

PYTHON programazio-lengoaia: oinarriak eta aplikazioak

Iñaki Alegria Loinaz
Olatz Perez de Viñaspre Garralda
Kepa Sarasola Gabiola

PYTHON PROGRAMAZIO- LENGOAIA: OINARRIAK ETA APLIKAZIOAK

Iñaki Alegria Loinaz
Olatz Perez de Viñaspre Garralda
Kepa Sarasola Gabiola

Udako Euskal Unibertsitatea eta Euskal Herriko Unibertsitatea
Bilbo, 2016

© Udako Euskal Unibertsitatea

© Euskal Herriko Unibertsitatea

© Iñaki Alegria Loinaz, Olatz Perez de Viñaspre Garralda, Kepa Sarasola Gabiola

UEUren ISBNa: 978-84-8438-600-1

Lege-gordailua: BI-1001-2016

Inprimategia: PRINTHAUS S.L., Bilbo

Azalaren diseinua: Igor Markaida Uriagereka

Hizkuntza-zuzenketen arduraduna: Ander Altuna Gabiola

Banatzailerak: UEU. Erribera 14, 1, D BILBO. Telf. 946790546

Helbide elektronikoa: argitalpenak@ueu.eus

www.ueu.eus

Euskal Herriko Unibertsitatea

Helbide elektronikoa: argitalpenak@ehu.eus

www.ehu.eus/argitalpenak

Elkar Banaketa: Igerabide, 88 DONOSTIA

Galarazita dago liburu honen kopia egitea, osoa nahiz zatikakoa, edozein modutara delarik ere, edizio honen Copyright-jabeen baimenik gabe.

Liburu honek UEUren argitalpengintzako ebaluazio-prozesua gainditu du; liburuaren jakintza-alorreko hiru adituk ebaluatu dute jatorrizkoa, peer review erako ebaluazioan, horietako bik egilearen daturik ezagutu gabe (double-blind).

Informatika eta euskara lotzen aitzindaria izan zen Klaudio Harluxet gogoan

Joserra Etxebarria maisu handiari

Aurkibidea

1. SARRERA	11
1.1. Helburua	11
1.2. Historia	12
1.3. Zergatik Python?.....	13
1.4. Ezaugarri orokorrak	14
1.5. Python2 vs Python3	15
1.6. Lan-ingurunea.....	16
1.7. Kodea.....	19
1.8. Bibliografia	19
 2. LEHEN PROGRAMAK ETA OINARRIZKO DATU-MOTAK	 21
2.1. Kaixo guztioi!	21
2.2. Aldagaiak, konstanteak eta datu-motak	22
2.3. Programa sinpleak. Agindu-segidak	25
 3. ERAGILEAK ETA ESPRESIOAK.....	 27
3.1. Espresioak.....	27
3.2. Eragileen arteko lehentasuna	27
3.3. Eragileen arteko elkargarritasuna	28
3.4. Esleipena.....	29
3.5. Karaktere-kateen gaineko eragileak	29
3.6. Eragile aritmetikoak	29
3.7. Eragile logikoak eta erlazioak	31
3.8. Bit-maneiua	32
3.9. Oinarrizko funtzio batzuk.....	33
3.10. Programen testuaren formatua: tabulazioak eta zuriuneak.....	34
 4. KONTROL-EGITURAK.....	 35
4.1. <i>if</i> kontrol-egitura	36
4.2. <i>while</i> kontrol-egitura.....	38
4.3. <i>for</i> .. <i>in</i> kontrol-egitura	39
4.4. <i>break</i> eta <i>continue</i> aginduak	41

4.5. Adibideak.....	42
4.5.1. Enuntziatua: Hirutan handiena	42
4.5.2. Enuntziatua: Zerrendakoen batura.....	42
4.5.3. Enuntziatua: Atzekoz aurrera	43
4.5.4. Enuntziatua: Zenbat bateko zenbaki bitarrean	43
4.6. Proposatutako ariketak	44
5. FUNTZIOAK, KLASEAK ETA METODOAK	45
5.1. Sarrera.....	45
5.2. Funtzioak	46
5.2.1. Definizioa, deia eta emaitzak.....	46
5.2.2. Argumentuak.....	47
5.2.3. Kanpoko funtzio bat erabiltzea	48
5.2.4. Funtzioak: kontzeptu aurreratuak	49
5.3. Adibideak funtzioekin.....	50
5.3.1. Enuntziatua: Zenbaki handiena.....	50
5.3.2. Enuntziatua: Mezu laburrena	50
5.3.3. Enuntziatua: Faktoriala funtzio errekursibo gisa.....	51
5.3.4. Enuntziatua: Funtzioen proba	51
5.4. Objektuak, klaseak eta metodoak	52
5.5. Klase berriak definitzen	53
5.6. Adibideak klaseekin.....	55
5.6.1. Enuntziatua: Denbora ordu eta minututan.....	55
5.7. Programa nagusiari parametroak pasatzea	55
5.8. Proposatutako ariketak	56
6. OHIKO DATU-EGITURAK ETA DAGOZKIEN METODOAK	59
6.1. Ohiko datu-moten gaineko metodoak.....	59
6.1.1. Karaktere-kateen gaineko oinarrizko metodoak	60
6.1.2. Zerrenden gaineko metodoak	61
6.1.3. Zerrenden eta karaktere-kateen arteko bihurketa	63
6.1.4. <i>List comprehension</i>	64
6.1.5. Espresio erregularrak.....	64
6.1.6. Bektoreak eta matrizeak	65
6.1.7. Tuplak eta heina (<i>range</i>).....	66
6.1.8. Bestelakoak	66
6.2. Hiztegiak	67
6.2.1. Hiztegien gaineko metodoak	68
6.3. Erroreak eta salbuespenen kudeaketa	69

6.4. Adibideak	71
6.4.1. Enuntziatua: Palindromoak detektatu.....	71
6.4.2. Enuntziatua: Bokal gehieneko hitzak	72
6.4.3. Enuntziatua: Matrise-biderketa	73
6.4.4. Enuntziatua: Hitza atzekoz aurrera jarrita ere badago listan.....	73
6.5. Proposatutako ariketak.....	74
7. SARRERA/IRTEERA. INTERNET	75
7.1. Mezuen formatua.....	75
7.1.1. Modu zaharra	75
7.1.2. format metodoa	76
7.2. Fitxategiak	78
7.2.1. Ohiko metodoak	81
7.2.2. Erroreen maneiua	82
7.2.3. Zuzeneko atzipena.....	83
7.2.4. Katalogoen maneiua	84
7.2.5. Bestelakoak	85
7.3. Internet.....	85
7.4. Adibideak.....	86
7.4.1. Enuntziatua: Hitzen maiztasuna	86
7.4.2. Enuntziatua: Zuhaitz bitarra idatzi	87
7.5. Proposatutako ariketak	88
8. KLASA BERRIAK ETA OHIKOENAK.....	89
8.1. Klaseetan sakontzen	89
8.1.1. Aldagaien eta funtzioen esparrua	89
8.1.2. Herentzia eta polimorfismoa	91
8.2. Moduluak instalatzen.....	93
8.3. Kalkulurako moduluak	93
8.3.1. Funtzio matematikoak.....	94
8.3.2. Zorizko zenbakiak eta estatistikak	95
8.4. Datuen formatuak	96
8.4.1 CSV	96
8.4.2 JSON	99
8.4.3 XML gaineko metodoak.....	100
8.5. Konexioa datu-baseekin	102
8.6. Grafikoak eta irudiak	103
8.6.1. Irudiak	103
8.6.2. Grafikoak.....	105

8.7. Bestelakoak.....	107
8.7.1. Internazionalizazioa (i18n)	107
8.8. Proposatutako ariketak	109
9. SISTEMAREN GAINEKO AUKERAK.....	111
9.1. Sistemaren komandoak.....	111
9.2. Administrazioa.....	112
9.3. Sistema-deiak.....	116
9.4. <i>Multithreading</i>	117
9.5. Adibideak.....	120
9.5.1. Enuntziatua: <i>grep</i> paraleloa	120
9.5.2. Enuntziatua: FIFOak harien bidez.....	121
9.6. Proposatutako ariketak	122
10. APLIKAZIOAK I: INTERNET	123
10.1. Ariketa: <i>Wikinews</i> -etik berrien izenburuak erauzten.....	123
10.2. Ariketa: Euskal Herriko futbol-selekzioko jokalarien ibilbidea erauztea Wikipediatik.....	125
10.3. Ariketa: Jokalarien ibilbidearen web orria sortzea.....	130
10.4. Ariketa: Twitterreko traola baten jarraipena egitea.....	132
10.5. Proposatutako ariketak	135
11. APLIKAZIOAK II: ARGAZKIEN TRATAMENDUA ETA JOKOAK.....	137
11.1. Argazkien tratamendua	137
11.1.1. Instalazioa	137
11.1.2. Ariketa: Argazkiei logo bat itsasi nahian.....	138
11.1.3. Proposatutako hobekuntzak	139
11.2. Jokoak	140
11.2.1. <i>Hiru lerroka</i> edo artzain-jokoa (lehen hurbilpena).....	140
11.2.2. <i>Hiru lerroka</i> edo artzain-jokoa (bigarren hurbilpena)	144
11.2.3. Mahai-tenisa (ping-pong)	146
11.2.4. Proposatutako hobekuntzak.....	150
12. APLIKAZIOAK III: TESTUAK LANTZEN.....	153
12.1. Testuak internetetik eskuratzea	155
12.2. NLTK tresnak.....	158
12.2.1. Instalazioa eta baliabideen deskarga.....	159
12.2.2. Ariketa: Corpusak, zenbaketak eta stopword-ak	159
12.2.3. Ariketa: Lemak eta kategoriak.....	162

12.2.4. Ariketa: Antzeko hitzak lortzen	164
12.3. NLTK gure testuekin erabiltzen.....	165
12.3.1. Ariketa: Euskarazko n-gramen maiztasunak	165
12.3.2. Ariketa: Zuzentzaile ortografiko sinple bat	167
12.3.3. Ariketa: Zuzentzaile ortografiko oso bat integratzen	168
12.4. Proposatutako ariketak	170

1. Sarrera

Lehen kapituluaren sarrera baten ohiko osagaiak bildu ditugu: liburuaren helburua, programazio-lengoiaren historia, lengoaia hau aukeratzeko arrazoiak, lengoiaren ezaugarriak eta aukerak, eta bukatzeko programa sortzeko lan-ingurunea, programa-adibideen kodea aurkitzeko bidea eta erabilitako bibliografia.

1.1. HELBURUA

Liburu honen helburua ez da Python lengoiaren aukera guztiak zehaztasun osoz deskribatuko dituen eskuliburu bat egitea, ikasteko material motibatzaile eta gertukoa sortzea baizik. Hori dela-eta, adibideak eta ariketak izango dira liburuaren muina; adibideak aztertzea, probatzea eta programa berriak sortzea baita ikasteko metodorik onena. Beraz, nagusiki tutorial bat da, eta bigarren maila batean erreferentziazko liburu bat. Hala eta guztiz ere, adibide kopuru mugatu batekin lengoaia ahaltsu honek eskaintzen dituen aukera guztiak erabiltzea ezinezkoa denez, hainbat kasutan adibidean erabili ez diren beste aukera posible batzuk ere azaltzen dira gero deskribapen osagarri baten bidez, edo taula batean laburtuta.

Horrela kapituluak laburrak izango dira, adibideetatik abiatuta aukera ohikoenak azalduko dira hasierako ataletan, gero ariketa osatuagoak aurkeztu, eta bukaeran ariketa berriak proposatuko dira.

Liburuari etekina ateratzeko derrigorrezkoa da aldeztu aurretik algoritmikako oinarritzko kontzeptuak ezagutzea, liburua ez da egokia hutsetik abiatuta programatzen ikasteko. Horretarako, eta liburu honen osagarri gisa, UEUK argitaratutako *Oinarritzko programazioa: Ariketa bilduma* liburua erabil daiteke. Atzigarri dago helbide honetan: http://www.buruxkak.eus/liburuak_ikusi/1945/oinarritzko_programazioa__ariketa_bilduma.html.

Liburua bi partetan banatzen da:

- **Oinarriak**, batez ere adibide sinpleen bidez azaltzen direnak.

Lehen partearen helburuak bi dira: lengoiaren nondik norakoak ondo ulertzea eta programa sinpleak edo ertainak programatu ahal izatea. Atal honetan erabiltzen den kodearen ezaugarri nagusia argia eta egituratua izatea da, eta ez hainbeste eraginkorra edo trinkoa izatea.

- **Aplikazio edo proiektu errealak**, ariketa luzeagoak eta erabilgarriagoak direnak, askotan modu librean Internetetik eskura daitezkeenak (halakoetan euskaraz komentatuta azalduko dira gure liburu honetan). Bigarren parte honen helburuak lau dira: (1) ezaugarri konplexu batzuk lantzea, (2) proiektu errealak ulertzea, (3) eginda dauden programak aldatu edo egokitzeko gaitasuna lantzea eta (4) sarean informazioa bilatzen trebatzea (programatzean aurkituko diren arazoentzako konponbideak lortzeko, edo behar ditugun programak bilatzeko). Azken hori oso inportantea da gaur egun, sarean baliabide asko baitago eta askotan lizentzia librearekin, eta beraz, halako kasuetan behintzat, ez dezagun gurpila berriz ere asmatu.

Geroago azalduko denez Python lengoaiaren hainbat bertsio daude, eta, duela gutxi arte Python2 hedatuena izan bada ere, etorkizunari begira ez dago zalantzarik: Python3 ikastea merezi du.

Liburuarekin batera adibide eta ariketen kodea eskaintzen da helbide honetan:

<<http://www.unibertsitatea.net/blogak/python>>

Webguneko programa horiek probatzeko eskaintzea eta jendeak berauen gainean hobekuntzak proposatzea izango da webgunearen helburua. Gainera, webgunearekin liburuaren aberasteko asmoa dugu, ariketa berriak proposatuko dira, soluzioak komentatu, erroreak zuzendu...

1.2. HISTORIA

Guido van Rossum herbeheretarrak sortu zuen Python lengoaiaren lehen inplementazioa 1989an Amsterdamgo Matematika eta Informatikako Zentroan (CWI, *Centrum Wiskunde & Informatica*). Lengoiari *Python* izena jarri zion hainbat pelikula komiko kritiko egin dituen Monty Python taldearen omenez.

2003. urteaz gero, Python etengabe egon da gehien erabilitako 10 lengoaien artean TIOBE programazio-komunitatearen sailkapenean, eta *urteko programazio-lengoaia* izendatu zuten 2007 eta 2010 urteetan. 2015eko irailean bosgarrena zen sailkapen horretan.

