mugikor bat,

egin zerbait zehatza, 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 baliteke

etsaiak oso bizkorrak diruditen jolasen aurka. Egia esan, ordenagailuak jokabide adimendunak imitatzen dituzten programak exekutatzen ditu.

2. oharra:

Ohiko programazio-ingurune bat instalatu nahi baduzu, zoaz X eranskinera.

#Ingurunea

Interneterako konexioa duen edozein ordenagailutan sartu:

https://repl.it /

! [Site de replt.it]
(https://raw.githubusercontent.com/pxai/mocos/master/book/images/00.png)

Hortik, `Start coding

! [Start Coding replt.it]
(https://raw.githubusercontent.com/pxai/mocos/master/book/images/01.png)

Eta hautatu** Python** lengoaia:

! [Python en replt.it hautaketa]
(https://raw.githubusercontent.com/pxai/mucos/master/book/images/02.png)

Behin hori eginda, programazio-ingurunea kargatuko da:

! [Replt.it garapen-ingurunea]
(https://raw.githubusercontent.com/pxai/mocos/master/book/images/03.png)

- Ezkerrean, 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 alta ematea gomendatzen dizugu. Horrela, egiten dituzun programa guztiak gordeta eta lokalizatuta izango dituzu.

Online aukerak:

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

#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 frogatzen baduzu, pantailatik horrelako zerbait ikusi beharko zenuke:

```
Kaixo mundua!
```

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 jarraibideen 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.
```

KONTUZ! Python hizkuntzan oso garrantzitsua da lerroaren hasieran tabulaziorik edo espaziorik ez erabiltzea (hurrengo kapituluetan ikusiko ditugun blokeak erabili ezean). Horrek huts egingo luke:

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

Iruzkinak

Programa batean, iruzkinak jar daitezke. Exekutatzen ez den testua da, ordenagailuak erabat ezagutzen ez duena. Zertarako erabiltzen da? Oro har, iruzkinak programaren zati jakin batzuk azaltzeko erabiltzen dira.

````python

#Programa honek dio Kaixo

print (“Kaixo”)
````

```
Pythonek ez dio jaramonik egiten iruzkinari, eta "Telefonoa pantailan"
erakutsiko du.

Hainbat lerrori buruzko iruzkinak ere egin daitezke:
```

````
```python

"""

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 behar ez dutenak.

#Aldagaiak

Aldagaiek datuak gordetzeko balio dute. Ordenagailu-programek, funtsean, arazo bat konpontzeko eta emaitza bat emateko datuak maneiatzen dituzte. Prozesu horretan guztian beharrezkoa da datuak gordetzea, eta horretarako aldagaiak erabiltzen dira. Aldagaiak datu-edukiontziak bezalakoak dira. Kozinatzeko erabiltzen diren edalontziak, platerak eta iturriak bezalakoak dira nolabait: zerbait dute, horrekin lan egiten da, nahastu egiten da, prozesatu egiten da eta emaitza bat lortzen da: zorte pixka batekin zerbait aberatsa lortzen da.

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

````python

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:

````python
print (izena)
````

Pantaila bidez,

```
Ada
```

Edozein unetan alda dezakegu aldagai horren balioa:

````python
izena = “Ada”
print (izena)
izena = “Neko”
print (izena)
````

```
Eta pantailan ikusiko dugu:
```

```console
Ada
Neko
```

Oharra:

Aldagai baten edukia mezuaren zati gisa ere erakuts daiteke.

Horretarako hainbat aukera daude:

Komen bidez bereizi

Nahikoa da aldagaiak eta testua komekin tartekatzea:

````python
izena = “Bug”
adina = 10
print (“Kaixo, izena dut”, izena)
print (“Naiz”, izena, “dut”, adina, “urteak”)
````

```
Pantaila bidez,
```

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

Orain zu! 0.1 ariketa

Sortu bi aldagai `izen eta edadetu, eta erakutsi haien balioa pantailan.

```
izena = "Ada"

adina = 14
print ("Zure izena da", izena, "duzu", adina, "urteak")


#Alternatiba:

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

Emaitza:

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

Formatuko stringa

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

````python

izena = “Bug”

adina = 10

print (f “Kaixo, {izena} dut izena”)

print (f “{izena} naiz {} urte ditut”)

```
Pantaila bidez,


```console
Kaixo, Bug dut izena.

Bug naiz, 10 urte ditut.
```

KONTUZ, aukera hau Python 3.6tik bakarrik dago eskuragarri

## Ehunekoa ordeztea

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

````python

print (“Kaixo, % s dut izena” % izena)

```
Pantaila bidez,


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

Gauza bera egin dezakezu hainbat aldagairekin

````python

izena = “Neko”

adina = 5
````

print (“Kaixo, % s dut izena eta % d urte ditut” % (izena, adina)

`Pantaila bakoitzeko:`

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

#Datu-motak

Datuak! Programek lantzen duten lehengaia da. Gure programek eraldatzen duten elementua 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 arrautzak datuak izango lirateke, ontziak aldakorrak 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 hizkuntza batzuetan bezala. Horregatik da hizkuntza sinpleagoa eta malguagoa. Baina ezin izango dugu nahi duguna egin datuekin.

Jarraian, oinarrizko datu motak ikusiko ditugu.

Zenbakiak

Mota guztietako zenbakiak dira:

- Osoak:

1, 2, 3, 4,…

```
kontagailua = 10

adina = 12
```

- Hamartarrekin:

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

```
pisua = 34.67

prezioa = 242.9943
```

- Negatiboak:

0 zenbaki txikiagoak marratxo batekin adierazten dira: -4, -5, -3.1415,…

```
oharra = -5

tenperaturaMarten = -50.676
```

Testua

Testua, kateak edo* strings* ere deituak, komatxo bikoitz edo sinpleen arteko edozein hitz-multzo da.

```
izena = "Ada"

esaldia = "

hitzak = 'Lanera noa'

novia = '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.

- Lineako jauzia

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

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

honela erakutsiko da:

``shell

Kaixo,

zer moduz

```
Hainbat lerrotako testu bat ere defini daiteke:


```python

Igande bat zen.

arratsaldean

autoetara joan nintzen

talka "" "
```

- Tabulazioak

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

```
esaldia = "Izena\Abizena\Adina"
```

honela erakutsiko da:

``shell

Izena Abizena Adina
```

```
Beste karaktere berezi batzuk:


-\\Testu batean kontra-barra erakusteko.

-\"Komatxo bikoitza erakusteko testu batean.

-\'Komatxo sinple bat erakusteko testu batean.

-\a Txistua jotzeko.


## Boolearrak

Boolear motak bi balio posible bakarrik izan ditzake:
Datuak, hau da, egiazkoak edo okerrak. Programazioan funtsezko datua
da, erabakiak hartzeko erabiltzen baita.


```python

amaituta = False

esMayor = True

pythonMola = True
```

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:

```
etsaiak = []

[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"
```

Izendapenen 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.

None

Arraroa dirudien arren, programetan batzuetan hutsa, ezereza irudikatzen duen zerbaitekin hitz egin behar izaten da.
```

Badago hitz bat Python-en ezereza irudikatzeko aukera ematen diguna, eta hori da:

```
Hasierako balioa = None

dato = None
```

Egia esan, ez da aldagaiak sortzeko erabiltzen. `Nonoporek balio bat adierazten du egoera berezietan. Adibidez, informazioa fitxategi bat edo erabiltzaileak ematen ez digun datu bat ez dagoen leku batetik atera nahi badugu.

0.2 ariketa

Idatzi hemen ikusitako mota bakoitzaren aldagai bat definituko duen programa bat, eta kontsola bidez erakutsiko ditu.

```
izena = "Ada"

adina = 42

pisua = 101.54

vivo = True

aberastasunak = None

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


print (izena)

print (adina)

print (pisua)

print (bizirik)

print (aberastasunak)

print (lagunak)
```

Emaitza:

```
Ada

42

101.54

true

null

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

#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, lehenik eta behin datu bat eskatu beharko du programak.

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 funtzio hau erabiltzen da:

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

print ("Kaixo", izena)
```

Pantailan honako hau ikusiko zenuke:

```
Esadazu zure izena:
```

Input funtzioaren ondorioz, pantailan agertzen da “Zure izena adierazi:” mezua. Era berean, programa gelditu egiten da, erabiltzaileak zerbait idatzi arte.

Erabiltzaileak «Kontsola ezazu» idazten badu, honela ikusiko da:

```
Esadazu zure izena: Rosa

Kaixo, 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?
```

Kontuz datuekin

Erabiltzaileak zerbait idazteko,*** delako 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:

```
Emadazu zenbaki bat: 4

44
```

`4 + 4 'gehitu eta`8 “erakutsi beharrean, `4' eta`4 ’batu egin ditu, izan ere,” 4 “testua irakurri duenean,” 4 “testua da.

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, inputatu egin behar da:

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

bikoitza = balioa + balioa

print (bikoitza)
```

Orain bai:

```
Emadazu zenbaki bat: 4

8
```

0.4 ariketa

Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat eta gehitu 10. Jarraian, kontsola bidez erakutsi behar duzu emaitza. 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
```

Beste bihurketa batzuk

Hizkuntzan aldagai motak adierazi behar ez baditugu ere, motak (testua, zenbakia, etab.) kontuan hartzen dira programa gauzatzerakoan.

Adibidez, zenbaki bat duen aldagai bat badugu eta testu batean kateatu nahi badugu, errore bat lortuko genuke:

```
balioa = 66

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

Errorea hau izango litzateke:

```
TypeError: cannot concatenate 'str' and 'int' objects
```

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 zenbaki bihurtzen du hamartarrekin.
- bool (): balio bat boolear batera bihurtzen du. `bool (“False”) Itzuli egingo luke.

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

#Operadoreak Programek zenbakiekin kalkuluak egin behar dituzte, datuak prozesatu, balioen arabera erabakiak hartu. Horretarako operadoreak behar ditugu.

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 igaro zara.

Hauek dira programazioko oinarrizko eragiketak:

- Batuketa: `+
- Kenketa:
- Biderketa:
- Dibisioa:

Adibidez, egun batek zenbat segundo dituen kalkulatzeko:

```
minutuak = 60

segundoak = 60

orduak = 24


Segundoak = segundoak* minutuak* orduak
```

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

```
ada = 14

bug = 10

neko = 2

batez bestekoa = (ada + bug + neko)/3
```

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
```

Modulua eta esponentziala

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-balioa hauxe izango da:

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

```
balioa = 2

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

Emaitza `8 “izango litzateke.

#Zeinu-aldaketa

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

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

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

```
tenperatura = -11

kontua = 200


tenperatura = -tenperatura # 11

kontua = -kontua # -200
```

#Operadore laburtuak

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

```
kontagailua = 0

kontagailua = kontagailua + 2
```

Horrelako egoeretan, operadore** 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 handitu egingo duen programa bat. Jarraian, kontsola bidez erakutsi behar duzu emaitza. Ondoren, balioa murriztu behar du eta emaitza kontsola bidez erakutsi. Operadore laburtuak erabili!

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

balioa + = 1


print ("Gehikuntza da", balioa)


balioa – = 1


print ("Beherakada da", balioa)
```

Emaitza:

```
Sartu zenbaki bat: 6

Gehikuntza 7 da

Beherakada 6 da
```

Konparazio-operadoreak

Balio bat bestearekin alderatzeko aukera ematen diguten operadoreak dira. Normalean zenbakiekin erabiltzen da, eta eragiketa horien emaitza `Trueedo `False«da.

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

```
balioa = 5

emaitza = balioa = = 5
```

Emaitza hauxe izango litzateke: `Trueu.

Hona hemen konparazio-operadoreak:

- Berdin:
- Desberdina:
- Baino handiagoa:
- Honako hauek baino txikiagoa:

- Handiagoa edo berdina: `> =
- Txikia 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: `Trueu.

0.7 ariketa

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

```
valor1 = input ("Sartu zenbaki bat:")

valor2 = input ("Sartu beste zenbaki bat:")


emaitza = 1 balorazioa! = 2


print ("Desberdinak dira?", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 42

Sartu beste zenbaki bat: 42

Desberdinak dira? Falsea
```

Operadore boolearrak

Operadore boolearrek aukera ematen digute eragiketak egiteko balio boolearrekin `Trueedo `Falseere.

#and

Operadore horrek `Trueu 'itzuliko du, baldin eta bi operadoreak ereTrueu:

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

Emaitza hauxe izango litzateke: `Trueu.