Bereziki egokia da string-en tratamendu sakona eta hiztegi-bilaketa asko egin behar dituzten programazio-problema bideratzeko. Enpresa handi askotan erabiltzen da Python, besteak beste honako hauetan: Google, Yahoo, CERN eta NASA.

2000. urtean sortu zen Python 2.0 bertsioa ezaugarri berriekin, Unicode erabiltzeko aukerarekin, esate baterako. Geroago 2008.ean sortu zen Python 3.0 (Python 3000 edo py3k ere deitua). Bertsio horrek funtsezko aldaketa batzuk ekarri zituen eta ez da bateragarria aurreko bertsioaren programekin. Python 3.0 bertsioaren ezaugarri berri asko Python 2 lengoaiara ere eraman izan ziren geroago eta Python 2.7 bertsioa sortu zuten horrela.

1.3. ZERGATIK PYTHON?

Python lengoaia sinple, ulergarri eta ahaltsua da. Lengoiak bere barruan dakarren funtzionalitatea ez da oso zabala, baina oso erraza da oinarritzko muin hori zabaltzea. Modulu eta metodo ahaltsu asko erraz aurkitzen dira sarean, eta gehienetan lizentzia libre batekin. Horrela, esate baterako, erraz txerta daiteke interfaze programagarria behar duten aplikazioetan.

Lengoaia interpretatua da, erabiltzeko erraza hasieran. Aginduak banan-banan egikaritu daitezke aldez aurretik konpiladore batek programa osoa konpilatu gabe.

Python software libre eta irekia da. Mozkin-asmorik gabeko fundazio batek (*Python Software Foundation*) kudeatzen du Python lengoiaren ohiko inplementazioa (*CPython*).

Testu-tratamenduetan oso erabilia den Perl lengoiaren filosofia eta Pythonena nahiko diferenteak dira programen idazteko filosofiaren ikuspuntutik. Perl lengoia hainbat modu diferente egoten dira tratamendu bera lortzeko (*There is more than one way to do it*), baina Python lengoiaren filosofia horren kontra doa eta saiatzen da adierazpide bakar eta intuitiboa definitzen (*There should be one—and preferably only one—obvious way to do it*). Helburua da programa sinple, ulergarri eta ahaltsuak lortzea.

Bestalde, webguneak garatzeko ere egokia da Python. Aplikazio horiek liburu honen esparrutik kanpo geratzen badira ere, Django framework-a erabiltzeko, adibidez, derrigorrezkoa da Python jakitea.

Programazio-lengoaia	Zertarako sortua	Agintzailea	Objektuei zuzendua	Funtzionala	Prozedurazkoa	Generikoa	Erreflektiboa	Gertaeretara zuzendua	Beste paradigmak
Ada	Aplikazioak, txertatua, denbora erreala, sistemak	X	X		X	X			konkurrentea, banatua
C	Aplikazioak, sistemak, helburu orokorra, behe-mailako eragiketak	X			X				
C#	Aplikazioak, RAD, gestioa, bezeroaren aldea, orokorra, zerbitzariaren aldea, Web	X	X	X	X	X	X	X	egituratua, konkurrentea
C++	Aplikazioak, sistemak	X	X	X	X	X			
COBOL	Aplikazioak, gestioa	X	X		X				
Cobra	Aplikazioak, gestioa, orokorra, Web	X	X	X		X	X		
Common Lisp	orokorra	X	X	X	X	X	X	X	
Fortran	Aplikazioak, zenbaki-konputazioa	X	X		X	X			
Haskell	Aplikazioak			X		X			ebaluazio alperra
Java	Aplikazioak, gestioa, bezeroaren aldea, orokorra, mobile development, zerbitzariaren aldea, Web	X	X	X	X	X	X		konkurrentea
JavaScript	bezeroaren aldea, zerbitzariaren aldea, Web	X	X	X			X		prototipotan oinaritua
Mathematica	kalkulu sinbolikoa	X	X	X	X				logikoa, banatua
MATLAB	Highly domain-specific, zenbaki-konputazioa	X	X		X				
Oxygene	Aplikazioak	X	X			X			
Pascal	Aplikazioak, hezkuntza	X			X				
Perl	Aplikazioak, scripting, text processing, Web	X	X	X	X	X	X		
PHP	Zerbitzariaren aldea, Web Aplikazioak, Web	X	X		X		X		
Prolog	Aplikazioak, adimen artifiziala								logikoa
Python	Aplikazioak, orokorra, Web, scripting, Adimen artifiziala, kalkulu zientifikoa	X	X	X			X		aspektuetara zuzendua
Ruby	Aplikazioak, scripting, Web	X	X	X			X		aspektuetara zuzendua

1.1. taula. Programazio-lengoaia ezagunen arteko konparazioa (Ingeles Wikipediatik erauzia).

1.4. EZAUGARRI OROKORRAK

Programatzeko paradigma asko erabil daitezke Pythonekin: *Objektuei orientatutako programazioko* eta *programazio egituratuko* kontzeptu guzti-guztiak hartzen ditu bere gain, baita *programazio funtzionaleko* eta *aspektuetara zuzendutako programazioko* hainbat kontzeptu ere. *Programazio logikoa* eta *kontratu bidezko diseinua* ere erabil daitezke lengoaiaren luzapenen bitartez. Lengoiaren ezaugarri horiek kontuan hartuta, 1.1. taulan konparazio bat ikus daiteke beste lengoaia ezagun batzuekin, ingelesezko Wikipediatik erauzi dugu hori.

Python bidez idatzitako kodea oso trinko eta irakurgarria izan ohi da, horri begira diseinatuta dago-eta. Trinkoa da, beste arrazoi batzuez gain duen ezaugarri bati esker: aldagaiak ez dira deklaratu/erazagutu behar; erabili ahala esleitzen baitzaie mota. Hori errore-iturri arriskutsua izan badaiteke ere, interpretatzaileak errore asko detektatzen ditu exekuzio-garaian. Bestalde, interpretazioa eraginkorra izan dadin programen formatua eta programazio-egitura batzuk zurrin samarrak dira. Hauek dira kontuan hartzeko xehetasun garrantzitsu batzuk:

- Tabulazioa. Ikusiko denez, agindu-blokeen mugak tabulazio bidez antzematen ditu interpretatzaileak (C lengoaiako giltzen orde, adibidez). *Tab* teklarekin edo zuriuneekin sortuko da, baina modulu berean era kontsistentean egin behar da, beti berdin, edo beti *Tab* bitartez edo beti zuriune bitartez.
- ':' karakterea: programazio-egitura batzuetan (*if*, *else*, *while* eta *for* esaterako) karaktere hori jarri behar da lehenengo lerroaren bukaeran egiturako espresioaren ondoren (benetako helburua bloke-hasiera markatzea da). Beste programazio-lengoaietan aritutakoei ':' karakterea idaztea ahaztu egiten zaie lengoia honetan hastean.
- Parentesien erabilera. Baldintzak ez dira derrigorrez parentesi artean idatzi behar, baina funtzio-deietan beharrezkoak dira. Python2 lengoian ez ziren beharrezkoak funtzio deietan, baina Python3-n bai. Adibidez, inprimatzeko erabiltzen den *print* funtzioan argumentuak parentesi artean zehaztu behar dira beti, bai ala bai, baina ohitura zaharrak direla-eta horiek ahaztea ohiko errorea izaten da.

Programaren konplexutasuna minimo batetik pasatzen denean, eta programazio-egituratuaren eta objektuei orientatutako programazioaren metodologiari jarraitu ahal izateko funtzio asko definitu behar direnez, funtzio eta definizio lagungarri horien guztien definizioa ondo antolatu behar da. Horrelakoetan aparteko fitxategi batean bildu ohi dira objektu bakoitza definitzeko behar diren funtzio guztiak (klase bat, bere metodoekin), edo elkarren artean oso lotuta dauden hainbat funtzio (nahiz eta klase bat ez osatu). Fitxategi horiei *modulu* esaten zaie. Modulu asko erabil daitezke aplikazio batean, eta beren artean erabilera-menpekotasun ugari ere egon daitezke. Interneten hainbat klase eta modulu publiko aurki daitezke.

1.5. PYTHON2 VS PYTHON3

Python3 bertsioa etorkizuneko Python izango da, zenbait hobekuntza garrantzitsu baitauzka. Baina arazoa da gaur egun oso lagungarri diren hainbat eta hainbat programa-liburutegi oraindik Python2 bertsioan eskaintzen direla. Gauzak ondo badoaz, liburutegi interesgarrienak urte gutxi barru Python3 bertsioan ere eskuragarri egongo dira. Horregatik uste dugu Python ikasi nahi duen hasiberriarentzat egokiena Python3 ikastea dela.

Desberdintasunak ez dira asko, baina noski, arazoak eman ditzakete exekuzioan. Diferentzia garrantzitsuenetako batzuk hauek dira:

- Python3-k hobeto erabiltzen ditu Unicode karaktereak. Hainbat alfabeto desberdin (arabierarena, errusieraren edo japonierarena, esaterako) erabili ahal izateko aukera ematen duen kodetze-sistema da Unicode.
- *print* funtzioan parentesiak beti erabili behar dira Python3 erabiltzen denean, gainerako beste funtzioetan bezala. Hau da gehien-gehienetan nabaritutako duzun diferentzia. Adibidez:

```
# Python3-z ez da onartzen da Python2-ko agindu hau:  
print 'Hello'  
# Python2-z eta Python3-z onartzen da agindu baliokide hau:  
print('Hello')
```

- Beste aldaketa txiki batzuk idazkeran: erroreak jasotzeko salbuespenetan, zenbaki osoen arteko zatiketan, zenbaki oso luzeetan, hiztegietako iterazioetan...

Diferentziei buruz informazio gehiago bilatzeko, eta Python2 eta Python3 bertsioen arteko bateragarritasuna lantzeko, hainbat webgune dago, adibidez helbide haueetan:

- *Python2 or Python3* - Wiki Python
<https://wiki.python.org/moin/Python2orPython3>
- *Easy, clean, reliable Python 2/3 compatibility*
http://python-future.org/compatible_idioms.html

Etorkizunari begira jarrita gure liburuan Python3 aukeratu dugu, noski. Zenbait kasutan aurreko bertsioarekiko desberdintasunak azaltzen dira liburuan, baina ez modu sistematikoan.

1.6. LAN-INGURUNEA

Python programazio-lengoaia interpretatua da, banan-banan egikaritu daitezke aginduak aldeztu aurretik programa osoa konpilatu beharrik gabe, eta, beraz, programa zuzenean proba daiteke. Probatu ahal izateko, noski, Python interpretatzaile bat instalatu egin beharko da.

Python3 lengoaia instalatuta etortzen da Linux sistema eragilearen banaketa guztietan. Windows erabiltzen baduzu, berriz, bi aukera dituzu: instalatu edo *hodeitik* zuzenean erabili. Lehen aukerakoa lortzeko zenbait urrats bete beharko dira (<https://docs.python.org/3/using/windows.html>), eta kontuan hartu beharko dira Windowsen bertsioak. Hodeiarena, berriz, aukera neutroena da, nabigatzailea bat

baino ez baita behar. Aukera horretarako *codeskulptor.org* gunea gomendatzen da ikastaro askotan (baina Python2 erabiltzen du).

Linuxera itzuliz, hasierako programa txikiak martxan jartzeko edo aldeztatik ditugun programen probak egiteko nahikoa da terminal batean *python3* interpretatzailea martxan jartzea eta aginduak edo funtzio-deiak egitea. Adibidez:

```
$ python3
Python 3.4.3 (default, Oct 14 2015, 20:33:09)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("Kaixo guztioi")
Kaixo guztioi
>>>
```

Baina programa ez oso trivialak idatzi nahi badira, hobe da ez idaztea kode osoa zuzenean interpretatzailearekin. Horrelakoetan lan egiteko modu ohikoenak bi hauek izaten dira:

- Programa-fitxategiak osatzea testu-editore batean (*emacs* edo *gedit* esaterako) eta aldi berean paraleloan terminal batean programa horren exekuzioak egitea.
- *idle3* Python ingurunea erabiltzea. Kasu honetan ere beti izango dugu gutxienez bi leiho zabalik, bat programa idazteko eta bestea aginduak interpretatzeko.

Noski, kasu bietan funtsezkoa izango da jakitea editorean nola egin funtzio hauek:

- Lerro baten edo lerro-multzo baten tabulazio-maila aldatzea (Python lengoaiari tabulazio-maila da programaren barne-egitura definitzen duena).
- Lerro bat edo lerro-multzo bat iruzkin bihurtzea.
- Fitxategia gordetzea.
- Fitxategia egikaritzea.

Ariketa: Programa baten fitxategiaren exekuzio-adibide bat gehitzea editorea erabiliz.

Programa editorearekin apur bat praktikatzeko, osa ezazu 2. kapituluko *p2-2.py* fitxategia (*kap2* karpetan dago) bukaeran haren exekuzio bat iruzkin gisa gehituz. Hau da fitxategiak duen testua:

```
izena = input("Sartu zure izena: ")      # sarrera
print("Kaixo", izena)                    # irteera
```

Eta hau da bukaeran fitxategi horretan eduki nahi dugun testua:

```
izena = input("Sartu zure izena: ")      # sarrera
print("Kaixo", izena)                  # irteera

##      >>>
##      Sartu zure izena: Pythondegi
##      Kaixo Pythondegi
##      >>>
```

idle3 editorea erabiltzen baduzu, urrats hauek egin beharko dituzu:

- Terminal batean kokatu *kap2* direktorioan eta exekutatu *idle3* komandoa.
- *p2-2.py* fitxategia zabaltzeko:
(*File > Open...*)

```
izena = input("Sartu zure izena: ")      # sarrera
print("Kaixo", izena)                  # irteera
```

- Zabaldutako fitxategiko programa exekutatzeko (fitxategiaren leihoan):
(*Run > Run Module*) edo *F5* tekla bakarrik

```
Python 3.4.3 (default, Oct 14 2015, 20:33:09)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
Sartu zure izena: Pythondegi
Kaixo Pythondegi
>>>
```

- Exekuzioko testua kopia eta eraman programaren fitxategira, bukaeran jarri.

```
izena = input("Sartu zure izena: ")      # sarrera
print("Kaixo", izena)                  # irteera

>>>
Sartu zure izena: Pythondegi
Kaixo Pythontdegi
>>>
```

- Exekuzioko lerroak aukeratu eta tabulazio-maila bat emanda eskuinera eramateko:
(*Format > Indent region*) edo *ALT+]* tekla-konbinazioa

```

izena = input("Sartu zure izena: ")      # sarrera
print("Kaixo", izena)                    # irteera

>>>
Sartu zure izena: Pythondegi
Kaixo Pythondegi
>>>

```

- Lerro-multzo hori aukeratu eta iruzkin bihurtzeko:
(*Format > Comment out region*) edo *ALT+3* tekla-konbinazioa

```

izena = input("Sartu zure izena: ")      # sarrera
print("Kaixo", izena)                    # irteera

## >>>
## Sartu zure izena: Pythondegi
## Kaixo Pythondegi
## >>>

```

- Fitxategia gordetzeko:
(*File > Save*)

Programaren funtzionamendua egokia ez denean, askotan zaila da jakitea zergatik den eta garrantzi handiko tresna bat erabiltzea gomendatzen da halakoetan: *debugger* edo arazle izeneko tresna. Pythonekin arazle desberdinak erabil daitezke, baina PDB izenekoa da erabiliena. Agian hemengo ariketak osatzeko ez duzu beharko, baina benetako proiektu baterako derrigorrezko tresna da. Hona hemen dagokion eskuliburua: <<https://docs.python.org/3.2/library/pdb.html>>.