Aukera guztiak laburbiltzeko, horri esaten zaio operadorearen egiaren taula.

a | | b | emaitza |
– | – | – | – | |
`Helburua |`Argazkia | `Faialdea |

Helburua |Arduria | Txurruna |Falseere |
Leharetxea |Danera | Faialdea |Falseere |
Erresuma |Txurruna | Trueu |Trueu |

0.8 ariketa

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

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

balioa = int (balioa)

emaitza = (balioa > = 0) and (balioa % 2 = = 0)


print ("Parea eta positiboa al da?", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 14

Parea eta positiboa da? True
```

#or

Operadore horrek honako hauek ere itzultzen ditu: `Trueu, baldin eta** edozein**, eta hauek ere bai:

```
balioa = 5

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

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

a | | b | emaitza |
– | – | – | – | |
Helburua |Ordua | `Ordua |
Orduera |Ordua | Txurruna |Trueu |
Errueu |Orduera | Faialdea |Trueu |
Erresuma |Ordua | Txurruna |Trueu |

0.9 ariketa

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

```
valor1 = input ("Sartu zenbaki bat:")

valor2 = input ("Sartu beste zenbaki bat:")


emaitza = (int (valor1) > = 0) or (int (valor2) > = 0)


print ("Bietako bat positiboa al da?", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: -4

Sartu beste zenbaki bat: 6

Bietako bat positiboa da? True
```

#not

Operadore horrek kontrako balioa itzultzen du eragiketan. Aplikatzen bazaio, itzuli `Frueere 'ri, eta aplikatzen bazaio, itzuli`Falseere' ri:

```
balioa = True

emaitza = not balioa;
```

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

a	emaitza
albistea	Helburua	`False«
oharra	Turruna	`Helburua	

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)


positiboaYPar = (balioa > = 0) and (balioa % 2 = = 0)

emaitza = not positiboaYPar

print ("Ez al da parea eta positiboa?", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: -4

Ez al da bikoitia eta positiboa? True
```

#Operadoreak konbinatzen

Operadoreak behar adina konbina ditzakegu:

```
erretiratua = 65

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

print ("Lan egin dezakezu")
```

Oro har, baldintzapeko eragileak baldintzapeko blokeen, begizten eta abarren baldintzen barruan erabiltzen dira.

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 da:", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 3

Biderketa hau da: 21
```

0.1 ariketa

Idatzi erabiltzaileari zenbaki bat eskatu eta bitan banatuko 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

Banaketa hau da: 30
```

0.2 ariketa

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

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

emaitza = int (balioa) % 3


print ("Modulua da:", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 7

Modulua hau da: 1
```

0.3 ariketa

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

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

emaitza = int (balioa)** 2


print ("Esponentziala da:", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 4

Esponentziala hau da: 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 zenbaki bat:")

resta = int (balioa) - 5

emaitza = -resta


print ("Azken kenketa da:", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 4

Hau da azken kenketa: 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 kenketa 0 bada, zenbaki hori bikoitia dela esan nahi du.

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

modulua = int (balioa) % 2


emaitza = modulua = = 0


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

Emaitza

```
Sartu zenbaki bat: 8

Kemena 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 al da?", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: 6

Positiboa da? True
```

0.7 ariketa

Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat. Ondoren, lehenengoa 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 al da?", emaitza)
```

Emaitza:

```
Sartu zenbaki bat: -3

Negatiboa da? True
```

Baldintzak

Uneren batean, 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, eta aukera ematen digute kode bat baldintza batzuk betetzen direnean bakarrik exekutatzeko.

if

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

```
if* baldintza*:

* jarraibideak*

* jarraibideak*

*...*
```

Ikus dezakezunez, baldintza batekin hasten da. Baldintza boolear batek itzultzen duen edozein adierazpen izan daiteke, hau da, `Trueu edo`Falseu, egiazkoa edo faltsua.

Hala bada, bete egingo dira jarraibideak, eta, bestela, salto egingo da.

Adibidez:

```
balioa = -2

if balioa < 0:

print ("Balioa 0 baino txikiagoa da")

print ("Programaren amaiera")
```

Honela geratuko litzateke:

```
Balioa 0 baino txikiagoa da

Programaren amaiera
```

Aldiz:

```
balioa = 5

if balioa < 0:

print ("Balioa 0 baino txikiagoa da")

print ("Programaren amaiera")
```

Honela geratuko litzateke:

```
Programaren amaiera
```

Oharra:

Oso garrantzitsua den zerbait ere kontuan hartu behar duzu: “ifi” aren barruko jarraibideak 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. Modu horrek irakurketa errazten du eta programa baten egitura erraz ezagutzea ahalbidetzen du beste programatzaile batzuentzat. Baita zuretzat ere, zure programa bada.

1.0 ariketa

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

```
balioa = input ("Idatz ezazu zenbaki bat:")


if int (balioa) < 0:

print ("Negatiboa da")
```

Emaitza:

```
Idatz ezazu zenbaki bat: -5
Negatiboa da
```

if else

IFarekin bloke bat sor dezakegu, baldintza bat betetzen bada bakarrik exekutatzen dena. Baina zer gertatzen da programak baldintza baten arabera gauza bat edo bestea egitea nahi badugu?

“Beste” aukera sartu ahal izateko, if-else egitura bat erabiltzen dugu:

```
if* baldintza*:

* jarraibideak*

LKse:

* jarraibideak*
```

Adibidez:

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

if izena! = "":

print ("Kaixo", izena)

LKse:

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

Horrelako zerbait ikus liteke, erabiltzaileak sartzen duenaren arabera:

```
Esadazu zure izena: Ada

Kaixo Ada
```

Baina erabiltzaileak ezer idatzi gabe “jakin” sakatu besterik ez badu egiten:

```
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 ("Idatzi testu bat:")


if testua = = "agurra":

print ("Kaixo!")

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

Emaitza:

```
Idatzi testu bat: ez dakit

Ez dut ulertzen.
```

if elif else

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

```
if* 1 baldintza*:

* jarraibideak*

LKif:* 2. baldintza*:

* jarraibideak*

elif* baldintza3*

* jarraibideak*

LKse:

* jarraibideak*
```

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

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


if idioma = = "Español":

print ("Kaixo"):

elif idioma = = "Inglés":

print ("Hello"):

elif idioma = = "Francés":

print ("Salut")

LKse:

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

Behar adina `hautetsi ’izan ditzakegu.

1.2 ariketa Idatzi erabiltzaileari testu bat eskatzen dion programa bat. Testua “bihar” bada, “Egun on” mezua erakutsi behar duzu, testua “berandu” bada, “Arratsalde on” mezua erakutsi behar duzu, eta, bestela, “Gabon” mezua erakutsi behar duzu.

```
testua = input ("Idatzi testu bat:")


if testua = = "bihar":

print ("Egun on. ")

elif testua = = "arratsaldea":

print ("Arratsalde on. ")

elif testua = = "gaua":

print ("Gabon. ")

LKse:

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

Emaitza:

```
Idatzi testu bat: arratsaldez

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.

```
numero1 = input ("Sartu zenbaki bat:")

numero2 = input ("Sartu beste zenbaki bat:")


if 1 zenbakia > 2 zenbakia:

print (1. zenbakia, "baino handiagoa da", 2. zenbakia)

LKif 1. zenbakia < 2. zenbakia:

print (1. zenbakia, "baino txikiagoa da", 2. zenbakia)

LKse:

print (1. zenbakia, "berdin da", 2. zenbakia)
```

Emaitza:

```
Sartu zenbaki bat: 5

Sartu beste zenbaki bat: 10

5 10 baino txikiagoa da
```

Proposatutako ariketak

1.0 ariketa

Sortu programa bat erabiltzaileari bi balio oso eskatuko dizkiona eta pantaila bidez erakutsiko duena lehenengoa bigarrenaren multiploa den. Zenbaki bat bestearen multiploa den jakiteko, moduluaren eragiketa egin behar duzu (% `u) haien artean: 0 bada, multiploa izango da.

```
numero1 = input ("Sartu zenbaki bat:")

numero2 = input ("Sartu beste zenbaki bat:")


gainerakoa = int (1. zenbakia) % int (2. zenbakia)


if hondarra = = 0:

print (1. zenbakia, "multiploa da", 2. zenbakia)

LKse:

print (1. zenbakia, "EZ da multiploa", 2. zenbakia)
```

Emaitza:

```
Sartu zenbaki bat: 40

Sartu beste zenbaki bat: 4

40 4ren multiploa da
```

1.1 ariketa

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

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

kopurua = int (zenbakia)


if zenbakia > = 0:

print (zenbakia, "positiboa da")

LKse:

print (zenbakia, "negatiboa da")


numero = −numero


print ("Bihurketa:", zenbakia)
```

Emaitza:

```
Sartu zenbaki bat: −6

− 6 negatiboa da

Bihurketa: 6
```

1.3 ariketa

Idatzi urteko hilabete bateko zenbakia eskatzen duen programa bat eta erakutsi zenbat egun dituen. Hilabete ezezagun bat sartuz gero, “Ez dakigu” mezua erakutsi.

```
mes = input ("Sartu urteko hilabete bat:")


if mes = = "Urtarrila":

print ("31 egun")

elif mes = = "Otsaila":

print ("28/29 egun")

elif mes = = "Martxoa":

print ("31 egun")

elif mes = = "Apirila":

print ("30 egun")

elif mes = = "Maiatza":

print ("31 egun")

elif mes = = "Ekaina":

print ("30 egun")

elif mes = = "Uztaila":

print ("31 egun")

elif mes = = "Abuztua":

print ("31 egun")

elif mes = = "Iraila":

print ("30 egun")

elif mes = = "Urria":

print ("31 egun")
elif mes = = "Azaroa":

print ("30 egun")

elif mes = = "Abendua":

print ("31 egun")

LKse:

print ("Hilabete ezezaguna")
```

Emaitza:

```
Sartu urteko hilabete bat: Ekaina

30 egun
```

1.3 ariketa

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

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

kopurua = int (zenbakia)


if zenbakia > = 0 and % 2 = = 0:

print (zenbakia, "parea eta positiboa da")

elif zenbakia < 0 and % 2! = 0:

print (zenbakia: "bakoitia eta negatiboa da")

LKif zenbakia < 0:

print (zenbakia, "negatiboa da")

LKse:

print (zenbakia, "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); emaitza erakutsi behar du eta gero mezu bat erakutsi: GMI 16 baino txikiagoa bada, “Jan behar duzu” mezua agertuko da.

GMI (> =) 16 eta 25 (<) artean badago, “Ondo zaude” mezua agertuko da.

GMI 25 eta 30 artean badago (<), mezua agertuko da: “Gehiegizko pisua duzu”.

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)


imc = pisua/(altuera* altuera)


imc = (imc* 10000)

print ("Zure imc:", imc)


if imc < 16:

print ("Gehiago jan behar duzu")

elif imc > = 16 and imc < 25:

print ("Ondo zaude")

elif imc > = 25 and imc < 30:

print ("Gehiegizko pisua duzu")

LKse:

print ("Obesitate arazo bat duzu")
```

Emaitza:

```
Sartu zure pisua: 70

Sartu zure altuera: 172

Zure imc: 23.66143861546782

Ondo zaude
```

1.5 ariketa Sortu erabiltzaileari jokalari-dortsal bat eskatzen dion programa bat eta egin hau: 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 1 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("Atari gizona")
  elif zenbakia >= 1 and zenbakia <= 5:
      print("Defentsa jokalaria")
  elif zenbakia >= 6 and zenbakia <= 8 or zenbakia == 11:
      print("Erdilari jokalaria")
  elif zenbakia == 9:
      print("Aurrelari jokalaria")
  else:
      print("Norbera")
else:
    print("Errorea, zenbakia ez dago 0 eta 99 artean")
```

Emaitza:

```
Dortsala sartu: 11

Erdilaria
```

#Begiztak

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

Ordenagailu baten zeregin ohikoenetako bat jarraibide 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.

Begizta bat ikus dezakezu, errusiar mendi bat bezala, non buelta batzuk ematen dituzun.

Jarraian, begizta mota desberdinak ikusiko ditugu.

Bucle while

Baldintza bat betetzen den bitartean, “whilee” begizta exekutatu egiten da. Egitura oso sinplea du:

```
while* baldintza*:

* jarraibideak*
```

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 bukle infinitua sortuko genuke. Programa ez litzateke inoiz amaituko eta betiko trabatuta geratuko litzateke!

2.0 ariketa

Sortu 0an hasitako kontagailu-aldagai bat definituko duen programa bat. Ondoren, idatzi while begizta bat. Kontagailua 5 baino txikiagoa den bitartean, erakutsi mezu hau: `Estoy en el bucle«eta gehitu» kontagailu 1.

```
kontagailua = 0

while kontagailua < 5:

print ("Begiztaren barruan nago")

kontagailua = kontagailua + 1
```

Emaitza:

```
Begiztaren barruan nago

Begiztaren barruan nago

Begiztaren barruan nago

Begiztaren barruan nago

Begiztaren barruan nago
```

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

```
izena = "

while nombre = = "":

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 gutxieneko bat:")

min = int (min)


max = input ("Sartu gehienez:")

max = int (max)


while min < max:

print (min)

min = min + 1
```

Emaitza:

```
Sartu gutxieneko bat: 3

Sartu gehienez: 8

3

4

5

6

7
```

Begizta For

Begizta for ekintza bat errepikatzeko erabiltzen dugu. Baldintza bat baino gehiago, kontagailu moduko bat erabiltzen du:

```
for∗ aldakorra∗ in∗ tartea∗:

∗ jarraibideak∗
```

Adibidez, honako begizta honek “hola” mezua 4 aldiz erakutsiko du:

````
````python:

for i in range (4):

print (“Kaixo”)

Hona hemen emaitza:

```code

Kaixo

Kaixo

Kaixo

Kaixo
````

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

````
````python:

for i in range (4):

print (“Kaixo”, i)

Eta emaitza egiaztatu:

```code

Kaixo 0

Kaixo 1

Kaixo 2

Kaixo 3
````

Hori baliagarria izan dakiguke 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))

#Alternatibak:

#print (" % d x % d = % d" % (balioa, i, balioa* i))
```

Emaitza:

```
Sartu zenbaki bat: 3

0 x 3 = 0

1 x 3 = 3

2 x 3 = 6

...
```

2.3 ariketa Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat. 0 edo txikiagoa bada, zerbait handiagoa sartzeko adierazi behar duzu, eta bestela, “Python mola!” mezua erakutsi behar duzu pantailan, zenbakiak adierazten duen adina aldiz:

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

kopurua = int (zenbakia)


if zenbakia < = 0:

print ("0 baino gehiago sartu behar duzu")

LKse:

for i in range (0, zenbakia):

print ("Python mola!")
```

Emaitza:

```
Sartu zenbaki bat: 3

Python mola!

Python mola!

Python mola!
```

#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:

````python:

for i in range (5, 9):

print (“Kaixo”, i)

```
Hona hemen emaitza:
```

```code

Kaixo 5

Kaixo 6
Kaixo 7

Kaixo 8
```

Hirugarren parametro bat ere adieraz daiteke, balio batetik bestera nola salto egiten den adierazteko. Adibidez, 2tik 2ra:

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

Hurrengo adibidean, begizta zenbaki bikoitiekin egingo litzateke.

````python:

for i in range (0, 6, 2):

print (“Kaixo”, i)

```
Hona hemen emaitza:
```

```code

Kaixo 0

Kaixo 2

Kaixo 4
```

# #Atzerantz

Begizta atzerantz ere ibil daiteke, jauzi negatiboa eginez:

````python:

print (“Atzerako kontaketa hasten:”)

for i in range (5, 0, -1):

print (i)
````

Hona hemen emaitza:

```code
5
4
3
2
1
```

2.4 ariketa

Idatzi erabiltzaileari zenbaki bat eskatzen dion programa bat. Ondoren, 0tik erabiltzaileak sartu duen zenbakira arteko zenbaki bikoiti guztiak erakutsi behar ditu. Erabili `formako ’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
```

#Zerrenden gaineko begiztak

Begizta `indartsuak ’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 (objektuak)
```

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

```
izarra
perretxikoa
lorea
```

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 horren arazoa da, nahiz eta `Aneere ’aurkitu, begiztak zerrendaren amaierara arte jarraituko duela. Zerrenda hori oso handia bada, gure programa ez da eraginkorra izango! Hasieran esan bezala, ordenagailuak oso tentelak dira. Gelditzeko esaten ez badiegu, aurrera jarraituko dute.

Zorionez, begiztetan break jarraibidea 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 zehazten duen programa bat. Ondoren, sortu begizta for bat, zerrenda gainditu eta zenbakiak adierazten duen beste aldiz errepikatuko duena.

```
errepikatzea = [3, 6, 2]


for aldiz in errepikatu:

for i in range (batzuetan):

print ("Python")
```

Emaitza:

```
Python

Python

Python

Python

...
```

Noiz erabili while edo for? Biekin gauza bera egin zenezakeen arren, bakoitzak erabilera logikoagoa du.

Begizta hau argi eta garbi erabiltzen da zerbait exekutatu nahi denean, ez gehiago, ez gutxiago. Edo, aurrerago ikusiko dugun bezala, datu-egiturak hasieratik amaierarainoko zerrendetan sartu nahi direnean.

Baldintzak oso zehatzak ez direnean erabil daiteke `whilee ’begizta. Adibidez, erabiltzaileak datu bat sartzea nahi badugu, begizta batean sar dezakegu. Begizta ez da amaituko erabiltzaileak datu on bat sartu arte (hori litzateke baldintza).

Proposatutako ariketak

2.0 ariketa

Idatzi programa bat “whilee” begizta batekin, erabiltzaileari izen bat eskatzeko (adibidez, “Ada”) eta “Kaixo Ada” izen hori agurtzeko. “Irten” testua sartuz gero, buklea 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: salir

Amaiera.
```

2.1 ariketa

Idatzi programa bat, erabiltzaileari zenbaki bat eskatzen dion eta zenbakia 0 ez den bitartean amaitzen ez duen “whilee” begizta duena. Zenbakia sartu ondoren, agur bat erakutsi behar da zenbakiak adierazten duen beste aldiz. Zenbakia 0 baino txikiagoa bada, begizta “break” batekin amaitu behar da;

```
balioa = ''


while valor! = 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 den eta, gainera, bikoitia den. Hala bada, bistaratu pantailan zenbakiaren balioa adina aldiz karaktere* Erabili `print (“*“)

Adibidez, 8 sartuz gero, hau erakutsiko du:

****

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

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

kopurua = int (zenbakia)


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

print ("0tik gorako zenbaki bikoitia sartu behar duzu")

LKse:

izarrak = "

while zenbakia > 0:

izarrak = izarrak + "*"

numero = numero - 1


print (izarrak)
```

Emaitza:

```
Sartu zenbaki bat: 6

*****
```

2.3 ariketa

Sortu aurrekoaren antzeko proiektu bat, pare positiboak bakarrik onartu behar dituena, baina erakutsi behar duzun ildoak alderdi hau izan behar du:

2:

6: 00-** -* -*

Eta beti hemen amaitu behar da:

Adibidez, 4 zenbakia sartzen badute:

- -* -

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

kopurua = int (zenbakia)


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

print ("0tik gorako zenbaki bikoitia sartu behar duzu")

LKse:

sekuentzia = ""

kopurua = kopurua/2

while zenbakia > 0:
sekuentzia = sekuentzia + "* -"

numero = numero - 1


sekuentzia = sekuentzia + "*"


print (sekuentzia)
```

Emaitza:

```
Sartu zenbaki bat: 8

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

2.4 ariketa

Erabiltzaileari zenbaki oso bat eskatzen dion programa bat sortzen du, eta balio hori erabiliz, «marraztu» egin behar du kontsolan «*» lauki bat.

Adibidez, 4 sartuz gero, hau erakutsiko da:

****

****

****

****

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

kopurua = int (zenbakia)


if zenbakia < = 0:

print ("0 baino gehiago sartu behar duzu")

LKse:

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, 5eko faktoria 5 x 4 x 3 x 2 x 1 = 120 izango litzateke.

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

kopurua = int (zenbakia)


if zenbakia < = 0:

print ("0 baino gehiago sartu behar duzu")

LKse:

factorial = kopurua

while zenbakia > 1:

numero = numero - 1

factorial = factorial* numero


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

Emaitza:

```
Sartu zenbaki bat: 4

Emaitza: 24
```

2.6 ariketa

Sortu erabiltzaileari zenbaki oso bat eskatzen dion programa bat eta egiaztatu zenbaki hori lehengusua den ala ez, hau da, berez edo 1egatik zatigarria den.

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

kopurua = int (zenbakia)


if zenbakia < = 0:

print ("0 baino gehiago sartu behar duzu")

LKse:

zatigarria = False

originala = zenbakia

numero = numero – 1


while zenbakia > 1 and not zatigarria:

if originala % zenbakia = = 0:

zatigarria = True


numero = numero – 1



if not divisible:

print (jatorrizkoa, "lehengusua da. ")

LKse:

print (jatorrizkoa, "EZ da lehengusua. ")
```

Emaitza:

```
Sartu zenbaki bat: 5

5 lehengusua da.
```

2.7 ariketa

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

```
for i in range (11):

for j in range (11):

print (i, "x", j, "=", i* j)


#Gauza bera, bestela

for i in range (11):

for j in range (11):

print (" % d x % d = % d" (i, j, i* j))
```

#Datu-egiturak

Orain arte datu sinpleekin, zenbaki bat, testu bat eta abar duten aldagaiekin jolasten aritu gara. Baina badira datu konplexuagoak sortzeko aukera ematen diguten beste mota batzuk ere. Ez dira zailak, zenbaki soil 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.

Zerrendak

Zerrendak zenbakiz indexatutako datu-multzo bat dira. Hori da definizio oso formala, 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 aurkibide baten bidez eskura ditzakezula elementuak:

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

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

Zerrenda honela adieraz 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"]
```

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]

batura = 0.0


for zenbaki in zenbaki:

batuketa = batuketa + zenbakia


batez bestekoa = batuketa/len (zenbakiak)


print ("Batez bestekoa da:", media)
```

Emaitza:

```
Batez bestekoa: 5.06000000000005
```

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]

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

Edo azken balioak

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

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

Edo, besterik gabe, azkena:

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

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

Elementuak gehitu eta ezabatu

Zerrenda bati elementu bat gehitu nahi badiogu, nahikoa da funtzio hau erabiltzea:

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

idiomas.append ("Italiera")

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

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

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

hizkuntza [1]

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

Aurkibidea erabiliz ere egin daiteke zerrenda. Horretarako, `rangea 'funtzioa erabili beharko dugu, eta zerrendaren luzera eman beharko diogu,`lena' 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 hizkuntza batzuetan, zerrendei "array" esaten zaie.

Ezagutu beharko zenituzke, baina berriro gogoratuko dizugu.

3.2 ariketa

Definitu izenen zerrenda bat, erakutsi pantaila bidez. Gehitu elementu bat eta erakutsi zerrenda pantailaz pantaila. Ondoren, zerrendatik elementu bat ezabatu eta zerrenda pantailaz pantaila erakusten du.

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


print (izenak)


nombres.append ("Miranda")


print (izenak)


izenak [1]


print (izenak)
```

Emaitza:

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

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

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

Hiztegiak

Hiztegiak datu-multzoak dira, eta 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 baita gakoa 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": 66555333, "Bug": 111000111}


print (telefonoak)


for izena in telefonos.keys ():

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

Emaitza:

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

Bug 111000111

Ada 66555333
```

Elementu berriak gehitzeko, balio berri bat esleitu daiteke:

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


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


adinak ["Miranda"] = 23;


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


["Bug"]


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


for izena in edades.keys ():

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

3.4 ariketa

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

Amaieran, hiztegiko elementu guztiak erakusten ditu.

```
telefonoak = {"Ada": 66555333, "Bug": 111000111}


izena = input ("Sartu izen bat:")
telefonoa = input ("Sartu zenbaki bat:")


telefonoak [izena] = int (telefonoa)


for izena in telefonos.keys ():

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

Emaitza:

```
Sartu izen bat: Neko

Sartu zenbaki bat: 333222000

Ada 66555333

Neko 333222000

Bug 111000111
```

Hiztegiko balioak ezabatu ditzakegu, honako funtzio honekin:

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

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

["Bug"]

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

Eta hiztegiko balio guztiak ezagutu 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 edades.keys ():

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

Pantailaren arabera:

```
Ada 14

10. bug.

Neko 2
```

3.5 ariketa Erabili aurreko ekitaldiko telefonoen hiztegia. Eskatu erabiltzaileari izen bat, eta gero kendu elementu hori hiztegitik.

Amaieran, hiztegiko elementu guztiak erakusten ditu.

```
telefonoak = {"Ada": 66555333, "Bug": 111000111}


izena = input ("Sartu izen bat:")


telefonoak [izena]


for izena in telefonos.keys ():

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

Emaitza:

```
Sartu izen bat: Bug

Ada 66555333
```

Oharra:

Beste hizkuntza batzuetan, hiztegiei `hash 'edo`hashtables’ esaten zaie.

Datu konbinatuen egiturak

Oinarrizko egiturek, hala nola zerrendek eta hiztegiek, zerrendak eta hiztegiak ere izan ditzakete!

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 izan 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 bezeroa izeneko hiztegi-zerrenda bat zehazten duen programa bat, gakoak dituena: izena, emaila 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
```

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? Nola erabiliko duzun. Batzuetan denak ibili beharko dituzu, beste batzuetan elementu jakin bat eskuratu beharko duzu… zure programak eskatzen duenaren arabera, egitura jakin bat diseinatu beharko duzu.

#Textos

Testu-datu mota, katea edo* string* ere deitua, funtsezkoa da programetan. Horregatik, erabilgarritasun asko ditu datu mota horiek errazago erabiltzeko.

Jarraian, testuetarako baliagarriak diren zenbait funtzio ikusiko ditugu, baina, aurretik, testuari buruzko zerbait azaltzea komeni da:

Testuak zerrendak dira

Izan ere, ordenagailuarentzat, testu bat zerrenda bat bezalakoa da. Letra-zerrenda edo -katea, eta honela tratatu dezakegu:

```
testua = "Kaixo naiz Ada"

testua [1] # "o"
```

Testua zerrenda bat balitz bezala ere ibil dezakegu:

```
testua = "Ada"


for caracter in texto:

print (karaktere)
```

Irteera hau izango litzateke:

```
A

d

a
```

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

```
testua = "Neko maulla"

len (testua) # 11
```

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

```
testua = "Python mola"

testua [0:6] # "Python"

testua [7:12] # "mola"
```

Lehen karaktereak ere atera daitezke

```
testua = "Python mola"

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

Edo azken karaktereak

```
testua = "Python mola"

testua [-4:] # "mola"
```

Edo, besterik gabe, azkena:

```
testua = "Python mola"

[-1] # "a" testua
```

Letra larriak/Letra xeheak

Zenbait funtzio ditugu testua letra larriz edo xehez idazteko:

```
testua = "Irakasle Ada"

testu.upper () #IRAKASLE ADA

testu.lower () #Irakasle Ada
```

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

```
testua = "hau esaldi bat da"

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

split: testutik zerrendara

split funtzio interesgarria da testu bat zatika zatitzea eta zerrenda bihurtzea:

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

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

Besterik ezean, splita testuaren espazioetan mozten da. Baina beste edozein bereizle adieraz daiteke, adibidez:

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

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

Bilaketak

Batzuetan, testu baten barruan hitz bat bilatzea gustatuko zaigu. Horretarako, find funtzioa erabil dezakegu. Hitza aurkituz gero, hitz hori zein posiziotan hasten den erakutsiko du. Aurkitzen ez baduzu, itzuli `-1.