1.7. KODEA

Esan bezala, liburu honetan erakusten diren programa guztiak blog honetan eskuragarri daude:

<http://www.unibertsitatea.net/blogak/python>

Programa guztiak daude *kodea* izeneko karpeta batean eta kapituluaren arabera azpikarpetetan antolatu dira (*kap2, kap3...*).

Blog horretan azalduko dira liburuari buruz etorkizunean sortuko diren zehaztasun, zuzenketa, azalpen edo material osagarri berriak ere.

1.8. BIBLIOGRAFIA

Testuan zehar webgune askoko erreferentziak emango dira, baina hemen erreferentzia nagusiak bildu nahi izan ditugu.

Liburuak euskaraz

Arantza Díaz de Ilarraza, Kepa Sarasola. *Oinarrizko programazioa: ariketa bilduma*. Udako Euskal Unibertsitatea. 1999. <<http://ueu.eus/download/liburua/Oinarrizkoprogramazioa.pdf>>.

Aitzol Astigarraga, Koldo Gojenola, Kepa Sarasola, Aitor Soroa. *TAPE Testu-analisirako PERL erremintak*. Udako Euskal Unibertsitatea. 2009. <<http://ueu.eus/download/liburua/PERL.BERRIA.osoa.pdf>>.

Iñaki Alegría, Nestor Garay. *C programazio-lengoaia*. Elhuyar. 1995.

Liburuak (on-line) ingelesez

The Python Tutorial: <<https://docs.python.org/3/tutorial/>>.

Introduction to Python: <http://nbviewer.jupyter.org/github/ehmatthes/intro_programming/blob/master/notebooks/index.ipynb>.

Erreferentziak

Ohiko erreferentzia: <<https://docs.python.org/3/>>.

Kontsultak: <<http://stackoverflow.com>>, <<http://stackoverflow.com/questions/tagged/python>>.

Esan bezala, programazio-lana, gero eta gehiago, bilaketan oinarritzen da: batzuetan proiekturako baliagarri izango diren modulu/klaseak bilatzea, beste batzuetan aurretik norbaitek egindako (antzeko) algoritmoa bilatzea, eta sarritan guri gertatzen zaigun antzeko akatsa/arazoa notifikatu duten mezuak bilatzea, erantzunaz baliatzeko. Azken bilaketarako berebiziko garrantzia du *Stackoverflow* webguneak, programatzaileen sare soziala, laguntza biltzeko eta eskaintzeko diseinatua. Software librerako gero eta erabiliagoa da *github* plataforma (<https://github.com/>), softwarea lankidetzan garatzeko pentsatua.

2. Lehen programak eta oinarrizko datu-motak

2.1. KAIXO GUZTIOI!

Agurtxo bat pantailaratzea izan ohi da programazio-lengoaia batean hasten denean probatzen den lehen programa.

Horretarako, 2.1. programan ikus daitekeenez, Pythonez lerro bakar bat nahikoa da.

```
print ("Kaixo guztioi")
```

2.1. programa: Kaixo guztioi.

Programa hori martxan jarri eta probatzeko hiru aukera hauek ditugu¹:

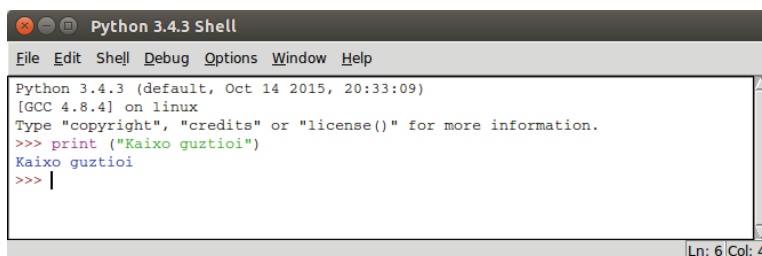
1. `.py` luzapeneko fitxategi batean gorde (`p2-1.py` esaterako) eta `python3` komandoaren bitartez abiaraztea:

```
$ python3 p2-1.py
Kaixo guztioi
```

2. `#!/usr/bin/python3` zehaztea programaren lehen lerroan, exekutatzeko baimena eman (`chmod +x p2-1.py` komandoa) eta programaren izena (bidea zehaztuta) baino ez teklatzea (`./p2-1.py`):

```
$ ./p2-1.py
Kaixo guztioi
```

3. `idle3` ingurunea ireki eta zuzenean idaztea kodea:

A screenshot of a terminal window titled "Python 3.4.3 Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main area shows the following text: "Python 3.4.3 (default, Oct 14 2015, 20:33:09)", "[GCC 4.8.4] on linux", "Type \"copyright\", \"credits\" or \"license()\" for more information.", followed by a prompt ">>>". The user has entered "print ('Kaixo guztioi')", and the output "Kaixo guztioi" is displayed. The prompt ">>>|" is shown on the next line. The status bar at the bottom right indicates "Ln: 6 Col: 4".

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (default, Oct 14 2015, 20:33:09)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print ('Kaixo guztioi')
Kaixo guztioi
>>> |
```

1. Horretarako Pythonez instalatuta egon behar du. Linux Debian eta antzekoetan (Ubuntu esaterako) instalatzeko komandoa hau da: `sudo apt-get install python3`

Programek, erabilgarriak izango badira, datuak irakurri behar dituzte. Hori lortzeko modu errazena teklatutik datuak ematea da. Nola egin daiteke hori Pythonez? *input* funtzioaren bidez (*raw_input* funtzioa Python zaharretan):

```
izena = input("Sartu zure izena: ") # sarrera
print("Kaixo", izena)             # irteera
```

2.2. programa. Kaixo pertsonalizatua.

2.2. programa exekutatzean zure izena sakatu behar duzu agur pertsonalizatua jaso baino lehen. Proba ezazu.

Edozein lerrotan '#' karakterea azaltzen bada, eskuinaldean datorren testua iruzkin edo ohartzat hartuko da. Beraz, interpretatzaileak ez dio kasurik egingo. '#' karakterea lerroko lehen karakterea bada, orduan lerro osoa hartzen da iruzkintzat. Beraz, 2.2. programan agertzen diren *sarrera* eta *irteera* hitzak ez dira kontuan hartuko programa exekutatzerakoan.

Behin programa bat ondo dabilela ikusi dugunean, oso praktikoa izaten da programaren bukaeran exekuzio bat(zuk) gehitzea, programaren erabilera dokumentatzeko.

Adibidez, 2.2. programa honela osa liteke:

```
izena = input("Sartu zure izena: ") # sarrera
print("Kaixo", izena)             # irteera

#   >>>
#   Sartu zure izena: Pythondegi
#   Kaixo Pythondegi
#   >>>
```

2.2. ALDAGAIK, KONSTANTEAK ETA DATU-MOTAK

Jakina denez, programaren exekuzio osoan balio bera duten datuei *konstanteak* esaten diegu. Datu horiek erabili behar ditugunean zuzenean idazten ditugu ("*Kaixo*" katea, edo *13* zenbakia adibidez), edo identifikadore bat erabiltzen dugu programan zehar balio hori zein den hobeto ulertzeko, adibidez:

```
pi = 3.14
jaio_urtea = 1994
```

Aldagaiek, ordea, ez dute beti balio bera, baina identifikadore batez ezagutzen dira eta balio-aldaketak asignazio edo esleipen baten bitartez egin daitezke. Beste lengoaia batzuetan aldagai bat erabili baino lehen beharrezkoa da aldagai horrek zein motatako balioak hartu ahal izango dituen zehaztea, ea haren balioa zenbaki oso bat izango den edo zenbaki erreal bat, edo karaktere bat edo beste edozer. Horri esaten zaio aldagaia erazagutzea. Baina Python lengoaiari horrelakorik ez

da behar, zuzenean erabil daitezke aldagaiak lehenago erazagutu gabe; programen interpretatzaileak inplizituki inferituko du aldagaiaren mota aldagai horri esleituko zaion balioaren arabera.

Beraz, aldagai bakoitza balio mota batekin lotzen du interpretatzaileak, eta balio mota hori memorian errepresentatzeko behar den bit-segida egokia erabiliko du. Aldagai bati balio bat esleitzen zaionean erabakitzen da haren mota.

Konstanteei dagokionez, programazio egituratuak gomendatzen duen irizpide bat gogoratu nahi dugu: konstanteak identifikadoreekin (balioa aldatzen ez duten aldagaiak) erabiltzearena hain zuzen. Irizpide horren arabera, konstanteak zuzenean erabili beharrean, konstanteei hasieran izen edo identifikadore bat eman behar zaie eta geroko erabilera guztietan identifikadore hori erabili beharko da. Horrela, programak irakurgarritasunean irabaziko du, eta inoiz konstantearen balioa aldatu beharko balitz, behin bakarrik aldatu beharko litzateke, izen-ematean.

Adibidez, BEZ zergaren balioa erabili behar bada programa batean, behin bakarrik definituko da, hasieran. Aldatu beharko den egunean hasierako balio hori aldatzea izango da egin beharko den aldaketa bakarra.

```
BEZ = 0.21
```

Edozein kasutan Pythonek ez ditu konstanteak eta aldagaiak bereizten, programatzailearen lana da bereizketa hori egitea. Hauxe da horretarako proposatzen dena: konstanteen izenak letra larriz idaztea eta global moduan definitzea; aldagaiak, berriz, lokalak eta letra xehez (edo denak xehez baina lehen letra larriz).

Datu-motak

Datu-moteetara itzuliz, Pythonek ondoko oinarrizko datu-motak ezagutzen ditu (2.1. taula):

- Karaktere-kateak edo *string*ak (*str*). Adibidez: *'Kaixo'* edo *'emaitza'* edo *"Agur"*. Komatxo sinpleak zein bikoitzak erabil daitezke (Pythonek ez ditu bereizten). Kate hutsa ere izan daiteke.
- Osoko zenbakiak (*int*). Adibidez: *12* edo *-5*.
- Zenbaki errealak (*float*). Adibidez: *3.14* edo *-0.5*.
- Boolearrak (*bool*). Bi balio baino ez ditu onartzen *True* (egiazkoa) eta *False* (faltsua).
- Zerrendak. Adibidez: *[31, 28, 31, 30]* edo *['positibo', 'negatibo', 'zero']*. *[]* zerrenda hutserako erabiltzen da, hasieraketetan batez ere. Zenbakienak, karaktere-kateenak eta bestelako osagaiz osatuta egon daitezke, baita elementu heterogeneoz ere.
- Baliorik gabekoak (*None*).

KODEA	ADIBIDEA	ESANAHIA
str	'Euro'	karaktere-katea
int	1024	osoko zenbakia
float	3.14	zenbaki erreala; kontuz, puntu hamartarra (ez koma)
bool	True	<i>egiazkoa</i> edo <i>faltsua</i>
[]	['ireki', 'itxi']	zerrenda (osagaiak komaz bereizita zehaztu behar)
None		mota hutsa

2.1. taula. Oinarrizko datu-motak.

Beste oinarrizko mota batzuk badaude, hiztegiak eta tuplak esaterako, baina konplexuagoak direnez, hurrengo kapituluetarako geratuko dira, batez ere 6.erako.

Datu-motak bereiztea garrantzitsua da programazioan, konputagailuan modu oso differentean errepresentatzen baitira zenbaki osoak, errealak, eta karaktereak esate baterako. Karaktere bat errepresentatzeko nahiko izan daitezke 8 bit, baina zenbaki erreal bat errepresentatzeko ez da nahikoa, gehiago behar izaten dira, 64 adibidez. Bestalde zenbaki osoekin erabil daitezkeen eragiketak ezin daitezke erabili karakterekin. Zein litzateke *'kaixo'*'jon'* eragiketaren emaitza? Ez dauka zentzurik. Beraz, aldagai baten mota bat finkatzen dugunean finkatzen ari gara aurrerantzean aldagaiak hartu ahal izango dituen balio posibleen mota (aldagaiaren mota zenbaki errealena bada, ezin izango diogu jarri karaktere bat balio gisa), eta finkatzen ari gara aldagaia zein eragiketarekin erabili ahal izango dugun eta zeinekin ez.

Aldagaiei hasierako balio bat esleitu behar zaie, horrela balioaren mota aldagaiari lotzen baitzaio. Esleipenak idazteko eredua hau da:

```
aldag_izena = balioa
```

Hiztegi eta zerrenda motetako aldagaien balioak funtzio batez ere esleiri daitezke; *dict()* funtzioa erabiltzen da hiztegiatarako eta *list()* zerrendatarako. Funtzio eraikitzaileak esan ohi zaie azken horiei.

Ondoko lerroan karaktere-kate motako *mezul* aldagaia definitzen da, balioa karaktere-katea da eta:

```
mezul = 'Eskerrik asko'
```

Karaktere-kateak komatxo bakunen artean zein komatxo bikoitzen artean idatz daitezke (biak erabat baliokideak dira, beste lengoaietan ez bezala). Karaktere bakunak karaktere-kateen kasu partikularrak dira. Hona hemen karaktere-kate motako adibide batzuk: *'a'*, *"eguna"*, *'Euro'*, *"John's house"*².

2. Komatxo sinplea kate barruan dagoela adieraztea errazten du hemen komatxo bikoitzaren erabilerak.

Karaktere ez-inprimagarriak (*lerro-bukaera*ko karakterea adibidez), ordea, ezin dira teklatuaren karaktere bakunen bidez adierazi. Hori dela-eta, Pythonek alderantzizko barra ('\ ' karakterea) duten konstanteak sortu ditu (C lengoaiarenak ere halaxe dira), 2.2. taulan ikus daitezke batzuk. Alderantzizko barra duen karaktere berezi bat beste edozein karaktere bezala erabil daiteke karaktere-kateetan.

KODEA	ESANAHIA
\n	lerro-bukaera
\'	komatxo bakuna
\"	komatxo bikoitza
\t	tabulazioa
\\	alderantzizko barra

2.2. taula. Ihes-karaktereen erabilera.

Lerro bat baino gehiago hartzen duten esaldietan ''' (hiru komatxo) erabiltzea gomendatzen da. Adibidez:

```
bertsoa = '''Baga, biga, higa,
laga, boga, sega,
zai, zoi, bele,
arma, tiro, pum!'''
```

Hiru komatxoak hasieran eta bukaeran jartzen dira, tartean hainbat lerro sar daitezkeela. Lerro-bukaera adierazten duten karaktereak ere biltegitratuko dira.

Balio bat beste mota batera pasatzeko bihurketa esplizituak

Bihurketa esplizitua adierazteko datu-motari dagokion kodea funtzio gisa erabiltzen da. Adibidez, karaktere-kate bat eta zenbaki erreal bat zenbaki oso bihurtzeko *int()* funtzioa erabiliko da, jatorrizko espresioa parametro gisa zehaztuz. Ildo beretik datu bat karaktere-kate bihurtzeko *str()* erabiliko da. Adibidez:

```
PI = 3.14
sarrera = '5'
pistr = str(PI)           # pi zenbakia karaktere-kate moduan
zenb = int(sarrera)       # '5' karakterea zenbaki oso moduan erabiltzeko
```

2.3. PROGRAMA SINPLEAK. AGINDU-SEGIDAK

Programa bat agindu-segida ordenatu bat da. Programa sinpleenetan hiru moduko aginduak baino ez daude:

- datuak irakurtzeko `input` modukoak
- esleipenak (kalkuluak egiteko eta aldagaietan gordetzeko)
- emaitzak inprimatzeko `print` modukoak

Programa bat exekutatzea (martxan jartzea edo egikaritzea ere esaten dugu) bere aginduak banan-banan (programan dauden ordena berean) exekutatzea da.

2.3. programan kapitulu honetan azaldu diren hainbat kontzeptu agertzen dira, adibide praktikoen bitartez. Iruzkina hutsak diren lerroak kenduta, programan sei lerro daude, eta bakoitzean agindu bat, lehenengo laurak esleipenak dira eta azken biak emaitzak idazteko `print` modukoak.

```
# Konstantea
PI = 3.14159
# Irakurketa eta bihurketa
r = float(input("Sakatu erradioaren neurria: "))
# Kalkuluak
perimetroa = 2 * PI * r
azalera = PI * r * r
# Idazketa
print ("Azalera:", azalera)
print ("Perimetroa:", perimetroa)
```

2.3. programa. Konstanteak, aldagaiak eta mota-bihurketak, oinarrizko geometriaren erabilera.

3. Eragileak eta espresioak

Aurreko kapituluan ikusi dugu datu sinpleak nola errepresentatu programa batean, datu horiek nola definitu eta identifikatu aldagai eta konstanteen bidez, eta zein diren datu-mota ohikoenak. Baina nola erabili aldagai eta konstante horiek kalkulatu nahi ditugun espresioak osatzeko? Hori da kapitulu honen helburua.

3.1. ESPRESIOAK

Python lengoaian eragiketa bat kalkulatu behar denean, eragile bat erabiltzen da. Aldagaiak edota konstanteak eragileekin konbinatuz zehazten da nola kalkulatu balio bat, aldagaien uneko balioekin eta programako agindu baten barruan. Formula moduko horiei espresioak esaten diegu.

Espresio bat eragigai batez edo gehiagoz, eta zero edo eragile gehiagoz osatuta dago, dena batera balio bat kalkulatzeko helburuarekin. Horrela, ' $a+2$ ' espresio bat da, a aldagaiak duen balioa eta 2 zenbakia batuz lortzen den emaitza itzultzen duena. Espresio bat da a aldagaia bera bakarrik ere, aldagaiak momentuan duen balioa itzultzen duena. 2 konstantea ere bera bakarrik beste espresio bat da.

Espresio bakoitzari mota bat dagokio beti, kalkulatu duen balioaren motarena.

Aurreko kapituluetako programetan ohiko eragile batzuk erabili ditugu: + (batuketa), - (kenketa) eta * (biderkaketa) eragileak. Baina badira beste eragile asko ere. Banan-banan aztertu baino lehen, eragileen bi ezaugarri nagusi aztertuko ditugu: lehentasuna eta elkargarritasuna.

3.2. ERAGILEEN ARTEKO LEHENTASUNA

Espresio batean eragile bat baino gehiago dagoenean, zein ordenatan ebaluatuko dira? Adibidez, $2+3*4$ espresioa kalkulatu, zein izan beharko du emaitzak?

$2+3*4 \rightarrow 5*4 \rightarrow 20$?

Ez, ez da? Emaitzak 14 izan beharko du.

$2+3*4 \rightarrow 2+12 \rightarrow 14$

Pertsonok ondo dakigu hori, baina nola zehaztuko dugu hori programetako espresioetan? Batuketa bat eta biderketa bat maila berean badaude, biderketa kalkulatu behar dugu lehenago. Bada, horixe da eragileen lehentasuna. Konputagailuari zehazten diogu eragile bakoitzaren lehentasuna, maila berean dauden eragiketen artean lehenago zein kalkulatu behar duen jakin dezan.

Adibidez, $2+3*4$ espresioaren ebaluazioak 14 sortzen du, biderketak batuketak baino lehentasun-maila handiagoa duelako. Parentesiek, beste lengoaietan bezala, ebaluazioaren ordena aldatzen dute. Horrela, $(2+3)*4$ espresioaren ebaluazioak 20 balioa sortzen du. Ebaluazio-ordena eragileen lehentasunaren arabera gertatzen da. Lehentasun handieneko eragiketak burutzen dira lehen, tarteko emaitzak lortuz. Geroago, eta lehentasunaren ordena beheranzkorrari jarraituz, beste eragileak aplikatzen dira bitarteko emaitzen gainean.

Ebaluazioaren ordenaren inguruan zalantzak edukiz gero parentesiak erabiltzea gomendatzen da, horrela, bide batez, errazago ulertuko da programa, haren irakurgarritasuna handiagoa izango baita.

3.3. ERAGILEEN ARTEKO ELKARGARRITASUNA

Espresio batean eragile bat baino gehiago dagoenean, edozein datu edozein eragilerekin aplikatu daiteke? Adibidez, $3+'kaixo'$ espresioa zilegi da? Posible da zenbaki baten eta karaktere-kate baten arteko batuketa? Ez, ez da? Txikitan esaten ziguten ezinezkoa zela sagarrak eta madariak batzea. Antzekoa gertatzen da programazioan: eragile bakoitzak ondo zehaztuta dauka bere eragileek zein datu-motatakoak izan behar duten, eta eragiketaren emaitza ezin izango da lortu, eragigaiak mota horretakoak badira. Beraz, kontuz espresioak idazterakoan! Eragileen arteko elkargarritasuna kontuan hartu beharko da espresioak kalkulatzekoan, espresio baten emaitza beste batentzako datua izango denean lortutako emaitza hark datu egokia izan beharko du bestearentzat.

Adibidez, ondoko espresio hau prozesatzean interpretatzaileak errorea emango du:

```
'8'+7
```

Baina ondoko hiru hauetan ez³:

```
8+7
int('8')+7
'8'+str(7)      # kasu honetan + sinboloak kateaketa adierazten du
```

Dena den eragile batzuen semantika oso zabala da eta, geroago ikusiko dugunez, ahaltsua bezain korapilatsua izan daiteke. Horrela, biderketa (*) karaktere-kate bati aplikatu dakioke (katea hainbat aldiz errepikatuz) edo batuketa (+) kateen arteko kateaketarako erabil daiteke.

3. Ikusiko dugunez, + eragilea karaktere-kate artean aplikatu daiteke kateatzea adierazteko.

3.4. ESLEIPENA

Esleipena aldagai baten balioa aldatzeko erabiltzen da, eta Python bezalako programazio-lengoaia agintzaileen funtsezko eragiketa da. Esleipen-espresio baten eredu orokorra honakoa da:

$$\text{aldagai_izena} = \text{espresioa}$$

Noski, '=' karakterearen ondoan azaltzen den espresioa edozein izan daiteke: konstante bakar batez, aldagai bakar batez, edo konstante, aldagai eta eragilez osaturiko edozein espresio.

Esleipenaren bi aldeetako datuak mota berekoak izatea gomendatzen da. Ezkerreko aldagaia programan azaltzen den lehen aldia baldin bada, eskuineko espresioak zehazten du aldagai horrek hartuko dituen balioen mota. Mota esplizituki aldatzeko aurreko kapituluak azaldutako bihurtak adieraz daitezke (*str()*, *int()*, *float()*)

3.5. KARAKTERE-KATEEN GAINEKO ERAGILEAK

Esan dugun moduan *str* mota oinarrizko mota da. Mota horretako aldagaiak eta konstanteak konbina daitezke esleipenetan edo kateaketaren eragilea erabilita (+). Beraz, garrantzitsua da ulertzea + eragileak esanahi bikoitza duela: zenbakien artean batuketa adierazten du, baina karaktere-kateen artean kateaketa adierazten du. Biderketa ere aplika daiteke kateen gainetik edukia errepikatzeko. 3.1. programan adibideak ditugu.

```
lengoaia = 'Python'
mezua = lengoaia + ' ' + 'ikasten'
print (mezua)
print (2*lengoaia)
```

3.1. programa. Karaktere-kateen adibideak.

Programa exekutatzuz gero, emaitza hau idaztea izango da:

```
Python ikasten
PythonPython
```

3.6. ERAGILE ARITMETIKOAK

3.1. taulan ohiko eragiketak agertzen dira: batuketa (+), kenketa (-), biderketa (*) eta zatiketa erreala (/). Plus eta minus zenbaki bakar batekin eta aurritzki moduan erabilita zeinu aritmetikoa adierazi ahal da (positiboa eta negatiboa, hurrenez hurren). Horrez gain, beste eragiketa hauek ere agertzen dira 3.1. taulan:

- Modulua (%). Bi zenbaki osoren arteko zatiketa osoaren hondarra kalkulatzen du.
Adibidez: $12 \% 3 \rightarrow 0$; $13 \% 3 \rightarrow 1$; $14 \% 3 \rightarrow 2$; $15 \% 3 \rightarrow 0$
- Zatiketa osoa (/). Bi zenbaki osoren arteko zatiketa osoa edo modulua kalkulatzen du.
Adibidez: $12 // 3 \rightarrow 4$; $13 // 3 \rightarrow 4$; $14 // 3 \rightarrow 4$; $15 // 3 \rightarrow 5$
- Berreketa edo esponentziala (**).
Adibidez: $25^{**}2 \rightarrow 625$; $25^{**}0.5 \rightarrow 5.0$

ERAGIKETA	ERAGILEA	FORMATUA	AZALPENA
batuketa	+	x + y	x gehi y
kenketa	-	x - y	x ken y
biderketa	*	x * y	x bider y
zatiketa (erreal)	/	x / y	x zati y (emaitza beti da erreal)
zatiketa (osokoa)	//	x // y	x zati y (emaitza beti da osokoa)
modulua	%	x % y	x zati y eragiketaren hondarra
berreketa	**	x ** y	x ber y

3.1. taula. Eragile aritmetikoak.

Zatiketa osoa eta modulua, // eta % eragileak, osoko eragigaiekin baino ezin dira erabili, eta emaitza zenbaki oso bat izango da beti. Osokoekin / eta % eragileen erabilerak argitzeko, 3.2. programa proba daiteke.

```
# Zatidura eta zatidura osoa. Hondarra.
x1 = 11
x2 = 8
y = 4
print("Zatidura:", x1/y, x2/y)
print("Zatidura osoa eta hondarra:", x1//y, x1%y)
```

3.2. programa. Zatiketa osoa eta erreal.

Programa horren emaitzak hauek dira:

```
Zatidura: 2.75 2.0
Zatidura osoa eta hondarra: 2 3
```

Bestalde, eragile aritmetiko nagusiak (+, -, * eta /) metagailu moduan erabil daitezke +=, -=, *= eta /= eragile berriak sortuz. 3.2. taulan baliokidetzak azaltzen dira.

ERAGILEA	FORMATUA	BALIOKIDEA
$+=$	$x += y$	$x = x + y$
$- =$	$x -= y$	$x = x - y$
$* =$	$x *= y$	$x = x * y$
$/ =$	$x /= y$	$x = x / y$

3.2. taula. Eragile aritmetikoak metagailu gisa⁴.

3.7. ERAGILE LOGIKOAK ETA ERLAZIOAK

Erlazio-eragileak balioen arteko konparazioak adierazteko erabiltzen dira, adibidez balio bat beste bat baino handiagoa den aztertzeko. Emaita beti izango da balio boolear bat: *egiazkoa* (bai, handiagoa da) edo *faltsua* (ez, ez da handiagoa).

Balio boolearrak konbina daitezke eragile logikoekin beste balio boolear bat lortzeko. Adibidez: $(a > b)$ and $(a > c)$. Espresio logiko horrek aztertzen du ea a aldagaiaren balioa b -rena eta c -rena baino handiagoa den.

Espresio boolearrak baldintzak adierazteko erabiltzen dira batik bat (ikus 3.3. taula). Hurrengo kapituluaz azalduko diren programazio-egituretan (baldintzapekoak zein errepikazkoak).

ERAGIKETA	ERAGILEA	FORMATUA	AZALPENA
handiago	$>$	$x > y$	Ea x -ren balioa y -rena baino handiagoa den
txikiago	$<$	$x < y$	Ea x -ren balioa y -rena baino txikiagoa den
berdin	$==$	$x == y$	Ea x -ren balioa eta y -rena berdinak diren
ezberdin	$!=$	$x != y$	Ea x -ren balioa eta y -rena ezberdinak diren
handiago edo berdin	$>=$	$x >= y$	Ea x -ren balioa y -rena baino handiagoa edo berdina den
txikiago edo berdin	$<=$	$x <= y$	Ea x -ren balioa y -rena baino txikiagoa edo berdina den
eta	and	$b1 \text{ and } b2$	Ea $b1$ -en balioa eta $b2$ -rena biak <i>egiazkoa</i> diren
edo	or	$b1 \text{ or } b2$	Ea $b1$ -en balioa edo $b2$ -rena <i>egiazkoa</i> den, bietako bat gutxienez
ez	not	$\text{not } b1$	Ea $b1$ -en balioa faltsua den

3.3. taula. Erlazioak adierazteko eragileak.

Kontuz! Pythoneko bertsio zaharragoetan *and*, *or* eta *not* eragileen ordez $\&\&$, $\|$ eta \sim eragileak erabiltzen ziren. Kontu handia eduki behar da berdintasuneko erlazioarekin; oso erraza baita $'=='$ eragilea jarri beharrean $'='$ (esleipena) jartzea, beste lengoaietan egiten den bezala. Zorionez, nahi gabe baldintza bat idaztean errore hori eginez gero, interpretatzailea gai da errorea detektatzeko, eta hori zuzentzeko eskatuko du.

4. $+=$ eta $*=$ karaktere-kateekin ere erabil daitezke, kateatzeak eta errepikapenak bideratzeko.

Dena den, Python lengoaiaren edozein espresio erabil daiteke baldintza moduan, espresio aritmetikoak adibidez. Espresioaren emaitza *None* denean, faltsutzat jotzen da, eta, gainerako kasuetan, ordea, egiazkotzat.

Adibide batzuk azter ditzagun:

```
a < b      # egiazkoa a b baino txikiagoa bada
a > b      # egiazkoa a b baino handiagoa bada
```

Aurreko kapituluan azaldutako *True* eta *False* konstanteak ere mota booleanrekoak dira eta, noski, baldintzak adierazteko erabil daitezke. Adibidez, inoiz bukatuko ez den errepikapen infinitu bat programatu nahi badugu, *True* balioa izango da egitura errepikakorren baldintza. Denbora errealeko aplikazioetan, esaterako, erabiltzen dira horrelakoak.

3.8. BIT-MANEIUA

Python lengoaiaren ezaugarrien artean, memoriako bitak atzitzeko gaitasuna duela aipatu behar da, eta horretarako 3.4. taulan azaldutako eragileak dauzkagu. Ez dira aplikazio askotan erabiltzen baina interesgarriak dira behe-mailako programazioari begira.

Jakina denez, AND zenbait bit 0 egoeran jartzeko erabili ohi da, OR eragiketa 1 egoeran jartzeko eta XOR eragiketa biten egoera aldatzeko. Horretarako, aldagaia kode hamaseitarrez (0x aurrizkia) adierazitako maskara batekin parekatu ohi da. Mantendu nahi diren bitei 0 balioa egokitzen zaie maskaran, eta batean jarri (OR) edo aldatu (XOR) nahi direnei bat balioa. AND eragiketetan alderantziz egin behar da, hau da, mantendu nahi diren bitei 1 balioa egokitzen zaie eta horretarako ~ eragilea, osagarri bezala, erabil daiteke.

ERAGIKETA	ERAGILEA	FORMATUA	AZALPENA
AND	&	x & mask	x AND maskara
OR		x mask	x OR maskara
XOR	^	x ^ mask	x XOR maskara
ukapen logikoa	~	~x	EZ x
ezkerrerako desplazamendua	<<	x << n	x-ren bitak n posizio ezkerrerantz
eskuinerako desplazamendua	>>	x >> n	x-ren bitak n posizio eskuinerantz

3.4. taula. Eragile bitarrak.

Adibidez, *n* aldagaia 32 bitez osatuta egonda (haien balioa jakin gabe), eta ezkerreko bita batean jarri eta eskuinekoa zeroan jarri nahi baditugu, ondokoa egin behar da:

```
# n <- xxxx ... xxxx
n = n | 0x80000000 # n <- 1xxx ... xxxx
n = n & (~0x00000001) # n <- 1xxx ... xxxx0
```


Eragile hauen erabilera argitzeko, 3.5. taulan adibide batzuk azaltzen dira, marraren gainean dauden balioak eragigaiak eta behean daudenak emaitzak izanik.

$$\begin{array}{rclcl}
 10101010 & & 10101010 & & 11110000 \\
 \& \underline{01010101} & | & \underline{01010101} & ^ & \underline{10101010} & \sim & \underline{01010101} \\
 00000000 & & 11111111 & & 01011010 & & 10101010
 \end{array}$$

3.5. taula. Bit-maneiturako eragileen erabilera: adibideak.

Desplazamenduek bit-maneiua egitura errepikakorretan erabiltzeko balio dute batez ere, 4. kapituluan ikusiko dugunez.

3.9. OINARRIZKO FUNTZIO BATZUK

Aurrerago ikusiko dugunez, balio bat kalkulatzeko 'funtzioak' erabil daitezke, alegia, beste programa lagungarri batzuk. Funtzioen eta metodoen erabilera 5. kapituluan azalduko den arren, datu-motekin lotutako bi funtzio interesgarri azalduko ditugu orain: *type()* eta *len()* funtzioak.

Datu-moten inguruan zalantza izanez gero, *type* funtzioa erabil daiteke programan bertan edota *idle* ingurunean. Parentesi artean aldagai baten identifikadorea edo konstante bat jarrita, dagokion mota itzultzen du. Adibidez, honako kode honen bidez:

```
print(type(PI))
```

Lehen azalduetako PI konstantearen mota lortu eta inprimatu egingo da:

```
<class 'float'>
```

str motako aldagai/konstanteetan eta zerrendetan *len()* funtzioa erabil daiteke, karaktere kopurua eta osagai kopurua lortzeko, hurrenez hurren. Adibidez:

```
mezu = "Kaixo guztioi"
print(len(mezu))
```

kodeaz 13 zenbakia lortuko da.

Esan bezala, zerrendekin ere erabil daitezke. Adibidez, ondoko kodearen ondorioz:

```
aukera2 = ['bai', 'ez']
print(type(aukera2))
print(len(aukera2))
```

hauxe inprimatuko da:

```
<class 'list'>
2
```

3.10. PROGRAMEN TESTUAREN FORMATUA: TABULAZIOAK ETA ZURIUNEAK

Lehen kapituluan programen egituraz azaldu dena gogoratu behar da: programen sintaxia nahiko zurruna da, eta tabulazioak berebiziko garrantzia du Python lengoaian. Tabulazioa *Tab* tekla bitartez edo zuriuneekin sortuko da, baina modulu berean modu kontsistentean egin behar da, edo beti *Tab* bitartez edo beti zuriuneen bitartez. Programa bat agindu-segida bat da eta normalean programako aginduak lerro banatan idazten dira, denak tabulazio-maila berean hasita. Agindu bat konplexua bada (4. kapituluan ikusiko ditugun kontrol-egiturak, adibidez) eta haren barruan beste agindu-multzo bat sartzen bada, haren barruko agindu horiek guztiak beste tabulazio-maila batekin barrurago idatzi beharko dira beti. Hurrengo kapituluaren adibideetan argiago ikusteko aukera egongo da.

Alderantzizkoa ere egin daiteke: lerro berean agindu bat baino gehiago biltzea. Horretarako, sententzien artean ';' karakterea (puntu eta koma karakterea) jarri beharko da.

Bestalde, programak hobeto irakurri ahal izateko, zuriuneak gehi daitezke espresioetan, beti kontuan hartuz ezin direla txertatu karaktere anitzeko eragileen artean zein identifikadoreen barruan. Parentesiak erabiltzea ere komenigarria da, kalkulatu behar diren espresio luzeen ebaluazioaren ordena argitzeko, bai norberarentzat, baita programa irakurriko duen edonorentzat ere. Erredundanteak izan daitezkeen parentesiak erabiltzeak ez du errore-abisurik sortzen, ezta espresio baten ebaluazioan abiadura moteltzen ere.

4. Kontrol-egiturak

Kontrol-egituren bidez programako agindu bat baldintza baten arabera exekutatzeko ala ez, edo agindu batzuen exekuzioa hainbatetan errepikatzeko aukera dago. Aurreko kapituluan aztertutako programak guztiz linealak eta sekuentzialak dira; bakarrik balio dute formulak kalkulatzeko, espresioak kalkulatzeko. Baina programako aginduen exekuzioa modu ez-linealean ere egin daiteke, horretarako balio dute kontrol-egiturek. Programako aldagaien balioen arabera zenbait agindu exekutatu ala ez kontrola dezakegu baldintzazko egiturak erabiliz, edota agindu-multzo baten egikaritzapena hainbatetan errepika dadin kontrola dezakegu egitura errepikakorrek erabiliz. Programetako aginduen exekuzioa kontrolatzeko egitura horiek Python lengoaiatz nola idatzi eta erabiltzen diren azaltzea da kapitulu honen helburua.

Baldintzapeko egituren aldetik *if* kontrol-egitura da ohikoena. Espresio logiko baten balioaren arabera agindu bat edo beste agindu alternatibo bat exekutatuko da.

Agindu bat edo agindu-multzo bat hainbat alditan errepikatzea nahi bada, orduan egitura errepikakorrek erabiliko dira. Errepikapenen kopurua hasieran jakina den kasuan edo ez-jakina denean, kontrol-egitura desberdinak erabiliko dira. Jakina bada, errepikapenen kopurua zenbatuko duen kontrol-aldagai bat erabiliko da, soilik errepikapen horiek egikari daitezten. Ez-jakina bada, baldintza baten menpe egotea edo begizta infinitua izatea gerta daiteke. Horrelakoetan, *while* kontrol-egitura da erabili ohi den egitura errepikakorra, aurrekorako, berriz, *while* edo *for* egiturak.

Kontrol-egitura oso bat (*if*, *for* edo *while* erakoak) agindu moduan erabil daiteke programetan. 2. kapituluan esan genuen programa bat agindu-segida ordenatu bat dela, eta programa sinpleenetan hiru moduko aginduak baino ez zeudela:

- datuak irakurtzeko `input` modukoak
- esleipenak (kalkuluak egiteko eta aldagaietan gordetzeko)
- emaitzak inprimatzeko `print` modukoak

Kontrol-egiturak ere beste aginduak (sententziak) dira. Kapitulu honetan kontrol-egitura horiek guztiak deskribatuko dira, eta egitura horiekin egindako programak azalduko dira, ezaugarriak ezagutzeaz gain egitura horiei dagozkien aplikazioak ere erakusteko asmoz. Azkenik, kasu berezi batzuetan egitura errepikakorren baldintza sinplifikatzearen badaude bi agindu berezi errepikatze-kontrollean eragiteko: *continue* eta *break*. Horien erabilera ere azalduko da.

4.1. IF KONTROL-EGITURA

if egiturak balio du agindu-multzo bat soilik kasu batzuetan egikaritzeko, baldintza bat betetzen denean, alegia. Kontrol-egitura hau programazio-lengoaia guztietan dago. Hala ere, Python lengoian erabiltzen denean, idazkeran (sintaxian) bi diferentzia azpimarratu behar dira:

- Baldintza ondoren ':' karakterea idatzi behar da, bai eta *else* hitz erresebatuaren ondoren ere.
- Esan den bezala, tabulazioaren erabilera oso zurruna da Pythonen, programako blokeen egitura ondo islatu behar da tabulazioaren bitartez. Agindu-bloke berri bat tabulazio batez bereizten da, eta blokeko agindu guztiak lerrokatuta azaldu behar dira tabulazio-maila berdinarekin.

Egituraren eredu hau da:

```
if baldintza:
    aginduak
else:
    aginduak
```

Baldintza espresio bolear baten bidez (ikus aurreko kapitulua) adierazten da. Baldintza betetzen bada, segidan duen agindu-multzoa egikaritzeko da. Baldintza bete ezean, *else* adarra baldin badago adar horri dagokion agindu-multzoa egikaritzeko da. Ez badago *else* adarra, berriz, egitura honetan ez da ezer egingo.

Hautazkoa da *else* adarra idaztea. Adibidez, ondoko programa-zatian agertzen den *if* egitura sinplea da:

```
if n < 0:
    n = -n;           # Balio absolutua
```

Aldiz, honako zati honetan egitura osoa dugu:

```
# Zenbaki positiboaren balioaren erdia
if n < 0:
    print ("Errorea: negatiboa")
else:
    em = n/2
    print (em)
```

4.1. programan adibide oso bat dugu, bi zenbakiren artean handiena kalkulatzeko duena.

```
# zenbaki handiena kalkulatzeko
a = int(input("sakatu bi zenbakietako bat: "))
b = int(input("sakatu bestea: "))
if a > b:
    hand=a
else:
    hand=b
print("handiena:", hand)
```

4.1. programa. Baldintzazko egituraren adibidea, bi balioren artean handiena kalkulatzeko duena.

Ondoko ñabardura hauek hartu behar dira kontuan:

- Baldintzazko espresioan berdintasuna adierazteko `==` eragilea erabili behar da eta ez `=` eragilea (esleipenetan bakarrik erabiltzen da hori).
- `!=` eragilea erabiltzen da balio desberdinak bereizteko.
- Baldintzazko espresioa *and*, *or* eta *not* eragileekin osaturiko baldintza konposatua izan daiteke.
- Eragilerik erabiltzen ez bada, alegia, aldagai baten izena besterik erabiltzen ez bada *if* egituraren baldintza gisa, emaitza egiazkoa izango da aldagaiaren balioa *None* edo hutsa⁵ ez den bitartean.
- *if* egitura baten barruan beste *if* egitura bat egon daiteke, horrelakoetan *if habiaratua* esaten diogu bigarrenari. Horren ondoren *else* bat aurkitutakoan, lehenengoari edo bigarrenari ote dagokion zalantza egon daiteke. Horrelakoetan tabulazioa da agintzen duena. 4.2. programan adibide bat ikus daiteke.