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

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

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

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

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

```
hitzak = "Python hizkuntza ona da"

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

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

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

```
hitzak = "Python hizkuntza ona da"

acaba = palabra .endswith ("aje") #True

acaba = palabra .startswith ("ajes") #False
```

Soberakinak ezabatu

Testuak zuriuneekin edo ezabatu nahi ditugun beste karaktere batzuekin has edo buka daitezke, esaterako, 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 () delakoarekin 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 = "--Arloi-testua"

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

Fitxategi batetik edo kontsolatik testua irakurtzen dugunean ere, lerro-jauziak egiten ditugu:

```
testua = "Honek lerro-jauzi bat du\n"

garbia = testua.rstrip ("\n")
```

Proposatutako ariketak

3.0 ariketa

Idatz ezazu 5 zenbakiko zerrenda bat hasten duen programa bat (0ra hasitakoak), eskuz hasitako 5 izeneko beste bat eta 5 boolear balioko beste bat (false erara hasitakoak)

```
izenak = ["Frodo", "Sam", "Merrin", "Pippin"]

booleanos = [True]* 5

zenbakiak = [0]* 5


print (izenak)

print (zenbakiak)

print (boolearrak)
```

Emaitza:

```
[Frodo, Sam, Merrin, Pippin]

[0, 0, 0, 0, 0]

[True, True, True, True, True]
```

3.1 ariketa

Idatzi 10 zenbakiko zerrenda bat zehazten 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:

[i] = 0 zenbakiak


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 bat erakusteko: (1) elementua sartu, (2) ezabatu, (3) erakutsi eta (4) irten. Menua erabiltzaileak 4. aukera sartzen ez duen bitartean erakutsi behar da. 1. aukera aukeratuz gero, edozein balioko push egingo da, 2 aukeratuz gero, pop egingo da, eta 3 aukeratuz gero, zerrendako edukia agertuko da.

```
zenbakiak = []

aukera = -1


while aukera! = "4":

print ("Aukeratu aukera")

print ("1. Elementua sartu. ")

print ("2. Elementua atera. ")

print ("3. Erakutsi zerrenda. ")

print ("4. Irten. ")

aukera = input ("Aukeratu aukera:")


if "1":

ugari.append (0)

elif "2":

ugari.pop ()

elif "3":

print (zenbakiak)

elif "4":

print ("Beste batera arte")

LKse:

print ("Aukera ezezaguna")
```

Emaitza:

```
Aukeratu aukera

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 kateatzen dituen programa bat, erabiltzaileak puntu bat idatzi arte (.). Orduan programak sortutako esaldia erakutsi beharko du. Erabiltzaileak ez badu ezer idazten, ez da ezer kateatu behar.

```
esaldia = ''

hitza = ''


while palabra! = ".":

hitza = input ("Idatzi hitz bat:")


if hitza! = "." or hitza! = "":

esaldia = esaldia + "" + hitza


print ("Sortutako esaldia:", esaldia)
```

Emaitza:

```
Idatzi hitz bat: Kaixo

Idatzi hitz bat: zer

Idatzi hitz bat:

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 horrekin guztiarekin, interpolazioa edo kate-txantiloiak erabiliz.

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

lekua = input ("Idatzi zure jaioterria:")

data = input ("Idatzi zure jaioturtea:")


mezua = f "{izena} duzu izena {lekua} (e) n jaio zinen {data}"


print (mezua)
```

Emaitza:

```
Idatzi zure izena: Ada

Idatzi zure jaioterria: Teverga

Idatzi zure jaioturtea: 2006

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

3.5 ariketa

Idatzi erabiltzaileari esaldi bat eskatzen dion programa bat. Gero esaldi horretako hitz bat eskatu behar du, eta, ondorioz, programak esaldi bera itzuliko du letra larriz:

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

hitza = input ("Idatzi esaldi horretako hitz bat:")


posicion = frase.index (hitza)


if posicion! = -1:

hasiera = esaldia [0: posizioa]

amaiera = esaldia [posizioa + len (hitza): len (esaldia)]

emaitza = f "{hasiera} {palabra ra.upper ()} {amaiera}"


print (emaitza)

LKse:

print (hitza, "ez dago esaldian. ")
```

Emaitza:

```
Idatzi esaldi bat: Zein ona den profa Ada

Idatzi esaldi horretako hitz bat: ona

Zein ona den profa Ada
```

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)


#aldaera:

for i in range (len (izenak)):

print (izenak [i])
```

Emaitza:

```
Frodoa

Merrin

Sam

Pip

Bilbo
```

3.7 ariketa

Sortu 10 zenbaki osoko zerrenda bat zehazten duen programa bat. Gero, beste begizta bat sortzen du, elementu bakoitza handitu eta erakusteko.

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


for i in range (len (zenbakiak)):

[i] = zenbakiak [i] + 1


print (zenbakiak)



#Batuketarako alternatiba:

#Gehituak = ugari.map (zenbakia = > zenbakia + 1)
```

Emaitza:

```
[4, 6, -3, 3, 2, 5, 1, 7, 10, 9, 4]
```

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

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

errepikatua = False

i = 0

j = 0


while i < len (zenbakiak) and not errepikatua:

while j < len (zenbakiak):

if zenbakiak [i] = = zenbakiak [j]:

errepikatua = True

break

j = j + 1

i = i + 1


if errepikatua:

print ("Zenbaki bat errepikatuta dago")

LKse:

print ("Ez dago zenbaki errepikaturik")
```

Emaitza:

```
Zenbaki errepikatua dago
```

3.9 ariketa

Sortu 10 zenbaki osorekin hasitako zerrenda bat definituko duen programa bat. 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

zeroak = 0


for zenbaki in zenbaki:

if > 0 zenbakia:

positiboak = positiboak + 1

LKif zenbakia < 0:
negatiboak = negatiboak + 1

LKse:

zeroak = zeroak + 1


print ("Positiboak:", positiboak)

print ("Negatiboak:", negatiboak)

print ("Zeroak:", zeroak)
```

Emaitza:

```
Positiboak: 8

Negatiboak: 2

Zeroak: 1
```

3.10 ariketa

Sortu 5x10 elementuko bi dimentsioko zerrenda bat definituko duen programa bat. Zerrendako balioak ausazko 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 ("Aurkitu 15 in", i,)
```

3.11 ariketa

Sortu 10 zenbaki osorekin hasitako zerrenda bat definitzen duen proiektu bat. Begizta batean, pantaila bidez erakusten ditu elementu guztiak. Gero, beste begizta bat sortzen du, eta, bertan, elementuak aztertzen ditu, indizeetan aurreko ekitaldiko “random,” metodoa 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)):

aurkibide aleatorioa = random.randint (0, len (zenbakiak) - 1)

aurrekoa = zenbakiak [i]

[i] = zenbakiak [ausazko aurkibidea]

zenbakiak [ausazko aurkibidea] = aurrekoa


print (zenbakiak)
```

Emaitza:

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

#Funtzioak

Funtzioak programa txikiak dira programen barruan. Funtzio horrek pantailan agur bat besterik ez du ateratzen:

```
def agurra ():

print "Kaixo"
```

Ikus daitekeenez, funtzio bat definitzeko, `def, 'hitza erabiltzen da, eta, ondoren, funtzioaren izena, kasu honetan,`osasuna' eta "()" parametroen zerrenda, kasu honetan hutsik dagoena.

Funtzioaren gorputzean, nahi ditugun jarraibideak jar ditzakegu.

Eta Pythonen estiloarekin jarraituz, funtzioaren barruan dagoen kodearen aurretik tabulazioa edo espazioak daudela ikusten du.

Funtzio hori definitu ondoren, funtzio hori erabiltzen dugun bakoitzean, funtzio horretan dagoen kodea exekutatuko da:

```
agurra ()
```

Horrek erakutsiko du:

```
Kaixo
```

4.0 ariketa

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

```
def dias ():

print ("Egun on")


def tardes ():

print ("Arratsalde on")


def noches ():

print ("Gabon")


egunak ()

arratsaldez ()

gauak ()
```

Emaitza:

```
Egun on.

Arratsalde on.

Gau on.
```

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 saluda (pertsona):

print ("Kaixo", pertsona)
```

Funtzioaren barruan, pertsonen balioa desberdina izango da deian ematen zaionaren arabera.

Honela erabil liteke:

```
agur ("Neko") #pertsona izango da "Neko"

agur ("Ada") #pertsona izango da "Ada"
```

Hona hemen horren emaitza:

```
Kaixo Neko

Kaixo Ada
```

4.1 ariketa

Idatz ezazu programa bat “laukia” izeneko funtzio batekin, eta parametro hau izango du: Funtzioak emaitza biderkatu eta pantaila bakoitzean erakutsi behar du.

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



karratua (3)
```

Emaitza:

```
9
```

#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 agur (pertsona, batzuetan = 3):

for i in range (batzuetan):

print ("Kaixo", pertsona)
```

Honela deitzen bazaio:

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

Ikusiko litzateke:

```
Kaixo Neko

Kaixo Neko
```

Aldiz, bigarren parametroa pasatzen ez badugu, `vecesek` lehenetsitako balioa hartuko du:

```
agurtu ("Bug")
```

Hau litzateke:

```
Kaixo Bug

Kaixo Bug

Kaixo Bug
```

Parametro infinituak

Python-en funtzioei esker, parametro kopuru zehaztugabea pasatzen diezu. Arraro samarra dirudi, baina baliagarria da.

Imajinatu funtzio bat sortu nahi duzula, pasatzen dizkiozun**** parametro guztiak batuko dituena.

```
def batu (* balioak):

emaitza = 0


for balioa in balioak:
emaitza = emaitza + balioa


erreturn emaitza
```

Erreparatzen badiguzu, `baloreak ’parametroa zehaztu dugu, aurretik zenbaki bat jarrita. Horrekin esan nahi dizugu ez dela parametro bakarra, edozein luzera izan dezakeen zerrenda baizik.

Beraz, funtzio horri honela dei diezaiokegu:

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

batu (2, 45) # 47
```

4.2 ariketa

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

```
def batu (* hitzak):

esaldia = ''


for hitza in palabras:

esaldia = esaldia + "" + hitza


return esaldia



print ("Kaixo", "Zer", "Halakoa")
```

Emaitza:

```
Kaixo, zer moduz?
```

Itzulera

Funtzioek beharrezkoak diren eragiketa guztiak egin ditzakete, baina emaitza bat itzultzen dutenean erabilgarriagoak dira.

Horretarako, “itzuli” jarraibidea erabiltzen da, eta horrek datu jakin bat edo datu-egitura bat baino ezin du itzuli.

Adibidez, bi balioren batura kalkulatzen duen funtzio bat:

```
def suma (a, b):

emaitza = a + b

erreturn emaitza
```

Pixka bat laburtu daiteke zuzenean:

```
def suma (a, b):

return a + b
```

Honela erabil liteke funtzioa:

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

Gauza pare bat hartu behar dituzu kontuan, honako honekin:

1- Behin `itzuli ’egiten denean, programa funtziotik ateratzen da.

2- Funtzioan `itzuli ’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: “berdin dio eta”. Funtzioak bi zenbakien arteko zatiketa itzuli behar du.

```
def zatitu (a, b):

return a/b
```

```
print (zatitu (32, 4))
```

Emaitza:

```
8
```

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

```
def multiplika (kopurua, aldiz):

if (batzuetan > 0):

guztira = 1

for i in range (batzuetan):

guztira = guztira* kopurua

LKse:

return 0


biderkadura (2, 3) # 8
```

Oharra: aurreko funtzioa argiago geratzen da horrela.

```
def multiplika (kopurua, aldiz):

if (batzuetan < 1): return 0


guztira = 1

for i in range (batzuetan):

guztira = guztira* kopurua
```

Habia-deiak

Ez izan funtzio-deiei habia egiteko beldurrik. Oso gauza naturala da programazioan.

Imajinatu erabiltzaileari zenbaki bat eskatzeaz arduratzen den funtzio hau dugula eta sartutako zenbaki-ontzia itzultzen duela:

```
def lee zenbakia ():

numero = input ("Idatzi zenbakia:")

return int (zenbakia)
```

Demagun beste funtzio bat ere badugula zenbakiak kentzeko:

```
def restar (a, b)

return a - b
```

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

```
print (restar (leeNumero (), leeNumero ()))
```

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

Funtzio horiek `irakurtzenZenbakia () “, balio batzuk itzuliko dituzte.
Lehenengo”IrakurriZenbakia ()” deitu bezain laster, honela izango da:

```
print (kendu (5, irakurriZenbakia ()))
```

Eta gero, bigarrenari “IrakurriZenbakia ()” deitu ondoren:

```
print (kendu (5, 2)
```

Gero, kenketa egingo da eta balio bat itzuliko du:

```
print (3)
```

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

```
3
```

4.4 ariketa

Idatzi bi funtzio. Zenbaki bat jaso eta zenbaki bera positiboan itzultzen duena.

Emaitza positiboa (balioa)

Eta bi zenbaki jaso eta horien arteko potentzia kalkulatzen duen beste funtzio bat.

`potentzia (balorazioa 1, balorazioa 2)

Bigarren funtzioari deitu behar diozu, parametro hauek kontuan hartuta:

`potentzia (positiboa (-5), positiboa (4)

Oharra: ez erabili aurretik zeuden funtzioak.

```
def positiboa (balioa):
if balioa < 0:
return -balioa

Itzuli balioa



def potentzia (1, 2 balorazioak):
return valor1** valor2

print (positiboa (-1))
print (potentzia (2, 3))
print (potentzia (positiboa (2), positiboa (4)))
potentzia (positiboa (-5), positiboa (4))
```

Emaitza:

```
1
8
16
```

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.

0 arrazoia: zatitu eta irabaziko duzu Programak arazoak konpontzen saiatzen dira, konponbide bat eskainiz. Batzuetan arazoak oso konplexuak izan daitezke aurre egiteko. Funtzioek arazo horiei zatika heltzeko aukera ematen dizute. Funtzio bakoitzak arazoaren zati baterako konponbidea eman diezazuke. Beraz, arazoa arazo txiki askotan bana dezakezu eta arazo bakoitza funtzio batekin konpon dezakezu.

Jarduneko kodea idaztea programa konplexuagoak diseinatzeko lehen urratsa da.

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:

```
dato = ''


while dato = = "":

dato = input ("Mesedez, sartu datu bat:")

if dato = = "":

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.

3. araua ere aplika dezakezu. Zerbait errepikatu behar duzun hirugarren aldian, automatizatu egin behar duzu.

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)
```

Hau litzateke:

```
Holi

Holi

Holi

Aupa
```

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 benetako lana kodea denboran zehar mantentzea da.

Gure funtzioak ondo zehaztuak baditugu,* muuuuucho* lana aurreztuko dugu.

4. arrazoia: testak egiteko aukera ematen du

Agian gazte samarra zara honetarako, baina nik bezala gure programak probatzen ditugu. Zer esan nahi du horrek? Gure programek behar dutena egiten dutela egiaztatzea helburu bakarra duten programak idazten ditugula.

Zure kodeak funtzioak baditu, testak idatzi ahal izango dituzu funtzio horiek behar dutena egiten dutela egiaztatzeko.

Berez, aditua zarenean, zure eginkizuna bera baino lehen testa idaztea da zurea!

Nola egin funtzio onak

Edonork idatz ditzake funtzioak eta kodea zati txikitan taldekatu. 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 kanpoan dagoen ezer aldatu behar. Ezustekorik izan nahi ez baduzu, funtzio batek ez luke programaren barruan bildu behar.
- Funtzio batek zerbait itzuli beharko luke, eta horrek beti berdina izan beharko luke parametro jakin batzuetarako.

Proposatutako ariketak

4.0 ariketa

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

```
def diferentzia (1. balorazioa, 2. balorazioa):

diferentzia = 0


if balorazioa > 2. balorazioa:

diferentzia = 1. balorazioa - 2. balorazioa

LKse:

diferentzia = 2. balorazioa - 1. balorazioa


print ("Diferentzia da:", diferentzia)


aldea (10, 5)

aldea (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 kalkula (1, 2, op):

if op = "+": return valor1 + valor2

elif op = "-": return valor1 - valor2

elif op = = "*": return valor1* valor2

elif op = = "/": return valor1/valor2

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 saludo (momentoDelDia)

Funtzio honek parametro gisa eguneko une bat hartzen du: “goiz”, “arratsalde” edo “gau”, eta dagokion agurra itzuli behar du: “Egun on”, “Arratsalde on”, eta “Gau on”, hurrenez hurren.

```
def agurra (unea):

if momento = = "mañana": return "Buenos días"
elif momento = = "tarde": return "Buenas tardes"

elif momento = = "noche": return "Gabon"

else: return ""


print (agurra ("arratsaldez"))


def saludo2 (unea):

return {

"Bihar": "Egun on",

"Arratsaldez": "Arratsaldeon",

"Gaua": "Gabon"

} [unea]


print (osasuna2 ("arratsaldez"))
```

Emaitza:

```
Arratsalde on.

Arratsalde on.
```

4.3 ariketa

`hasiZenbakiarekin (zenbakiak, zenbakia)»

Funtzio honek array zenbakien elementu guztiak parametro gisa pasatzen dugun zenbakiarekin hasi behar ditu.

```
defEkinZenbakiarekin (luzera, zenbakia):

zenbakiak = []

for i in range (luzera):

ugari.append (kopurua)

zenbakien erreturn


def hasiConNumero1 (luzera, zenbakia): return [zenbakia]* luzera


print (hasiZenbakiarekin (10, 3))

print (hasiConNumero1 (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 aleatorioa (gehienez) Funtzio horrek 0 eta parametro gisa pasatzen den gehieneko balioaren arteko ausazko zenbaki bat itzuli behar du.

```
import random


def aleatorioa (max):

return random.randint (0, max)


print (ausazkoa (5))
```

Emaitza:

```
3
```

4.5 ariketa

Izena (silabak)

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

```
import random


def aleatorioa (max):

return random.randint (0, max - 1)



def crearIzena (silabak):

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

kontsonanteak = ["b", "c", "d", "f", "g", "h", "j", "k", "l", "m", "n",

izena = ''


for i in range (silabas):

kontsonantea = kontsonanteak [ausazkoa (len (kontsonanteak))]

bokala = bokalak [aleatorioa (len (bokalak))]

izena = izena + kontsonantea + bokala


return izena



print (sortu izena (3))
```

Emaitza:

```
xamozu
```

4.6 ariketa

`def crearPassword (length)

Luzera bat emanda, zorizko karaktereak dituen string bat sortzen duen funtzioa. Karakteredun* string* array bat erabil dezakezu, eta ausazko karaktereak atera arraytik, izen bat sortzeko. Ausazko zenbakiak sortzeko:

```
import random


def aleatorioa (max):

return random.randint (0, max - 1)


def crearPassword (luzera):

karaktereak = ["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):

caracter = karaktereak [ausazkoa (len (karaktereak))]

password = password + caracter


return password



print (sortuPassword (8))
```

Emaitza:

```
_! 5_flg $
```

4.7 ariketa

Sortu `faktura 'izeneko funtzioa (produktuak, kopuruak, prezioak), tamaina bereko hiru array jasotzen dituena, honako eduki hauekin:` 1.produktuak: produktuen izenak.

2. `kantitatea: zenbaki osoak, kantitatea adierazita.
3. `prezioa: zenbaki hamartarrak, produktu bakoitzaren prezioa adierazita.

Funtzioak produktu bakoitza zeharkatu eta guztizko prezioa kalkulatu behar du, kantitatearen eta prezioaren arabera. Produktu bakoitzaren guztizko prezio hori erakutsi behar da, eta programaren amaieran, guztizko prezioa erakutsi behar da.

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

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

guztira = 0


for i in range (len (produktuak)):

faktura = faktura + produktuak [i]

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

faktura = faktura + ":" + str (prezioak [i]) + "\n"


guztira = guztira + (kopuruak [i]* prezioak [i])


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

faktura = faktura + "Guztira:" + str (guztira)


fakturaren erreturn



#Deiaren adibidea

Faktura, guztira = faktura (["Ogia", "Arrautzak", "Irina"], [1,6,2], [1.

print (guztiraFaktura)
```

Emaitza:

```
FAKTURA

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

Ogia x 1: 1.2

Arrautzak x 6: 0.2

Irina x 2: 0.8


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

Guztira: 4.0
```

4.8 ariketa

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

`def crearAtributos (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 balio behar du puntu oso bereiziak edo berdinduak banatzen diren adierazteko, balio desorekatuena zenbat eta handiagoa izan, hau da, atributuen arteko aldea handiagoa da; nola egin programatzailearen kontua da.

Azkenean, programak esleitutako puntuen laburpena erakutsi behar du.

```
import random


def aleatorioa (max):

return random.randint (0, max)


def crearAtributuak (mailaKonpentsazioa):

darPuntosA = 0

adimena = 0

indarra = 0
abiadura = 0


gainerako puntuak = 20

puntu = 0


while puntuak > 0:

if mailaKonpentsazioa > puntuakGainerakoak:

puntuak = puntuakGainerakoak

Gainerako puntuak = 0

LKse:

puntuak = ausazkoa (mailaKonpentsazioa+1)

gainerakoak = puntuakGainerakoak – puntuak


darPuntosA = ausazkoa (3)


if darPuntosA = = 0:

adimena = adimena + puntuak

elif darPuntosA = = 1:

indarra = indarra + puntuak

elif darPuntosA = = 2:

abiadura = abiadura + puntuak


print ("\nKonpentsazio bidez esleitutako balioak:", mailaKonpentsazioa)

print ("Adimena:", adimena)

print ("Indarra:", indarra)

print ("Abiadura:", abiadura)
```

```
Atributuak sortu (3)
```

Emaitza:

```
Konpentsazio bidez esleitutako balioak: 3

Adimena: 3

Indarra: 1

Abiadura: 10
```

#Motak

Klaseek objektuei orientatutako programazioa izeneko teknika aplikatzeko aukera ematen digute. Arazo konplexuak konpontzeko beste estrategia bat da.

Funtzioekin, arazo bat arazo txikietan banatzen dugu. Aldiz, objektuetara bideratutako programazioarekin, arazoa klasetan banatzen saiatzen gara. Baina nola? Arazoaren zati den guztia irudikatuz, klaseak erabiliz.

Demagun Mario Kart bezalako lasterketa-joko baten programa egin behar dugula. Objektuetara bideratutako 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.

Nola sortu klaseak

Klase bat programazio-egitura bat da, entitate bat bere propietate eta metodoekin irudikatzeko aukera ematen diguna. Hau da, klase bat: - Propietateak ditu: berezko aldagaiak.

- Gauzak egiten ditu: funtzioak.

Adibidez, hurrengo klaseak katu oso sinple bat irudikatzen du, maullatzeko funtzio batekin:

```
class Katua:

def maulla (self):

print ("Miau")
```

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

Bestalde, kontuan izan behar duzu** guztiek** klase baten funtzioek parametro `self ’berberak izan behar dituztela, nahiz eta ez erabili. Parametro hori klaseari berari dagokio, eta haren propietateei eta funtzioei erreferentzia egiteko erabiltzen da, orain ikusiko dugun bezala.

Klasea vs instantzia

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

```
gato = Katua ()

gato.maulla ()
```

Pantailan honako hauek ikusiko ditugu:

```
Miau
```

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

5.0 ariketa

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

```
class Pertsona:

def dormir (self):

print ("ZZZZZ...")


def jan (self):

print ("Ñam, Ñam...")


def saludar (self):

print ("Kaixo, zer moduz!")



pertsona = Pertsona ()


pertsona.lo egitea ()

pertsona.jatea ()

pertsona.agurtu ()
```

Emaitza:

```
ZZZZZ...

Ñam, Ñam...

Kaixo, zer moduz?
```

Eraikuntza-funtzioa

Orain dela gutxi definitu dugun katu hori. Izen bat emango diogu. Gainera, funtzio berezi bat sortuko dugu, `**init**_ ’izenekoa.

*** eraikuntza-funtzioa* esaten zaio. Funtzio hau klaseko objektu bat sortzen denean deitzen da, eta, beraz, klasearen propietateei ekiteko lekurik egokiena da:

```
class Katua:

def __init__ (self, izena):

self.izena = izena


def maulla (self):

print ("Miau, naiz", self.izena)
```

Orain, katu klaseko objektuak sortzen ditugunean, izen bat emango diogu, eta hau jabego gisa geratuko da:

```
gato = Katua ("Pixi")

gato.maulla ()


Beste katu bat = Katua ("Txeto")

beste katea.maulla ()
```

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, `osasuna 'izeneko propietate batekin. Jabetza hori "kaixo" hitzarekin hasiko da.

Horrez gain, `decirHola«izeneko metodo bat erantsiko duzu, eta, pantailan, osasunaren edukia erakutsiko du.

```
class Kaixo:

def __init__ (self):

self.saludo = "Kaixo"


def decirHola (self):

print (self.agurra)



hola = Hola ()

hola.decirHola ()
```

Emaitza:

```
Kaixo
```

Herentzia

Herentzia kodea berrerabiltzeko klaseek duten mekanismo bat da. Demagun katakume bat ordezkatzen duen eskola bat egin nahi dugula. Katu klaseak egiten duen gauza bera egitea nahi dugu, baina, gainera, zurrunga egitea.

Txakurkume klaseak Katu klasetik heredatu ahal izango luke, honela:

```
class Cachorro (Katua):

def ronea (self):

print ("Purrrr")
```

Orain honako hau egin dezakegu. Txakurkume objektu bat sortzea, Katu klasearen propietate berberekin. Automatikoki jarauntsiko ditu jabetza (izena) eta metodo hau:

```
katutxoa = Txakurkumea ("Lucifur")

gatito.ronronea ()

gatito.maulla ()
```

Honela ikusiko litzateke:

```
Purrrr

Miau, Lucifur naiz.
```

#super ()

Azpiklase bat edo klase bat beste baten alaba sortzen duzunean, heredatzen duzun ikasgelatik `super () “funtzioa erabil dezakezu oinordetzan hartutako klasearen funtzioei deitzeko.

Adibidez, aurreko kasuan, `Cachorroere 'azpimailatik berezko eraikitzaile bat gehi genezake, eta`Gatoere’ superklasearen eraikitzaileari ere dei diezaiokegu:

```
class Cachorro (Katua):

def __init__ (self, izena):

super () .__init__ (izena)

print ("Gatete sortua")


def ronea (self):

print ("Purrrr")
```

5.2 ariketa

Idatz ezazu `Comidaa 'klasea definituko duen programa bat, izendapenaren atributuarekin. Sor ezazu`Frutak ‘izeneko azpiklase bat, `Comida`kizena eta bitaminak `jasotzen dituen eraikitzaile batekin, eta`infoagne’ izeneko metodo bat, bere informazio guztia itzultzen duena.

Sortu instantzia bat `Frutaa ’klasea probatzeko.

```
class Janaria:

def __init__ (self, izena):

self.izena = izena



class Fruta (Janaria):

def __init__ (self, izena, bitaminak):

super () .__init__ (izena)

self.vitaminak = bitaminak


def info (self):

#return f "{self.nombre} {self.vitaminas}";
return self.nombre + "" + str (self.vitaminak)


postre = Fruta ("Kiwi", [A "," C "])

print (postre.info ()
```

Emaitza:

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

Kapsularatzea

Katuaren adibidean, zuzenean sar daiteke izena jabetzara.

Horretarako, objektuetan nahikoa da horrelako zerbait jartzea:

```
objektua. IzenaJabetza
```

Katuak izena izeneko propietate bat du.