```
# if habiaratuak
x = int(input("Sakatu x: "))
y = int(input("Sakatu y: "))
# else bigarren if-ari dagokionean
if x == 1:
    if y == 2:
        print("12")
    else:
        print("Bigarren if egituran")
print(x,y)
# else lehen if-ari dagokionean
if x == 1:
    if y == 2:
        print("12")
else:
    print("Lehenengo if egituran")
```

4.2. programa. Bi aukera daude *if* egitura habiaratuan *else* adarra kokatzeko.

5. *string* edo zerrenda hutsa, 0 zenbakia...

Gainera, *elif* ere erabili daiteke *else* adarraren ordeztuz "*else if*" adierazteko. Horrela, ondoko kodean *if* baten barruan hiru adar agertzen dira:

```
if n < 0:
    print ("negatiboa")
elif n==0:
    print ("zero")
else:
    print ("positiboa")
```

Baliokidea den beste honen idazketa sinplifikatua da:

```
if n < 0:
    print ("negatiboa")
else:
    if n==0:
        print ("zero")
    else:
        print ("positiboa")
```

Esan bezala baldintza konposatuak adieraz daitezke *not*, *and* eta *or* eragileen bitartez. Adibidez, 4.2. programaren lehen bi sententziak (bi *if*) bakar batez ordezkatu litezke (baita programaren bigarren agerpenean ere):

```
if (x == 1) and (y == 2):
```

4.2. WHILE KONTROL-EGITURA

Egitura errepikakor orokorra da *while*. Prototipoa ondokoa da:

```
while jarraitzeko-baldintza:
    aginduak
```

Jarraitzeko baldintza —espresio booleartzat hartzen dena— ebaluatu ondoren, egiazkoa (ez *None*) bada, egiturari dagokion sententzia edo sententzia-multzoa exekutatu da, eta baldintzako espresioa berriro ebaluatzen itzuliko da, begizta osatuz. Espresioaren ebaluazioa faltsua denean, *while* egitura osoaren exekuzioa bukatutzat emango da eta ondoko sententziara (tabulazioaren arabera) pasatuko da, *while* egituraren gorputza berriro exekutatu gabe.

Python programaren idazkeraren berezitasunak kontrol-egitura honetan ere azaltzen dira: tabulazioa zurrunda da eta bi puntu karakterea jarri behar da baldintza ondoren. Bi puntuen aurreko espresioa da errepikatzen *jarraitzeko baldintza*, eta bi puntuen ondoan tabulazio-maila bat barrurago azaltzen diren agindu guztiak behin eta berriro exekutatu den gorputza (*aginduak*) da.

Zenbaki baten faktoriala kalkulatzen duen 4.3. programan *while* egitura bat ikus daiteke. Bertan, egitura errepikakorren baldintza $i \leq n$ da, eta bloke bat osatzen duten bi sententziek osatzen dute gorputza. *while* egitura errepikakorra *if* egitura baten barruan dago, *else* adarrean hain zuzen. Ohartzekoa da tabulazioa nola dagoen eginda: ezker-ezkerrean hasieraketa eta *if* egitura, koxka bat eskuinerantz sententzia gehienak *while* egitura barne, eta hirugarren mailan errepikatuko diren bi sententziak. Azken sententzia (*print*) ez da errepikatuko inolaz ere, 2. mailan dagoelako.

```
# n faktoriala
n=int(input('Sakatu zenbaki bat, bere faktoriala kalkulatzeko: '))
if n <= 0:
    print ("Errorea: zenbakia positiboa behar da");
else:
    fakt = 1
    i = 2
    while i <= n:
        fakt = fakt * i
        i = i+1
    print (fakt)
```

4.3. programa. Zenbaki oso baten faktoriala kalkulatzea *while* egitura batekin.

4.3. FOR .. IN KONTROL-EGITURA

Egitura errepikakor honetan posible da zerrenda eta datu-egitura batzuetan elementu bakoitzerako agindu batzuk errepikatzea. Egitura horiei iteragarri (*iterable*) esaten zaie. Kontuan hartu behar da beste lengoaietako bektoreak eta matrizeak ere zerrendak direla (hurrengo kapituluetan azalduko da). Prototipoa ondokoa da:

```
for aldagaia in zerrenda:
    aginduak           # aldagaia erabiliko da elementua aipatzeko
```

C bezalako lengoaietan ez dagoen egitura hau oso eroso da aurreko kapitulan aipatutako egitura konplexuetan: zerrendak eta hiztegiak.

Adibide gisa, 4.4. programak zerrenda bateko elementu guztiak tratatzen ditu.

```
zerrenda = ['negatiboa', 'zero', 'positiboa']
for elem in zerrenda:
    print ("Osagaia:", elem)
```

4.4. programa. Zerrenda bateko elementu guztiak idaztea *for .. in* kontrol-egitura batekin.

Programa horren exekuzioak honako irteera hau sortuko luke:

```
>>>
Osagaia: negatiboa
Osagaia: zero
Osagaia: positiboa
```

Beste lengoaia batzuetan ohikoa da *for* kontrol-egitura kopuru batez gobernatzea. C lengoaiatz, adibidez, *for(i=1;i<n;i++)* moduko egitura oso ohikoa da. Pythonez hori egiteko *range* erabiltzea da irtenbide zuzenena. Zerrenda baten osagai guztiei edo *string* baten karaktere guztiei dagokien iterazioa kontrolatzeko *range* funtzioa erabili ohi da. Osagai kopurua, zenbaki bat alegia, da *range* funtzioaren parametroa, emaitza 0 eta balio horren arteko osoko balio guztiak izanik. *for in range()* ohiko egitura bihurtuko zaigu. 4.5. programan adibide bat dugu.

```
katea=input('Sakatu karaktere-katea: ')
kar=input('Sakatu kontatzeko karakterea: ')
luzera=len(katea)
kont = 0
for i in range(luzera):
    if katea[i] == kar:
        kont = kont+1
print(kar, 'karakterearen kopurua:', kont)
```

4.5. programa. *string* baten karaktereak kontatzea *range* funtzioa erabiliz

Eta hau da programa horren exekuzioaren adibide bat:

```
>>>
Sakatu karaktere-katea: amama etorri zen atzo
Sakatu kontatzeko karakterea: a
a karakterearen kopurua: 4
```

Edozein kasutan *range* funtzioan hainbat parametro zehatz daitezke. Adibidean bezala, bakarra bada, balioen heina zero eta zehaztutakoa ken bat izango da. Bi parametroren kasuan, berriz, hasierako balioa eta bukaerakoa zehaztuko dira, bukaerako balio hori heinetik kanpo geratuz. Eta hirugarren parametro batez inkrementatzeko (edo dekrementatzeko) balioa gehi daiteke. Ondoan adibide batzuk, dagokien azalpenarekin:

```
range(10)           # 0-9 tartea
range(1,13)         # 1-12 tartea
range(2,12,2)       # 2 eta 10 arteko zenbaki bikoitiak
range(10,0,-1)      # 10 eta 1 artean ordena beheranzkorrean
```


4.4. *BREAK* ETA *CONTINUE* AGINDUAK

break sententziak begiztaren bat-bateko amaiera eragiten du, begizta ondoko hurrengo blokerara joanez. 4.6. programan "AGUR" (letra larriz) idatzi arte oihartzuna egiten duen programa ikus daiteke.

```
print("Oihartzuna. AGUR sakatu arte ez da ondoko begizta bukatuko");
while True:
    mezu=input("Zure mezua:")
    if (mezu == "AGUR"):
        break;
    print (mezu)
print("AGUR. Bukatu da.")
```

4.6. programa. Begizta infinitu batean *break* sententziaren erabilera.

Egitura errepikakorrek habiaratuak egon daitezkeenez, hau da, egitura errepikakor baten blokean beste egitura errepikakor bat egon daitezkeenez, *break* aginduak barrurago dagoen begiztaren amaiera eragiten du.

continue sententziak, begiztaren amaiera behartu ordez, uneko iterazioa saihestea eragiten du; hau da, uneko egitura errepikakorreko gorputzean dagoen gainerako aginduak exekutatu gabe berriro jarraitzeko baldintza ebaluatzerara doa.

Bien bitartez, jarraitzeko baldintza sinpleagoa izatea lortzen da, salbuespenezko kasuak tratatzeko erabiliz *break* eta *continue*. Baina ez da komenigarria *break* eta *continue* sententziak erabiltzea, programen irakurgarritasuna zailtzen dute eta; oso salbuespenezko kasuetan bakarrik erabili. 4.6. programa honela ere idatz daiteke modu egokiagoan:

```
print("Oihartzuna. AGUR sakatu arte ez da ondoko begizta bukatuko");
mezu=input("Zure mezua:")
while (mezu != "AGUR"):
    print (mezu)
    mezu=input("Zure mezua:")
print("AGUR. Bukatu da.")
```

4.5. ADIBIDEAK

4.5.1. Enuntziatua: Hirutan handiena

Irakurri hiru zenbaki oso, eta handiena idatzi (soluzioa 4.7. programan).

```
# Hiru zenbakien artean handiena
a = int(input("Sartu lehenengo zenbakia: "))
b = int(input("Sartu bigarren zenbakia: "))
c = int(input("Sartu azken zenbakia: "))
if a > b:
    if a > c:
        print ("Handiena:", a)
    else:
        print ("Handiena:", c)
else:
    if b > c:
        print ("Handiena:", b)
    else:
        print ("Handiena:", c)
```

4.7. programa. *if* habiaratuak 3 zenbakiren arteko handiena kalkulatzeko.

Exekuzioaren adibide bat:

```
>>>
Sartu lehenengo zenbakia: 23
Sartu bigarren zenbakia: 7
Sartu azken zenbakia: 15
Handiena: 23
```

4.5.2. Enuntziatua: Zerrendakoen batura

Taula baten elementuen batura kalkulatu (hurrengo kapituluetan adibide gehiago daude ohiko datu-egiturak erabiliz) (soluzioa 4.8. programan).

```
batura = 0
taula = [2, 4, 6]
for x in taula:
    batura = batura+x
print ("Batura:", batura)
```

4.8. programa. Taula baten elementuen batura.

Exekuzioaren adibide bat:

```
>>>
Batura: 12
```

4.5.3. Enuntziatua: Atzekoz aurrera

Karaktere-kate bat irakurri teklatutik eta idatzi karaktere horiek atzekoz aurrera (soluzioa 4.9. programan).

```
katea1=input('Tekleatu karaktere-kate bat: ')
luzera = len(katea1)
katea2 = ""
for i in range(luzera):
    katea2 = katea2 + katea1[luzera-(i+1)] # i 0tik hasten delako
print('Sarrera:', katea1)
print('Irteera:', katea2)
```

4.9. programa. *string* baten alderantzizkoa.

Exekuzioaren adibide bat:

```
>>>
Tekleatu karaktere-kate bat: Elantxobe
Sarrera: Elantxobe
Irteera: eboxtnaE
```

4.5.4. Enuntziatua: Zenbat bateko zenbaki bitarrean

Irakurri zenbaki oso bat eta kalkulatu zero egoeran zenbat bit dagoen karaktere horren ASCII adierazpidean (soluzioa 4.10. programa).

Esan bezala, bit-maneiuua oso zaila gertatzen da goi-mailako lengoaietan, baina Pythonen erraza da & (*and* logikoa), | (*or* logikoa), ^ (*xor*), << (ezker-desplazamendua) eta >> (eskuin-desplazamendua) eragileei esker. Adibidean, *and* eragiketaren bidez bit bat aztertzen da, zeren *and* eragiketetan bit bakar bat 1 egoeran duen maskara batekin datu bat parekatzen dugunean, 1ari dagokion datuaren bita zeroa bada emaitza zero izango baita, eta bata bada, emaitza desberdin zero. Eragiketa 32 aldiz errepikatzen den begizta batean dago (*while* egituraz), baina bit desberdinak aztertzeko, begiztaren barruan maskarako bitak desplaza daitezke (4.10. programan egin den legez), edo datu bera desplazatu.

```
maskara = 0x00000001 # 1 zenbakia hamaseitarrez (0x aurrizkia)
kont = 0
i = 0
datu = int(input("Sakatu zenbaki bat: "))
while i < 32:
    if (datu & maskara)==0:
        kont=kont+1
    maskara = maskara << 1
    i=i+1
print (datu, "zenbakian:", kont, "bit 0 egoeran")
print (datu, "zenbakian:", 32-kont, "bit 1 egoeran")
```

4.10. programa. Bit-maneiuaren adibidea: zenbaki baten adierazpide bitarraren azterketa.

25 zenbakia bitarrean hau da: 0000 0000 0000 0000 0000 0000 0001 1001
Hori kontuan hartuta, hau da exekuzioaren adibide bat:

```
>>>  
Sakatu zenbaki bat: 25  
25 zenbakian: 29 bit 0 egoeran  
25 zenbakian: 3 bit 1 egoeran
```

4.6. PROPOSATUTAKO ARIKETAK

- 1) Osatu 4.7. programa *if-else* egiturak erabiliz hiru zenbakiak goranzko ordenan idazteko
- 2) Aldatu 4.8. programa bigarren taula batean (taula2) zenbaki metatuak lortzeko: lehen osagaia bera izango da, bigarrena lehena eta bigarrenaren batura, 3.a lehen hiruren batura...
- 3) 4.9. programan oinarrituta egin programa bat karaktere-kate baten hitzak taula batean sartzeko. Hitzak zuriunez, komaz edo puntuz bereizitako zatiak izango dira. Bukaeran taula osoa inprimatu.
- 4) Egin programa bat lehen hogeitazko zenbaki lehenak kalkulatzeko. Egin bigarren bertsio bat 100 baino txikiagoak diren zenbaki lehenak taula batean jartzeko.

5. Funtzioak, klaseak eta metodoak

5.1. SARRERA

Ohiko diren prozedura-lengoaiei modura, Pythonek azpiprogramak idazteko aukera ematen digu. Lengoiaren arabera azpiprogramei izen desberdinak ematen zaizkie: prozedura, funtzioa, errutina, azpirrutina, etabar. Izendapen horien guztien arteko diferentziak garrantzi txikiko ezaugarriak dira, eta horretan *funtzio* kontzeptuaren ezaugarri nagusia zera da: funtzioak emaitza bat itzuliko duela suposatzen da.

Funtzioen helburua, azpiprograma guztiena bezala, programen diseinua eta idazketa erraztea da; beraien bidez posible baita problema konplexu bat azpiprograma errazagotan banatzea, horietako bakoitza geroago programa errazagoekin ebazteko asmoz. Behe-mailako funtzioek eragiketa sinpleak burutzen dituzte, eta goi-mailako funtzioek behe-mailako funtzioak erabiltzen dituzte. Teknika horri *beheranzko programazio* edo *programazio modular* esaten zaio, hari esker programak ulergarriago eta sendagoak izaten dira, eta programatzaileen eraginkortasuna handitzen da.

Funtzio baten izena programa-zati bat erreferentziatzeko laburdura bat dela pentsa daiteke. Funtzioa behin definitzen da, baina sarritan deitua izan daiteke gero. Maiz exekutatzen den sententzia-multzoa funtzio gisa definitu ohi da edozein programazio-metodologia erabiltzen denean. Funtzioen erabilerak, programaren irakurgarritasuna bultzatzeaz gain, handitzen du programak idazteko eta aldatzeko erraztasuna, malgutasuna eta fidagarritasuna.

Funtzioak erabiltzea eta programazio modularren teknika ia programazioaren hasieratik datoz, baina urte batzuk geroago beste ideia ahaltsu bat azaldu zen: *objektu* kontzeptua. Objektuei orientatutako programazioa dator hortik, eta Pythonek ere barneratu du kontzeptu hori. Objektuak (datu-motak edo) *klase* gisa definitzen dira eta *modulutan* kapsulatzen dira, testuinguru bat emanez. Modulu baten barruan pakete moduan definitzen diren konstante, aldagai, objektu eta horri dagozkion funtzioen artean lotura estua sortzen da. Klase barruko funtzioei *metodo* esaten zaie.

Beraz, objektu bati dagozkion metodoak eta bestelako definizioak (bertako aldagaiak, adibidez) klase baten barruan definitzen dira. Gainera, funtzioak edozein esparrutan erabil daitezke eta behar den informazio guztia parametro bidez jakinarazten zaien bitartean, metodoak dagozkien objektuaren gainean bakarrik

aplika daitezke, eta parametroetan agertzen ez diren balio lagungarri batzuk erabil ditzakete klasean definituta badaude. Programaziorako interesgarri eta ahaltsuak diren kontzeptu batzuk balia ditzakegu objektuen bidez, herentzia eta polimorfismoa esaterako. 8. kapituluaren arituko gara horietaz.

Kapitulu honen hasieran funtzioen erabilera aztertuko dugu, eta ondoren objektuak eta beraiei dagozkien metodoak.

5.2. FUNTZIOAK

Ikus dezagun adibide bat, faktoriala kalkulatzeko funtzioa esaterako. Aurreko kapituluaren, 4.3. programan, zenbaki oso baten faktoriala kalkulatzeko programa bat azaldu da, baina dena agindu-segida bakar batean bilduta. Hori ez da aukera bakarra; zeren faktorialaren kalkulua azpiprograma edo funtzio batean programatu baitaiteke behin betirako, eta programa nagusi desberdinetatik erabili 5.1. programan azaltzen den legez.

```
# Zenbaki baten faktoriala kalkulatu
# Funtzioaren definizioa
def faktorial (n):
    fakt = 1
    i = 2
    while i <= n:
        fakt = fakt * i
        i=i+1
    return (fakt)

# Programa nagusia, funtzioa erabiltzen duena
m=int(input('Idatzi zenbaki bat, bere faktoriala kalkulatzeko: '))
if m <= 0:
    print ("Errorea: zenbaki positiboa behar da")
else:
    print(faktorial(m))
```

5.1. programa. Faktorial funtzioa.

Exekuzioaren adibide bat:

```
>>>
Idatzi zenbaki bat, bere faktoriala kalkulatzeko: 4
24
```

5.2.1. Definizioa, deia eta emaitzak

Adibide horretan bi *hitz erreserbatu* erabili dira: *def* definiziorako eta *return* emaitza itzultzeko. Beste sententzia batzuetan bezala (*if* kontrol-egituran, adibidez), definizioaren lehen lerroaren bukaeran ':' karakterea zehaztu behar izan da.

Hasieran definitu den *faktorial* funtzioa geroago erabili da programaren azken lerroan (funtzioari *deitu* egiten zaiola esaten da). Bestalde, gogoratu tabulazioek berebiziko garrantzia dutela Python programazioan, funtzioaren definizioko gorputzean sartzen diren agindu guztiak tabulazio-maila berean azaltzen dira, hasierako *def* lerroa baino maila bat barrurago.

Orain funtzioaren definizioari helduko diogu. *def* hitz erreserbatuaren ondoan funtzioaren izena eta parentesien artean argumentuak agertzen dira. Ondoko lerroetan funtzioaren gorputza osatzen duten aginduak, eta bukaeran emaitza itzultzeko *return* sententzia azaltzen da. Azken sententzia horrek bi helburu ditu: azaltzen den funtzioaren exekuzioaren amaiera behartzea, eta itzuli beharreko balioa zehaztea. Itzultzeko baliorik zehazten ez bada, *None* balioa (hitz erreserbatua, '*ez*' adieraztekoa) itzuliko da.

Aurreko programan *faktorial* da funtzioaren izena, bera da funtzioa deitzeko erabiliko den *identifikadorea*, eta *n* definizioko *argumentua*. Argumentu bat baino gehiago egonez gero, koma karakterez bereiziko dira. Itzultzen den balioa, berriz, *fakt* aldagaiarena da.

Funtzioa erabiltzen denean (*deitu* egiten zaionean), normalean esleipen baten eskuinaldean agertu ohi da deia. Dei hori ebaluatuko denean, emaitza lortuko da, eta pentsatu behar dugu deiaren emaitza dela esleituko den balioa. Hala ere, espresio baten erdian edo beste funtzio baten parametro gisa ere ager daiteke, ondoko adibideetan bezala:

```
print(faktorial(m))
print(10 * faktorial(4))
```

Oro har, pentsatu behar dugu funtzioaren deia bere emaitzaren balioaren ordeztu erabiltzen dugula programan. Balio bat esleipen batean edo beste espresio batean erabil daitekeenez, funtzio bati egiten zaion dei bat ere esleipen batean edo beste espresio batean agertu ahal izango da.

Python lengoaia interpretatua denez, funtzioaren definizioa deia baino lehen behar du interpretatzaileak, bestela errorea adieraziko du.

5.2.2. Argumentuak

Aldagaietan bezalaxe, funtzioetan eta argumentuetan Python lengoia ez da datu-mota zehazten, baina modu implizituan inferitzen dira. Horrela, *return* sententzian zehazten den balioaren arabera inferitzen da funtzioaren emaitzaren mota, eta funtzioari deitzean pasatzen zaizkion balioen motaren arabera inferitzen da argumentu bakoitzaren mota. Gogoratu Python interpretatua dela eta exekuzioan zehar egingo direla moten arteko bateragarritasun-egiaztapenak. Horrela, 5.1. programan arreta jarritz, funtzioaren emaitza *fakt* aldagaiaren balioa da, osokoa,

hasierako balioa 1 delako; eta argumentu gisa m aldagaia, osokoa ere bai, erabiltzen da. Beraz, bateragarritasunaren aldetik arazorik ez dago funtzio-dei horretan.

Bestalde, definizioan argumenturen bat zehazten bada, deian argumentuari dagokion balioa zehaztu beharko da, bestela interpretatzaileak errorea adieraziko du. Batzuetan errore hori ez gertatzea nahi dugu, horrelakoetan parametroaren balio lehenetsia definitu beharko da definizioan. Adibidez, *faktorial* funtzioa definitzean balio lehenetsia 1 izatea nahi badugu:

```
def faktorial(n=1):
```

5.2.3. Kanpoko funtzio bat erabiltzea

Aurreko adibidean muga bat antzematen da: funtzioaren definizioa eta deia fitxategi edo modulu berean daude, horrela ez da erraza izango *faktorial* funtzioa beste programa batzuetan berrerabiltzea. Muga hori gainditzeko, funtzioaren definizioa aparteko fitxategi batean kokatzen da, eta funtzioa erabili nahi duten programek funtzioa *inportatuko* dute funtzioaren definizioa duen fitxategitik.

Horrela, 5.2. programan definizioa kendu dugu eta lerro hau gehitu dugu:

```
from funtz import faktorial
```

non *from* eta *import* hitz erreserbatuak diren, eta *funtz* da funtzioaren definizioa daukan fitxategia (*funtz.py*).

```
# n faktoriala (2)
# Importatu faktorial funtzioa funtz modulutik
from funtz import faktorial

# Exekuzioa
m=int(input('Sakatu zenbaki bat, faktoriala kalkulatzeko: '))
if m <= 0:
    print ("Errorea: zenbaki positiboa behar da")
else:
    print(faktorial(m))
```

5.2. programa. *faktorial* (faktoriala) funtzioaren inportazioa.

Ikusiko duzunez, dena berdin-berdin dabil, baina funtzioari beste hainbat programatatik ere dei dakioke bertan definitu behar izan gabe.

Funtzioetan erabiltzen diren aldagaiak, argumentuak barne, *aldagai lokalak* dira. Beraz, modulu bereko bi funtzioetan identifikadore bera erabiltzen bada barruko aldagai bat erabiltzeko, n aldagaia adibidez, bi aldagai horiek diferenteak izango dira, funtzio horietako batean n aldagaian egingo diren esleipenek ez dute inolako eraginik izango beste funtzioiko n aldagaian. Gauza bera gertatzen da programa deitzailearen aldetik, horretan ere n aldagaia erabiltzen bada, hori hirugarren aldagai bat izango

da, bere balio propioa izango duena. Propietate hau funtsezkoa da programazio-lana taldekideen artean banatu ahal izateko, kide bakoitzak egin ditzake hainbat funtzio bere aldetik, beste kideen funtzioetan zer identifikadore erabili diren jakin behar izan gabe. Aldagaiak lokalak direnez, kideen artean ez dago interferentziarik programazio-lanean.

Programatzaileak kontuan hartu behar ditu bakarrik funtzioari argumentu gisa pasatuko zaizkion balioak, eta *return* bidez itzuli behar duena. Harentzat argumentuak izango duen balioa definizioko argumentuan agertzen den identifikadorea da, berau aldagai moduan erabilita.

Aldagai globalak ere erabil daitezke. Hainbat funtziok konpartitzen duten aldagaia da aldagai globala. Baina ez da gomendatzen aldagai globalen erabilera. Halakorik beharrezkoa denean hobe da objektuak erabiltzea geroago ikusiko dugun moduan (8.1. atala).

5.2.4. Funtzioak: kontzeptu aurreratuak

Parametroak, posizioaren arabera pasa beharrean (ohikoa dena, deiko lehen balioa funtzioko lehen argumenturako da, deiko bigarren balioa bigarren argumenturako, eta abar), gakoien bidez ere (*keyword arguments* ingelesez) pasa daitezke. Funtzio-deian esplizitu idazten dira argumentu bakoitzaren identifikadorea eta hartuko duen balioa. Horrela, balioak azaltzen diren ordena ez da garrantzitsua izango. Kasu horretan definizioan erabiltzen diren argumentuek eta deian erabilitako balioek bateragarriak izan behar dute.

Adibidez, 5.3. programan berreketa kalkulatzeko funtzio bat definitzen da, gako bidezko bi argumentu duena eta berretzailean (*ber*) balio lehenetsia (2) esleituta duena. Lehen deian ez da ordena betetzen eta bigarrenetan argumentu bakarra erabiltzen da.

```
def berrek (oin,ber=2):  
    return (oin**ber)  
  
print (berrek(ber=3,oin=4))  
print (berrek(oin=4))
```

5.3a. programa. Gako bidezko argumentuak.

Argumentu bakar batean argumentu-multzo bat jaso nahi denean funtzioaren definizioko argumentu listan '*' karakterea erabiltzen da argumentuaren identifikadorearen aurrean. Horrela **arg* espresioa ikusten denean, funtzio baten argumentu-listako bukaeran, zera ulertu behar da: parametro bakar horretan hainbat balio pasako direla (gehienetan egitura errepikakor batean erabiltzeko).

Aurreko bi ezaugarriak (balio-multzoa eta gako bidez) ***kwargs* egiturak barneratzen ditu. Hurrengo kapituluan azalduko diren hiztegiekin lotuta dago, argumentuan hiztegi oso bat (*hash* taula bat) txertatzen baita. Hiztegi horretako gakoak indize gisa erabil daitezke, eta 5.3b. programan dokumentatzen da haien erabilera.

```
def berrek2 (**kwargs):
    return(kwargs['oin']**kwargs['ber'])

def berrek3 (*args):
    return(args[0]**args[1])

print(berrek2(ber=3,oin=4))
print(berrek3(4,3))
```

5.3b. programa. Balio anitzeko argumentuak.

5.3. ADIBIDEAK FUNTZIOEKIN

Atal honetan funtzioak definitzeko hainbat adibide azaltzen dira. Azken adibidean funtzio horien erabilera egiten da. Funtzio guztiak *funtz.py* fitxategian daude definituta.

5.3.1. Enuntziatua: Zenbaki handiena

Balio handiena itzultzen duen funtzioa. 4.1. programan egin da, baina funtzioa erabili gabe. Ohartu kasu honetan bi argumentu daudela.

```
# Zenbaki handiena
def handiena(a,b):
    if a > b:
        return(a)
    else:
        return(b)
```

funtz moduluko *handiena* funtzioa.

5.3.2. Enuntziatua: Mezu laburrena

Jasotako bi mezuren artean laburrena selekzionatzeko funtzioa. Bigarren kapituluan aipatutako *len* funtzioa berrerabiliko da funtzio honen barruan.

```
# mezu laburrena
def mezu_labur(m1,m2):
    if (len(m1) < len(m2)):
        return(m1)
    else:
        return(m2)
```

funtz modulua: *mezu_labur* funtzioa.

5.3.3. Enuntziatua: Faktoriala funtzio errekurtsibo gisa

Faktoriala modu errekurtsiboan kalkulatzeko. Modu errekurtsiboan funtzioaren definizioan bertan erabiltzen da funtzioaren deia.

```
# n faktoriala
# Funtzio errekurtsiboaren definizioa
def fakto_errek(n):
    if n < 2:
        return (1) # kasu nabaria
    else:
        return (n * fakto_errek(n-1)) # kasu errekurtsiboa
```

funtz modulua: fakto_errek funtzio errekurtsiboa.

Ikusten denez parametroa 1 bada, emaitza ere 1 izango da, eta bestelako kasuetan bere buruari deitzen dio parametroaren balioari bat kenduta.

5.3.4. Enuntziatua: Funtzioen proba

Aurreko funtzioak programa batetik erabili nahi dira. 5.4. programan azaltzen da hori. Ohartu *from/import* sententzien erabileraz eta nola funtzioak deitzean parametroen izenak desberdinak diren (*x* eta *y*, definizioan *a* eta *b* diren bitartean *handiena* funtzioaren kasuan).