```
miGato = Katua ("Pixi")

print (miGato.nombre) #Pixi

miGato.nombre = "Pixel"

miGato.maulla () #Miau, Pixel naiz
```

Propietate bat hain zuzenean eskuratzea ez dago gaizki, baina ni bezalako programatzaile onak klasea kapsulatzen saiatzen gara. Zer esan nahi du horrek? Ezin direla zuzenean haren propietateak edo metodoak sartu edo aldatu. Klasea erabiltzen dutenentzat beharrezkoa dena bakarrik. 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:

```
@nombre.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 ():

return self._izena


@nombre.setter

def izena (izena):

if izena! = "":

self._izena = izena


def maulla (self):

print ("Miau, naiz", self.izena)
```

Ikus ezazu propietatea `izena`del orain. Jabetza hori “pribatua” dela adierazteko modu bat da, eta ez litzatekeela ikasgelatik kanpo eskuratu behar.

Orain, “Izena” izeneko klasea erabiltzen dugunean, funtzio berri horien bidez egingo da.

```
miGato = Katua ("Pixi")

print (miGato.nombre) #deitu `def izena () metodora:

miGato.nombre = "Pixel" #deitu `def izena (izena) metodora:

miGato.maulla () #Miau, Pixel naiz
```

5.3 ariketa

Idatz ezazu `Vehiculoere 'klasea definituko duen programa bat, "matrikula" atributuarekin, get/set metodoekin eta "arrancar«" izeneko beste metodo batekin. Sortu "Zortzi" izeneko azpiklase bat,`Vehicullo ‘klasea zabaltzeko, eta eraikitzaile batek `matrikulak, modeloak eta koloreak’ jasoko ditu, eta informazio guztia itzultzeko funtzio bat izango du. Sortu instantzia bat “Zortzi” klasea probatzeko.

```
class Ibilgailua:

def __init__ (self, matrikula):

self._matricula = matricula


@property

def matrikula (self):

return self._matrikula


@matricula.setter

def matrikula (self, matrikula):

self._matricula = matricula

def arrancar (self):

print ("Abiatzen", self._matrikula)



class Autoa (Ibilgailua):

def __init__ (self, matrikula, modeloa, kolorea):

super () .__init__ (matrikula)

self._modelo = modelo

self._color = color


def info (self):

return f "{self.matricula} {self._modelo} {self._color}";



auto = auto ("0042ASI", "Opel Corsa", "zuri")

martxan jartzeko kotxea ()

print (kotxe.info ())
```

Emaitza:

```
0042ASI abiarazten

0042ASI Opel Corsa Blanco
```

Zer abantaila izan dezake kapsularatzeak?

Funtsean, “kanpotik” ezin da ikasgela kontrolik gabe manipulatu. Horregatik da kutxa beltz bat bezalakoa, bideo-kontsola bat bezalakoa. Joko bat jokatzeko eskuz ireki eta soldatu beharko bazenu, ziurrenik kontsola kargatuko zenuke. Horregatik, aparatuak kutxa beltz gisa diseinatzen dira, kanpotik manipulatzeko aukera baino ez dizute ematen.

Katu klasearen kasuan, ez dugu uzten izena zuzenean aldatzen. “Setter” funtzioaren bidez, esleitu nahi den izena zuzena dela kontrola dezakegu.

Beste mota batzuk dituzten klaseak

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

Adibidez, 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._estudiantes = []


def meteAlumno (self, ikaslea):

self._alumnos.append (ikaslea)


def pasarLista (self):

for ikaslea in self._ikasleak:

print (ikaslea_izena)


alumno1 = Ikaslea ("Gumball")

alumno2 = Ikaslea ("Darwin")


gela = Gela ()


ikasgela.meteIkaslea (1. ikaslea)

ikasgela.meteIkaslea (2. ikaslea)

aula.pasaLista ()
```

Diseinuak behar bezain konplexuak izan daitezke behar duguna irudikatzeko.

5.4 ariketa

Idatz ezazu programa bat, `Piloto«atributua» eta «get/set» funtzioak dituena. Era berean, sor ezazu`Aeroplana» izeneko klase bat, eredu, pilotu eta kopilotu atributuarekin, get/set funtzioekin modelorako, eta beste metodo batekin, «volard» izenekoa. Sortu instantzia bat bi klaseak probatzeko.

```
class pilotua:

def __init__ (self, izena):

self._izena = izena


@property

def izena (self):

return self._izena


@nombre.setter

def izena (self, izena):

self._izena = izena


class Aeroplanoa:

def __init__ (self, modeloa, pilotua, kopilotua):

self._modelo = modelo

self._pilotua = pilotua

self._kopilotua = kopilotua


@property

def modeloa (self):

return self._modelo


@modelo.setter
def eredua (self, modeloa):

self._modelo = modelo


def volar (self):

return f "Hegan {self._modelo} {self._piloto.nombre} {self._copiloto.nom



pilotu1 = Pilotua ("Han Solo")

pilotu2 = pilotua ("Murdock")

avioneta = Aireplanoa ("AirBluff 727", pilotu1, pilotu2)


print (avioneta.volar ())
```

Emaitza:

```
Hegan AirBluff 727 Han Solo Murdockekin
```

Metodo estatikoak

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

```
alumno1 = 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 eskola bat egin dezakegu, lehenengo letra larriarekin eta gainerako letra xeheekin:

```
class Formatua:

@staticmethod

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


print (Formato.formato ("gUmBaLl"))
```

Emaitza:

```
Gumball
```

5.5 ariketa

Idatzi programa bat, `Numeroa ’izeneko klase bat definituko duena, eta funtzio estatiko bat, “aleatorio (max)” izenekoa. Funtzio honek zenbaki bat itzuli behar du 0 eta max tarteen barruan.

```
import random


class Zenbakia:

@staticmethod

def aleatorioa (max):

return random.randint (0, max)


for i in range (5):

print (Zenbakia.aleatorioa (10))
```

Emaitza:

```
4

3

0

9

1
```

DENA

Zergatik egin klase bat horrelako funtzio batekin eta ez zuzenean funtzio batekin? Ikasgela 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.

Proposatutako ariketak

5.0 ariketa Idatzi `Instrumento 'izeneko klase bat definitzen duen programa bat. Eraikitzaileak parametro hauek izan behar ditu, hurrenez hurren:`izena 'eta `mota'. Gainera,`ukitu 'izeneko funtzio bat gehitu behar duzu, mezu bat eta beste dei bat aterako dituena, atributuen informazioa duen testu bat itzultzeko.

Sortu gelako instantzia bat eta deitu bere funtzioetara.

```
class Instrumentua:

def __init__ (self, izena, mota):

self._izena = izena

self._tipo = mota


def ukitu (self):

print ("Ukitzen", self._izena)


def info (self):

return f "{self._nombre} {self._tipo}"



miGitarra = Instrumentua ("Gitarra", "soka")

miGuitarra.jo ()

print (miGuitarra.info ())
```

Emaitza:

```
Gitarra jotzen

Hari-gitarra
```

5.1 ariketa Idatz ezazu programa bat, `NombreFormateado" izeneko klase bat definituko duena, eta, bertan, etxegile bat agertuko da, etaizena 'eta `abizena' izeneko funtzio bat ere izango du. Sortu zure ustez egokiak diren eginkizun osagarriak.

```
class IzenaFormateatua:

def __init__ (self, izena, abizena):

self._izena = izena

self._apellidos = abizena


def formateatu (self):

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


def _zuzendu (self, katea):

return self._primero (cadena) + self._resto (cadena)


def _lehenengoa (self, katea):

return cadena [0] .upper ()


def _hondarra (self, katea):

return cadena [1: len (katea)] .lower ()



formateatzailea = Formateatua ("JUAN", "PÉREZ")

print (formateatzailea.formatear ())
```

Emaitza:

```
Juan Pérez
```

5.2 ariketa

Idatz ezazu programa bat `Sumador, 'izeneko klase bat definitzeko, eta bi zenbakirekin idatzi. 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 "batuketa" funtzioa izango dute.

```
class Batugailua:
def __init__ (self, valor1, valor2):
self.valor1 = valor1
self.valor2 = valor2

@property
def valor1 (self):
return self._valor1

@valor1.setter
def valor1 (self, valor1):
if balora1 > 0:
self._valor1 = valor1
LKse:
self._valor1 = 0

@property
def valor2 (self):
return self._valor2

@valor2.setter
def valor2 (self, valor2):
if balora2 > 0:
self._valor2 = valor2
LKse:
self._valor2 = 0

def batuketa (self):
return self._valor1 + self._valor2



sumador = Batugaia (28, 14)
print (batuketa)

sumador.valor1 = 600
sumador.valor2 = 66
print (batuketa)
```

Emaitza:

```
42

666
```

5.3 ariketa

Sortu programa bat `Monedaa 'izeneko klase batekin. Klaseak eraikitzaile huts bat izan behar du, eta funtzio bakar bat,` tirarin 'izenekoa. Horren emaitza "aurpegiaren" edo "gurutzearen" artean ausaz aukeratutako* string* bat izan behar da. Sortu ikasgelako instantzia bat probatzeko.

```
import random


def aleatorioa (max):

return random.randint (0, max)


class Moneta:

def tira (self):

aldeak = ["aurpegia", "gurutzea"]

kopurua = ausazkoa (1)


[zenbakia] aldeak itzultzen dituzte


moneta = Moneta ()


for i in range (10):

print (moneda.tirar ())
```

Emaitza:

```
aurpegia

aurpegia

gurutzea

gurutzea

gurutzea

...
```

5.4 ariketa

Sortu programa bat "N aurpegiko dado baten portaera simulatzeko" Datuak "izeneko klase batekin. Sortu ikasgelako instantzia bat probatzeko. - `def __init__ 'eraikitzailea (self, aldeak, admitiCero = False):` lapur' atributuarekin: aurpegi-kopurua gordetzen duen atributua eta `○ zeroOnartu = Faltsutzea: emandakoak 0 balioa itzul dezakeen esaten digun atributua. Lehenetsita, balio du:

- setter `def lados (self, lados) Parametrodun setter funtzioa. Atributua ezartzen du.

Zero (self, aldeak, onargarriaZero): parametrodun funtzioa, bi atributuak ezartzen ditu.

Funtzio horrek dadoaren jaurtiketa simulatzen du eta emaitza bat itzultzen du. Kontuan hartu behar duzu “Onartu zeroa” 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 aleatorioa (max):

return random.randint (0, max)


class Dado:

def __init__ (self, aldeak = 6, onartuCero = False):

self._aldeak = alboak

self._admiteCero = Zero onartzen du


@property

def lados (self):
return self._aldeak


@lados.setter

def aldeak (self, aldeak):

self._aldeak = alboak


@property

defCero onartzen du (self):

return self._admiteZero


@admiteCero.setter

Zero onartzen du (self, Zero onartzen du):

self._admiteCero = Zero onartzen du


def tira (self):

numero = aleatorioa (self._aldeak)


if not self._admiteZero:

kopurua = zenbakia + 1


return zenbakia
```

```
dado = Emana ()

for i in range (10):

print (dado.tirar ())
```

Emaitza:

```
4

3

2

4

3

...
```

5.5 ariketa

Sortu bi motatako programa bat:

1- `Jugadora 'klasea, izen, postu eta dortsal ezaugarriak dituena. Parametro horiek guztiak dituen eraikitzaile bat ere badu, eta ezaugarri guztiak itzuliko dituen`informe ’izeneko funtzio bat. 2 - Mota `Ekipola, izena, fundazioa, aurrekontua eta jokalari motako instantziak gordetzeko zerrenda. Honako ezaugarri hauek dituen eraikitzaile bat izan behar du: “izena”, “fundazioa”, “aurrekontua”, “get/seta”, “informazioa” funtzioa eta beste bi funtzio:

- `def ficha (self, jugador)», zerrendara jokalariak gehitzeko.
- `def demostratu ’Jokalariak (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 dizkiozu.

```
class Jokalaria:

def __init__ (self, izena, posizioa, dortsala):

self._izena = izena

self._posicion = posizioa

self._dortsal = dortsala


def informe (self):

return f "{self._nombre} {self._posicion} {self._dorsal}"



class Taldea:

def __init__ (self, izena, fundazioa, aurrekontua):

self._izena = izena

self._fundacion = fundacion

self._aurrekontua = aurrekontua
self._jugadores = []


def fichaJokalaria (self, jokalaria):

self._jugadores.append (jokalaria)


def mostrarJokalariak (self):

for jokalari in self._jugadores:

print (jokalari.txostena ())



jokalaria = jokalaria ("Maradona", "aurrelaria", 10)

jokalaria ("Beckenbauer", "Defentsa", 4)


print (jokalaria1.txostena ())


equipo = Taldea ("New Team", 1983, 39944.45)

taldea. FitxatuJokalaria (1. jokalaria)

taldea. FitxatuJokalaria (jokalaria 2)


taldea.erakutsiJokalariak ()
```

Emaitza:

`Maradona Aurrelaria 10`

`Beckenbauer Defentsa 4`

5.6 ariketa

Sortu zenbait klase barne hartzen dituen programa bat.

1 - Mota `Dispositiboa: "izena", "marka" eta "balioa" atributuak ditu. Eraikitzaile bat atributuak, set eta get etatoString` 'funtzio bat erabiltzen, atributuak erakutsiz. 2 - Mota `Movilere:` Dispositivoere 'motako azpiklasea da, eta "numeroa" atributua gehitu behar zaio. Sortu eraikitzailea eta `def toString (self) 'metodoa, superklasekoak aprobetxatuz. Gehitu "def llamar (self, numero)" funtzioa. Atera dezala pantailatik kate bat "numero deituz" esanez.

3 - Mota `Ordenadorea:` Dispositivoere 'azpiklasea da, "prozesadorea" atributua gehitu behar zaio. Sortu eraikitzailea eta "def toString (self)" funtzioa superklasekoak aprobetxatuz

Gainera, sakelakoa, ordenagailua eta laginak sortzeko behar den kodea gehitzen du.

```
class Gailua:

def __init__ (self, izena, prezioa):

self._izena = izena

self._prezioa = prezioa


def get_izena ():

return self._izena


def set_nombre (izena):

self._izena = izena


def get_prezioa ():

return self._prezioa


def set_prezioa (prezioa):

self._prezioa = prezioa


def toString (self):

return f "{self._nombre} {self._precio}";



class Movil (Gailua):

def __init__ (self, izena, prezioa, zenbakia):

super () .__init__ (izena, prezioa)

self._numero = kopurua


@property
```

```
@property
def zenbakia (self):
return self._zenbakia

@numero.setter
def zenbakia (self, zenbakia):
self._numero = kopurua

def toString (self):
return f "{super () .toString ()} {self._numero}"

def deitu (zenbakia):
print ("Deituz", zenbakia)



class Ordenagailua (Gailua):
def __init__ (self, izena, prezioa, prozesadorea):
super () .__init__ (izena, prezioa)
self._prozesadorea = prozesadorea

@property
def prozesadorea (self):
return self._prozesadorea

@procesador.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:", ordenadorea. 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. Bazkaria mota askotakoa izango da. Hauek dira egin beharreko klaseak:

1 - Mota `Komidaa: "izena" eta "pisua" atributuak ditu. Eraikitzaile bat atributuak, set eta get etatoString` 'funtzio bat erabiltzen, atributuak erakutsiz.

2 - Mota `Futua:`Komidaa 'motako azpiklasea da, eta `bitamina' atributua gehitu behar zaio. Sortu eraikitzailea etatoString` 'funtzioa superklasekoak aprobetxatuz.

3 - Mota `Karamela:`Komidaa 'azpiklasea da, eta `kaloria' atributua gehitu behar zaio. Sortu eraikitzailea eta funciñontoStringere` 'delakoa, superklasekoak aprobetxatuz. 4 - Mota `Kestak,`elikadura 'izeneko atributua du, eta janari-motako elementu-arrai bat da (`FutuaCarameloa`). Eraikitzailean hasten da. Hiru funtzio ditu:

- `def meterOtordua (self, jana)
- `def pesoTotal (self) "-k saskiko janariaren guztizko pisua itzultzen du.
- `def toString (self)» saskiko janari guztia erakusteko.

Gainera, erantsi behar den kodea instantzia hauek sortzeko: `Fruta,`Carameloa, eta erantsi `Cestaa.

```
class Janaria:

def __init__ (self, izena, pisua):

self._izena = izena

self._pisua = pisua


@property

def izena (self):

return self._izena


@nombre.setter

def izena (self, izena):

self._izena = izena


@property

def pisua (self):

return self._pisua


@peso.setter

def pisua (self, pisua):

self._pisua = pisua
```

```
def toString (self):

return f "{self._nombre} {self._peso}"



class Fruta (Janaria):
def __init__ (self, izena, pisua, bitamina):

super () .__init__ (izena, pisua)

self._vitamina = bitamina


@property

def bitamina (self):

return self._bitamina


@vitamina.setter

def bitamina (self, bitamina):

self._vitamina = bitamina


def toString (self):

return f "{super () .toString ()} {self._vitamina}"



class Gozokia (Bazkaria):

def __init__ (self, izena, pisua, kaloriak):

super () .__init__ (izena, pisua)

self._calorias = calorias


@property

def calorias (self):

return self._kaloriak


@calorias.setter

def calorias (self, alorias):

self._calorias = calorias


def toString (self):

return f "{super () .toString ()} {self._calorias}"
```

```
class Zesta:

def __init__ (self):

self._alimentos = []


def meterJanaria (self, janaria):

self._alimentos.append (janaria)


def pisuaGuztira (self):

guztira = 0

for elikagai in self._alimentos:
guztira + = elikagaia.pisua


erreturn osoa


def toString (self):

info = ''

for elikagai in self._alimentos:

info = info + alimento.toString () + "\n"


return info



chicle = Gozokia ("Cheiw", 0.2, 100)

gominola = Gozokia ("Marrubia", 0.3, 210)

pera = Fruta ("Udarea", 0.1, "B")

sagarra = fruta ("Sagarra", 0.15, "A")


cesta = Saskia ()

cesta.meterJanaria (txiklea)

cesta.meterJanaria (gominola)

cesta.meterJanaria (udarea)

cesta.meterJanaria (sagarra)


print ("Saski-edukia:", saskia.toString ())

print ("Pisua guztira:", zesta.pisua guztira ())
```

Emaitza:

#Salbuespenak

Baliteke dagoeneko programazioan bikaina izatea. Baina, hala ere, zure programek huts egiten jarrai dezakete, gauza batzuk zure programaren kontroletik kanpo daudelako eta zure programak funtzionatzeari uzten diolako.

Adibidez, zure programak erabiltzaileak zenbaki bat idaztea espero badu, baina erabiltzaileak letrak idazten baditu edo 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 saio horietako bat gertatzen bada gure programak huts egin ez dezan eta besterik gabe amai dadin. Eta mekanismo hori salbuespenak dira.

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 "saltaexcepcion.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 hizkuntzaren egitura bat gehiago da, eta forma hau du:

```
try:

codigo_que_puede_huts egin

salbuespen:

codigo_que_se_ejecuta_si_hay_error
```

Ikus dezagun aurreko adibidea, salbuespen-blokearen barruan kode sentikorra babestuz:

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


try:

balioa = int (balioa)

karratua = balioa* balioa

print ("Karratua da:", karratua)

salbuespen:

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 espezifikoagoa erakusteko erabil daitezke.

#Fitxategiak maneiatzea

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.

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 ("texto.txt", "r")

edukia = fichero.read ()

print (edukia)

fichero.close ()
```

Kontuan hartzekoak:

- Fitxategia irakurtzeko, lehenik eta behin ireki egin behar da, honako honekin:
- Fitxategia irekitzean, haren izena adierazi behar da, eta, beste direktorio batean badago, harako bidea. Fitxatzeko karpetan egonez gero, hau izango litzateke bidea: `fichero/texto.txt`.
- “r” parametroak adierazten du fitxategia irakurtzeko 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
```

#Lerroz lerro irakurtzen?

Aurreko adibidean, bat-batean irakurri dugu fitxategiaren eduki osoa, testu-aldagai batean gordeta. Baina batzuetan, baliteke online fitxategia irakurri nahi izatea. Horretarako, honako funtzio hau erabili behar dugu:

```
fitxategia = open ("texto.txt", "r")

lineak = fichero.readline ()

for linea in lineak:

print (linea)


fichero.close ()
```

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. Hizkuntzarenak bezalako zerrendak irudikatzeko aukera ere badu.

Hurrengo edukia JSON formatuan dago. Hainbat objektu dituen zerrenda da.

Erreparatzen badiozu, 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 inporta daitekeela, betiere zuzena bada, noski.

Eduki hori irakurri eta datu horiek hiztegi zerrenda bihurtzeko, `json, 'liburu-denda 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


fitxategia = open ("texto.json", "r")

edukia = json.load (fitxategia)


for edukirik gabeko pertsonaia:

print (pertsonaia ["izena"])


fichero.close ()
```

Pantaila batetik, hau ikusiko dugu:

```
Ada

Neko

Bug
```

Fitxategien eskritura

Fitxategiak idazteko, prozesua antzekoa da, baina bi gauza egin behar ditugu:

- Fitxategia idazkera moduan irekitzea.
- Erabili `writee ’funtzioa edukia idazteko.

Kode honekin, testu-lerro pare bat idatziko ditugu fitxategian:

```
fitxategia = open ("texto.txt", "w")
fichero.write ("Lerro bat\n idazten dut")

fichero.write ("Beste lerro bat idazten dut\n")

fichero.close ()
```

KONTUZ! Horrela idazten badugu, fitxategiaren edukia txikituko dugu.

Fitxategiaren edukia:

```
Lerro bat idazten dut

Beste lerro bat idazten dut
```

Edukia gehituz idatzi nahi badugu, fitxategia “a” moduan ireki behar dugu:

```
fitxategia = open ("texto.txt", "a")

fichero.write ("Gehitu lerro bat\n")

fichero.write ("Gehitu beste lerro bat\n")

fichero.close ()
```

Orain, fitxategiaren edukia hau izango litzateke:

```
Lerro bat idazten dut

Beste lerro bat idazten dut

Gehitu lerro bat

Gehitu beste lerro bat
```

Fitxategi batean idatzi json

JSON formatuko fitxategi baten kasuan, kezkatzekoa da idazteko unean gure datuak testu bihurtzea.

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 `writee 'bat egingo dugu`,json.dumps, `erabiliz edukia testu bihurtzeko:

```
import json


fitxategia = open ("texto.json", "r")

edukia = json.load (fitxategia)

fichero.close ()


pertsonaia = {"id": 666, "izena": "Gumball"}

edukia.append (pertsonaia)


fitxategia = open ("texto.json", "w")

fichero.write (json.dumps (edukia))

fichero.close ()
```

#Liburu-dendak

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.

Kodea fitxategietan eta karpetetan antolatu ondoren, beste fitxategi batzuetan berrerabil ditzakegu. Ikus dezagun adibide sinple bat.

Funtzio hau definituko dugu `mates.pymes ’izeneko fitxategi batean:

```
Batuketa (a, b):

return a + b


def restar (a, b):

return a - b


def handitu (a):

return a - 1
```

Orain, fitxategi hori beste programa batean sar dezakegu, `axola dio 'aginduaren bidez. Direktorio berean badaude, besterik gabe egin daiteke:

```
import mates


balioespena 1 = 5

balioespena 2 = 10


emaitza = mates.batu (1. balorazioa, 2. balorazioa)

print (emaitza) # 15
```

6.0 ariketa

Aurreko ekitaldi batean, pasahitzen sorgailu bat egitea proposatu zen. Erabili kode bera, baina sinetsi fitxategi baten barruan. Sortu beste fitxategi bat kode hori inportatzeko eta erabiltzeko.

Generar.ete fitxategia:

```
import random


def aleatorioa (max):

return random.randint (0, max - 1)


def crearPassword (luzera):

karaktereak = ["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):

caracter = karaktereak [ausazkoa (len (karaktereak))]

password = password + caracter


return password
```

Eta 1 ETE fitxategian erabiltzen dugu:

```
inport sortu


password = generar.crearPassword (8)


print (pasahitza)
```

Emaitza:

```
g3ep-ahx
```

Klaseekin gauza bera egin daiteke. Demagun `LectorPantallaa 'izeneko klasea dugulalector_pantalla.py`izeneko fitxategi batean. Kontsolatik datuak irakurtzeko aukera ematen digun klase bat da:

```
class IrakurgailuaPantaila:

def leerOsoa (self, mensaje = "Sartu zenbaki bat:"):

numero = input (mezua)

return int (zenbakia)


def leerTestua (self, mensaje = "Sartu testua:")

testua = input (mezua)

return testua
```

Orain, klase hori beste fitxategi batean berrerabil dezakegu, gure fitxategiarekin batera.

```
inport lector_pantaila

import mates


irakurlea_pantaila. Pantaila irakurgailua ()

valor1 = irakurlea.osoa ()


print (mates.handitu (baloratu 1))
```

Pantailan horrelako zerbait ikus daiteke:

```
Sartu zenbaki bat: 6

7
```

6.1 ariketa

Definitu `Menue izeneko klase bat, honako funtzio hauek dituena:

1. `def init__ (self): aukeren zerrenda bat (* strings*) jasotzen du parametro gisa.
2. “def mostrar (self)”: aurretik zenbaki bat duen aukerak erakusten ditu, a print deituz.
3. `def selecciono (self, numero) ItzuliTruee`, baldin eta aukeratutako zenbakia menuan badago; bestela, itzuli `False«.

Gero klase hori 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._opciones)):

print (f "{i+1} {self._opciones [i]}")


def hautatu (self, zenbakia):

return numero > 0 and numero < = len (self._opciones)
```

`2 ETEa fitxategia:

```
import menu

miMenu = menu. Menua (["Erakutsi", "Ezabatu", "Irten"])


miMenu.mostrar ()

if miMenu.hautatu (1):

print ("Oraingo 1. aukera")

LKse:

print ("1. aukera ez dago")
```

Emaitza:

```
1 Erakutsi

2 Ezabatu

3 Irten

Oraingo 1. aukera
```

Proposatutako ariketak

6.0 ariketa

Sortu `Ficheroere ’izeneko klase bat, honako funtzio hauekin:

1. `def **init** (self, fitxategia): ireki beharreko fitxategia jasotzen du parametro gisa.
2. `def leer (self) “: fitxategiaren edukia jasotzen duen kate bat itzultzen du.
3. “def escribir (self, contenido)”: fitxategian parametro gisa pasatzen zaion edukia idazten du.

Gero, ikasgela erabili behar duzu beste fitxategi batean inportatuz.

Fitxategia:

```
class Fitxategia:

def __init__ (self, izenaFitxategia):

self._nombreFichero = nombreFichero


def leer (self):

fitxategia = open (self._nombreFichero, "r")

datuak = fichero.read ()

fichero.close ()


datuak itzultzea


def idatzi (self, edukia):

fitxategia = open (self._nombreFichero, "w+")

fichero.write (edukia)
fichero.close ()
```

Fitxategia: `6.0.ete:

```
from datetime import date


fitxategiaren inport


miFichero = fichero. Fitxategia ("6.0.txt")


print ("Aurreko edukia:", miFichero.leer ()


miFichero.escribir ("Edukia aldatuta!!!" + str (date.today ()))

print ("Edukia:", miFichero.leer ())
```

Testu-fitxategia:

```
Horixe da gaur egungo edukia.
```

Emaitza:

```
Aurreko edukia: Eduki aldatua!!! 2020-08-18

Edukia: Edukia aldatuta!!! 2020-08-23
```

6.1 ariketa

Sor ezazu “Zerrenda” izeneko klase bat, honako funtzio hauekin:

1. `def **init** (self, fitxategia): 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, «id» formatua duena: 1, «izena»: «Juan»}
2. `def (self, izena) dago: itzuli egin behar duzuTrue«/ `Faldea, parametro

gisa pasatzen den izena zerrendan badago.

3. `def aMinuskulak (self): zerrendako izen guztiak letra xeheetara pasatu behar dira.
4. “def posicion (self, nombre)”: izen hori dagoen posizioa itzuli behar duzu.

Gero, ikasgela erabili behar duzu beste fitxategi batean inportatuz.

Zerrenda fitxategia:

```
import json


class Zerrenda:

def __init__ (self, izenaFitxategia):

edukia = open (izenaFitxategia, "r")

self._datos = json.load (edukia)

edukina.close ()


def existitzen da (self, izena):

for dato in self._datos:

if datua ["izena"] = = izena:

return True


return False


def aMinuskulak (self):

self._datos = list (map (lambda dato: {"id": datua ["id"], "izena": datu


def posicion (self, izena):

i = 0

for dato in self._datos:

if datua ["izena"] = = izena:

return i

i + = 1

return -1


def print (self):

for dato in self._datos:
print (datua)
```

`6.1 ETEa fitxategia:

```
zerrenda import
miLista = lista. Zerrenda ("6.1.json")

if miListado.badago ("eugene"):
print ("Badago!")

NireZerrenda.aMinuskulak ()
miListado.print ()

if miListado.badago ("eugene"):
print ("Badago!")
print (miListado.posicion ('eugene'))
```

`6.1.json«fitxategia:

``javascript ’

```
[
{
“id”: 3,
“Izena”: “Juan”
},
{
“id”: 5,
“Izena”: “Eugene”
},
{
“id”: 10,
“Izena”: “Paul”
}
]
```

Emaitza:

```
```console
{'id': 3, 'nombre': 'juan'}
{'id': 5, 'izena': 'eugene'}
{'id': 10, 'izena': 'paul'}
Badago!
1
```
```

6.2 ariketa

Sor ezazu `Tareas«izeneko klase bat, honako funtzio hauekin:

1- `def **init** (self) «Tareas.json» izeneko fitxategia ireki behar duzu, eta honako formatu hau izango duten hiztegiak zerrenda batean kargatu: {id: 1, ataza: «Ikasi zerbait»}. Zerrenda hori atributua izango da.

2- `def sortu (self, zeregina) ": objektu berri bat sortu eta zerrendan gordetzen du. 3-`def eliminatu (self, id): ID horrek duen zerrendako ataza bat ezabatzen du.

4- `def guardar (self) “: gorde zerrenda fitxategi honetan.json.

5- `def mostrar (self) “: string batean itzultzen ditu ataza guztiak.

Gero, ikasgela erabili behar duzu beste fitxategi batean inportatuz.

Fitxategia: `tareas.py«»

```
import json


class Zereginak:
def __init__ (self):
fitxategia = open ("tareas.json", "r")
self._tareas = json.load (fitxategia)
fichero.close ()


def sortu (self, id, zeregina):
berria = {"id": id, "zeregina": zeregina};
self._tareas.append (berria)


def gorde (self):
fitxategia = open ("tareas.json", "w")
fichero.write (json.dumps (self._atazak))
fichero.close ()


def ezabatu (self, id):
self._tareas = list (filter (lambda dato: dato ["id"]! = id, self._tarea


def erakutsi (self):
emaitza = ''
for dato in self._atazak:
emaitza + = json.dumps (datua) + "\n"


erreturn emaitza
```

Fitxategia: `tareas.json«:

``javascript ’

[

“Go ikastea”, “id”: 3},

{“Zeregina”: “Rust ikertzea”, “id”: 5},

“Zeregina”: “Lo gehiago egin”, “Id”: 10}

] ```

Fitxategia:

```
Atazen inport

misTareas = atazak. Zereginak ()


print (nireTareak.erakutsi (), "\n---")


miTareak.sortu (2, "Jan")

print (nireTareak.erakutsi (), "\n---")


misTareas.eliminatu (2)

print (nireTareak.erakutsi (), "\n---")


misTareas.crear (66, "Irakurri")

print (nireTareak.erakutsi (), "\n---")

misTareas.gorde ()
```

Emaitza:

```
{"Zeregina": "Go ikastea", "id": 3}

{"ataza": "Rust ikertu", "id": 5}

"Ataza": "Lo m\u00e1s", "Id": 10}


- --

{"Zeregina": "Go ikastea", "id": 3}

{"ataza": "Rust ikertu", "id": 5}

"Ataza": "Lo m\u00e1s", "Id": 10}

{"ataza": "Jan", "id": 2}


- --

{"Zeregina": "Go ikastea", "id": 3}

{"ataza": "Rust ikertu", "id": 5}

"Ataza": "Lo m\u00e1s", "Id": 10}


- --

{"Zeregina": "Go ikastea", "id": 3}

{"ataza": "Rust ikertu", "id": 5}

"Ataza": "Lo m\u00e1s", "Id": 10}
"Ataza": "Irakurri", "Id": 66}


- --
```

6.3 ariketa

Sor ezazu `Jugadora ’izeneko klase bat, eta eduki hau izango du:

1. `def __init__ (self, izena, dortsala): parametroak esleitzen dizkie_izena_eta `_dorsal, atributuei.
2. Get/set metodoak `izenerako etadortserako ’.
3. “def info (self)”:* string* bat itzultzen du jokalariaren informazioarekin.

Sor ezazu `Ekipola ’izeneko klase bat, eta eduki hau izango du:

1. `def cargar (self) «Jokalari» izeneko fitxategi bat ireki behar duzu. Fitxategi horrek jokalari-hiztegien zerrenda bat izango du [{izena: “Pele”, dortsala: 10}, {).

Eta fitxategiaren objektu bakoitzeko, “JUgadora” motako instantzia bat sortu behar duzu, eta `this._jugadoresizeneko zerrenda batean sartu.

2. `def erakutsi (self) “: jokalarien zerrenda osoa erakutsi behar duzu.
3. `def fitxaketa (self, izena, dortsala) “: jokalari berri bat sartu behar duzu zerrendan,” JUgadora “ren instantzia bat sortuz.

Ikasgelan, `Jugadora ’klasea inportatu beharko duzu, erabili ahal izateko.

`○ jokalari.ete ’fitxategia:

```
class Jokalaria:

def __init__ (self, izena, dortsala):

self._izena = izena

self._dortsal = dortsala


@property

def izena (self):

return self._izena


@nombre.setter

def izena (self, izena):

self._izena = izena


@property

def dortsala (self):

return self._dortsala


@dorsal.setter

def dortsala (self, dortsala):

self._dortsal = dortsala


def toString (self):

return f "{self._nombre} {self._dorsal}"
```

· ETEen taldea fitxategia:

```
import json

Jokalariaren prezioa


class Taldea:

def kargatu (self):

edukia = open ("./jugadoreak.json")

jokalariak = json.load (edukia)

print ("Loaded:", jokalariak)

self._jugadores = []

for j in jugadores:

self._jugadores.append (jokalaria. Jokalaria (j ["izena"], j [dortsala "


def fitxaketa (self, izena, dortsala):

berriaFitxaketa = jokalaria. Jokalaria (izena, dortsala)

self._jugadores.append (Fitxaketa berria)

def erakutsi (self):

for jokalari in self._jugadores:

print (jokalaria.toString ())
```

Jokalarien fitxategia:

```
[

{

"izena": "Maradona",

"dortsala": 10

},

{

"Izena": "Pele",

"dortsala": 8

}

]
```

Fitxategia: `6.3 ETE:

```
taldeko import

miEquipo = taldea. Taldea ()


nire taldea.kargatu ()

NireTaldea.erakutsi ()

Nire taldea.fitxaketa ("Gento", 11)

NireTaldea.erakutsi ()
```

Emaitza:

```
Loaded: [{'dortsal': 10, 'nombre': 'Maradona'}, {'dorsal': 8, 'nombre':
'Pele'}]

Maradona 10

Pele 8

Maradona 10

Pele 8

Gento 11
```

#Gehigarriak

Python-i buruz

Zergatik aukeratu dugu Python? Bere bertute ugariengatik. Hizkuntza interpretatua da, sintaxi oso erraza duena, eta horrek ikasteko oso erraza egiten du. Ez da kezkatu behar (asko) hizkuntzaren mota eta zurruntasunez. Kontuan hartu behar dugun bakarra bloke bakoitzeko tabulazioak errespetatzea da.

Helburua ez da hizkuntza bere horretan ikastea, funtsezkoa programatzen ikastea da, eta Pythonek zeregin hori errazten du.

Gainera, oso hizkuntza erabilgarria da, oso hedatua eta profesionalki erabilia. Hori gutxi balitz, garatzaileek asko estimatzen dute, eta horrek kodea, liburu-dendak eta laguntza ematen duten pertsonen komunitate izugarria dakar.

Pythonek bi bertsio berezi ditu, 2 eta 3. Liburu honetan 3. sintaxia eta estiloa erabiltzen saiatu gara, egunean gehien dagoena erabiltzeagatik.

Python tokian-tokian instalatzen

Nahikoa da [python site] ra joatea (https://www.python.org) 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 seriean instalatuta izango duzu, edo, beharbada, ez duzu zertan esan nola instalatu;)

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.
- [atom] (https://atom.io /)

- [code] (https://vscode.io)
- [pydev] (http://pydev.org)
- [sublime] (http://www.sublimetext.com)

Test unitarioak

Unitate-testak funtzioak ondo eginda daudela egiaztatzen duten programak dira. Funtsean, funtzioak betetzen dituzten eta emaitza zuzena dela egiaztatzen duten programak dira. Testak egiteko hainbat liburu-denda daude, nahiz eta Python-en `unittest ’lehenetsia dagoen, eta, beraz, ez dago ezer instalatu beharrik.

Demagun kalkulagailu bat irudikatzen duen klase hau dugula:

```
class Kalkulagailua:

def batuketa (self, a, b):

return a + b


def restar (self, a, b):

return a – b


def multiplicar (self, a, b):

return a* b


def zatitu (self, a, b):

return a/b
```

Mota horretako test unitarioak egiteko, nahikoa litzateke beste mota hau sortzea, eta hori `unittestere ’liburu-dendako testa-mota baten oinordekoa izango litzateke. Kalkulagailua bera ere inportatu behar dugu, proban jarri behar baitugu.

Gelako 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_suma (self):

c = kalkulagailua. Kalkulagailua ()

self.assertEqual (c.batu (40, 2), 42)
```

Fitxategi `kalkulatzailea.test.ete:

```
import unittest

kalkulagailuaren inport


class TestStringMethods (unittest. TestCase):

def test_suma (self):

c = kalkulagailua. Kalkulagailua ()

self.assertEqual (c.batu (40, 2), 42)


def test_restar (self):

c = kalkulagailua. Kalkulagailua ()

self.assertEqual (c.restar (40, 2), 38)


def test_biderkatu (self):

c = kalkulagailua. Kalkulagailua ()

self.assertEqual (c.multiplicar (40, 2), 80)


def test_dividir (self):

c = kalkulagailua. Kalkulagailua ()
self.assertEqual (c.dividir (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
```

Python proiektu bati hasiera ematen tokian-tokian

Python proiektu bat hasteko modu gomendagarri bat izango litzateke `virtualenv`,pakete kudeatzailearekin instalatuko dugun pakete `birtuala 'erabiltzea, 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 instalatu birtualenv

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

...
```

Birtualenvekin proiektu berri bat sor dezakegu:

```
birtualenv proiektua
```

Eta horrek “proiektatua” izeneko karpeta sortuko du.

Jarraian, proiektuaren ingurune birtuala aktibatu behar dugu, honako hauek gauzatuz:

```
source proiektua/bin/activate
```

Orain, iturri-kodea gehitu dezakegu edo dependentziak instalatu.

Horretarako, `requirements.txtere ’izeneko testu-fitxategi bat sortzea komeni da. Fitxategi horrek formatu hau izan behar du:

```
#Bertsio zehatz bat instalatzeko

#nombre_pakete = = bertsioa


#Bertsio berdina edo handiagoa instalatzeko

#nombre_paquete> = bertsioa


#Bertsio berriena instalatzeko

#izena_paketea
```

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, komandoarekin adierazten ditugunak:

```
source bin/activate
pip3 instalatu -r requirements.txt
```

Beste aukera bat da pip3 duten beharrezko paketeak instalatzea:

```
pip3 instalatu pygame
```

Instalatuta dagoela egiaztatuko dugu:

```
pip3 list

Package Version

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

pip 20.2.2

ETE 1.9.6

setuptools 49.6.0

wheel 0.35.1
```

Eta `freeze`, 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, "w"
screen = pygame.display.set_mode ([500, 500])


#Run until the user asks to quit
running = True
while running:


Nola egin dezakezu klik?
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
```