```
# Probak

# Zenbaki handiena kalkulatzeko
from funtz import handiena
x = int(input("Sakatu bi zenbakietako bat: "))
y = int(input("Sakatu bestea: "))
print("Handiena:", handiena(x,y))

# Mezu laburrena inprimatzeko
from funtz import mezu_labur
x = input("Sakatu mezu bat: ")
y = input("Sakatu beste bat: ")
print("Mezu laburrena:", mezu_labur(x,y))

# n faktoriala errekurtsiboki
from funtz import fakto_errek
m=int(input('Sakatu zenbaki bat, faktoriala kalkulatzeko: '))
if m <= 0:
    print ("Errorea: zenbaki positiboa behar da")
else:
    print(fakto_errek(m))
```

5.4. programa. Funtzioen probak.

5.4. OBJEKTUAK, KLASEAK ETA METODOAK

Esan bezala Python objektuei orientatutako lengoaia da, prozedura-lengoaiei ezaugarriak ere barneratzen dituen arren. Paradigma honetan eragiketak datu-egiturekin daude estuki lotuta klaseak osatuz. Klase batean objektuak (datu-mota bat aurreko terminologian) eta dagozkien metodoak (eragiketak edo funtzioak) definitzen dira.

Are gehiago, oinarritzko datu-mota batzuk ere objektu moduan ikus daitezke, eta aurredefinitutako metodo batzuk dituzte barneratuta. Adibidez, karaktere-kateetan ondoko metodoak daude aurredefinituta: *lower* (letra xehea) eta *upper* (letra larria) besteak beste.

Metodo bat programa batean erreferentziatu nahi denean (erabili nahi denean), objektuaren identifikadorea azaltzen da aurretik, gero puntua eta metodoaren identifikadorea, eta, azkenean parametroa(k) parentesi artean. Azter dezagun ondoko kodea:

```
st = "PYTHON Lengoaia"
print(st.lower())
```

Bi agindu horiek egikarituta, letra xehez inprimatuko da esleitutako karaktere-katea ("*python lengoaia*"). Kasu sinple honetan ez dago parametrorik, baina metodoa objektuaren balioaren gainean aplikatzen da zuzenean.

Zerrenden kasuan aurrez definituta dagoen metodoetako bat *append* da. Horren bidez, zerrenda bati elementu berri bat eranstean zaio bukaeran. Adibidez 4.6. programan agertzen den taulan (3 elementu zuen) 2 elementu eransteko haxe egin daiteke:

```
taula = [2, 4, 6]
...
taula.append(8); taula.append(10)
...
```

Hori eginda, *taula* objektuaren balioa [2, 4, 6, 8, 10] izango da.

Ikus daitekeenez, *append* metodoak parametro bat hartzen du, erantsi nahi den elementua. Beraz, *append* metodoa *taula*-ri berari aplikatzen zaio 8 parametroarekin.

Ariketa moduan proposatzen da 4.4. programa osatzea aldaketa honekin: zerrendako 'negatiboa', 'zero' eta 'positiboa' balioei 'infinutua' balioa gehitzea eta berriro idaztea zerrenda.

5.5. KLASA BERRIAK DEFINITZEN

Ohikoa denez, adibideak erabiliko ditugu kontzeptu berriak azaltzeko. *Saila* klasearekin hasiko gara, *obj_saila.py* programan kontsulta daitekeena.

```
class Saila(object):

    def __init__(self, izena=None, url=None, luzapena=None):
        self.izena = izena
        self.url = url
        self.luzapena = luzapena

    def eskura(self):
        return(self.izena + ", " + self.url + ", " + self.luzapena)

s1 = Saila("ekonomia", "www.ueu.eus/ekonomia", "23")
s2 = Saila("zuzenbidea", "www.ueu.eus/zuzenbidea", "33")
print(s1.eskura())
print(s2.eskura())
```

***obj_saila.py* programa: sail bati dagokion objektua klase berri gisa.**

Bertan *Saila* klasea sortzen da bi metodorekin: *init* metodo eraikitzailea eta *eskura* izeneko metodoa. Lehenengoan klasearen informazioa definitzen da (kasu honetan *izena*, *url* eta *luzapena*). Metodo horrekin *Saila* klaseko objektu berri bat sor daiteke, hasierako balio batzuk esleituta. Beste metodoa, *eskura* izenekoa, *Saila* klaseko objektuak barruan duen informazioa jasotzeko definitu da. Ikus daitekeenez, hainbat hitz erreserbatu erabiltzen dira: *class*, *def*, *self*, *None* eta *__init__* besteak beste. Programako azken 4 lerroetan klasearen bi metodoen erabilera erakusten da. C programazio-lengoaiaz hau *struct* egituraren bidez egin ohi da.

Ohikoagoa da erabilera beste fitxategi batean egitea funtzioetan egiten den bezala (*from/import* sententzia erabiliz). Hain zuzen ere, horixe egiten da *obj_pila.py* eta 5.5 programetan. Pilak LIFO motako ilarak (sartzen azkena irteten lehena, *last input first output*) inplementatzeko erabiltzen dira.

Lehen moduluan, *obj_pila.py* programan, *Pila* klasea definitzen da hiru metodorekin: *pila* bat sortzeko metodo eraikitzailea (*init*), *push* eta *pop*. Lehenak zerrenda hutsa sortzen du, bigarrenak elementu bat gehitzen du, eta hirugarrenak gehitutako azken elementua kentzen du eta kendutako balioa itzultzen du.

```

class Pila :

    # Hasieraketa
    def __init__(self) :
        self.osagaiak = []

    # Elementu bat gainea
    def push(self, osag) :
        self.osagaiak.append(osag)

    # Eskuratu tontorrekoa
    def pop(self) :
        return self.osagaiak.pop()

```

***obj_pila.py* programa: pila objektua klase berri gisa.**

Behin definituta, hainbat programatik erabil daiteke, 5.5. programa horren adibide bat izanik. Bertan hiru datuak pilaratzen dira eta gero azken biak berreskuratuta idatzi.

```

from obj_pila import Pila
p = Pila()

p.push(4)
p.push(-7)
p.push(5)

print(p.pop())
print(p.pop())

```

5.5. programa Pila klasearen erabilera.

Exekuzioaren adibidea:

```

>>>
5
-7

```

Klaseek oso mekanismoa ahaltsua eskaintzen dute, objektuei orientatutako programazioaren ezaugarri interesgarriak eskuratuz. Horrela, objektuak babestuta geratzen dira eta objektu horiek erabiltzeko modu bakarra metodoak dira.

Hurrengo kapituluan, aurredefinitutako klaseetako metodo erabilgarri batzuk azalduko dira.

5.6. ADIBIDEAK KLASEEKIN

5.6.1. Enuntziatua: Denbora ordu eta minututan

Denbora (ordu eta minututan) objektua definitzea bi metodo barneraturekin: inprimaketa eta batuketa⁶. *obj_denbora.py* programan dago emaitza.

```
class Denb:
    def __init__(self, ord, min):
        self.ord = ord
        self.min = min

    def inpri(self):
        print(str(self.ord) + ":" + str(self.min))

    def batu(self, t2):
        self.ord += t2.ord
        self.min += t2.min
        if self.min > 60:
            self.ord += 1
            self.min -= 60
        if self.ord > 24:
            self.ord -= 24
```

***obj_denbora.py* programa: denbora kudeatzeko klasea.**

Funtzionamendua testatzeko kode hau erabil daiteke:

```
from obj_denbora import Denb
t1 = Denb(23,59)
t2 = Denb(5,55)
t1.inpri()
t2.inpri()
t1.batu(t2)
t1.inpri()
```

Ikus daitekeenez, 23:59 eta 5:55 denborak inprimatu eta gero haien batura inprimatzen da: 5:54. Ariketa gisa kenketa gehitzea proposatzen da.

5.7. PROGRAMA NAGUSIARI PARAMETROAK PASATZEA

Batzuetan programei parametroak pasa behar zaizkie, programa ahaltsuagoa eta malguagoa izan dadin. Horretarako, programa exekutagarria martxan jartzeko komandoan izenaren ondoren parametroak idatziko dira. Eta komandoan erantsi diren parametroen balioak jasotzeko, *sys* moduluko *argv* argumentu-zerrenda erabiltzen da. Parametroak banaka atzitu nahi badira, *argv* zerrendako osagaiak

6. adibidea webgune honetatik hartu eta moldatu da: <<http://www.greenteapress.com/thinkpython/thinkCSpy/html/chap14.html>>.

indizeen bidez lortu beharko dira. Horrela, *argv[0]* programaren izena da; *argv[1]* lehen argumentua, eta abar. Argumentu kopurua *len* funtzioaren bidez lor daiteke: *len(sys.argv)*.

Adibidez, 5.6. programak programaren izena eta argumentu guztiak inprimatzen ditu komando-lerrotik oihartzuna lortuz. Ondoren argumentu kopurua ere inprimatzen du:

```
import sys
for arg in sys.argv:
    print arg
print "Argumentu kopurua:", len(sys.argv)
```

5.6. programa. Argumentuen oihartzuna.

Adibidez komando-lerroan ondokoa zehazten bada: *python3 p5-6.py a b c*

Ondokoa agertuko da irteeran:

```
p5-6.py
a
b
c
argumentu kopurua: 4
```

Batzuetan programaren izena ez zaigu interesatzen eta orduan *sys.argv[1:]* erabil daiteke. Ariketa gisa probatu aldaketa hori 5.8. programan (*for* egituran).

Programaren hasieran parametroak jasotzeko erabili ohi da *argv*. Argumentuek fitxategi baten izena edo aukera bat adierazten dute askotan. Aukeren kasuan C lengoian ohikoa da *getopt* funtzioa erabiltzea eta Pythonen horren pareko klaseak ere badaude: *getopt* eta *argparse* klaseak (<<https://docs.python.org/3/library/getopt.html>> kontsulta daiteke adibidez informazioa zabaltzeko).

5.8. PROPOSATUTAKO ARIKETAK

1) *funtz* moduluko *mezu_labur* funtzioan oinarrituta, *mezu_luze* funtzioa osatu eta probatu.

2) Faktoriala maiz erabiltzen da konbinatorian. Adibidez, *m* elementu *n*-ka konbinatzeko aukera kopurua kalkulatzeko faktoriala erabiltzen da. Honako hau da formula:

$$faktorial(m) / (faktorial(n) * faktorial(m-n))$$

5.2. programan oinarrituta, kalkulatu 8 elementu 5naka konbinatzeko aukera kopurua

3) *obj_saila.py* programan oinarrituta, egin klase berri bat liburuen katalogaziorako. Ondoko metodoak izan behar ditu: eraikitzailea, inprimaketa eta egilearen zein ISBN bidezko bilaketa.

4) *obj_denbora.py* modulua osatu denboren artean kenketa eta konparaketa bideratzeko. Bi metodo berri beharko dira horretarako: *ken* eta *handiago*.

5) *getopt* klaseaz irakurri eta egin programa bat 4 parametro erabiltzen dituen, lehena karaktere bat eta gainerako hirurak zenbakiak. Lehen parametroaren arabera hauxe egin behar da: '+' bada, zenbakiak gehitu; '-' bada, batuketa baina baturari balio negatiboa eman, eta, '*' bada, zenbakiak biderkatu. Parametroen balioak egokiak direla egiaztatuko da.

6. Ohiko datu-egiturak eta dagozkien metodoak

Programazio-lengoaia guztietan bezala Pythonen ere bertako funtzio-multzo bat dago, eta erabiltzaileak erabil ditzake nahiz eta berak ez dituen definitu. *Built-in* funtzio esaten zaie funtzio orokor horiei. Asko dira, eta ez daude objektuekin lotuta, nahiz eta datu-mota zehatzen gainean aplikatu behar diren. Zerrenda luzea da, osoa <<https://docs.python.org/3/library/functions.html>> gunean dago. Atal honetan erabilienak aipatuko ditugu. Beste batzuk hurrengo orrietan agertuko dira.

Lehenik eta behin, aipa ditzagun aurreko kapituluetan azaldu direnak: *input*, *print*, *int*, *str*, *type*, *len* eta *range* (gogoratu zertarako diren eta, beharrezkoa izanez gero, kontsultatu aurreko kapituluak). Geroago aipatuko diren *dict*, *list* eta *open* funtzioak ere asko erabiltzen dira. Aski erabiliak diren eta zenbakiekin zerikusia duten funtzio batzuk 6.1. taulan deskribatzen dira.

PROTOTIPOA	EMAITZA
<code>abs (n)</code>	Balio absolutua
<code>bin (n)</code>	Baliokide bitarra
<code>chr (n)</code>	Zenbakiari dagokion Unicode karakterea
<code>float (n)</code>	Zenbaki oso bati edo <i>string</i> ari dagokion zenbaki erreala
<code>hex (n)</code>	Baliokide hamaseitarra
<code>oct (n)</code>	Baliokide zortzitarra
<code>ord (k)</code>	Unicode karaktereari dagokion zenbakia

6.1. taula. *Built-in* funtzio garrantzitsu batzuk.

Horrez gain, orain arte aipatutako datu-moten gainean metodo anitz daude aurredefinituta, datu-mota horiek berez klaseak direlako, eta dagozkien aldagaiak objektuak. Gainera, oinarritzko klase berri garrantzitsu bat eta dagozkion metodoak ere azalduko dira: hiztegiak. Eta garrantzi txikiagoko beste klase batzuk ere: tuplak, bektoreak, karaktere-bektoreak...

6.1. OHIKO DATU-MOTEN GAINEKO METODOAK

Karaktere-kateak eta zerrendak, dagozkien metodoekin, aztertuko ditugu bereziki hemen. <<https://docs.python.org/3/library/stdtypes.html>> webgunean informazio zabalagoa lor daiteke.

6.1.1. Karaktere-kateen gaineko oinarrizko metodoak

Metodoak azaltzen sartu baino lehen jakin behar da karaktere-kateak indizeen bidez kudea daitezkeela. Horrela, zerrenden portaera imitatzen dute. Lehen karaktereari 0 indizea dagokio, bigarrenari 1... Indize negatiboak ere erabil daitezke bukaeratik abiatzeko: -1 azken karakterea da, -2 azken-aurrekoa... Adibidez, lehen erabilitako *st* katean

```
st = "PYTHON Lengoaia"
```

zuriunea identifikatzeko *st[6]* edo *st[-9]* erabil daitezke.

Indize-tarteak ere erabil daitezke *string* osoaren azpikateak adierazteko. Horrela, lehen hitza (*PYTHON*) adierazteko *st[0:6]* erabiliko genuke; eta bigarrenarako (*Lengoaia*) *st[7:15]* edo *st[-8:0]*.

Aipatutako *len* funtzioa eta *lower* eta *upper* metodoez gain, hainbat metodo erabil daitezke: *count*, *startswith*, *endswith*, *find*, *replace*, *split* eta *join*:

- *count* metodoaren bidez kate batean karaktere bat edo azpikate bat zenbatetan azaltzen den kontatu daiteke. Kontatu nahi den karakterea edo katea izango da parametroa eta kopurua emaitza.
- *startswith* eta *endswith*: lehen edo azken karaktereez galdetzeko metodoak.
- *find* metodoaren bidez kate batean ea karaktere bat edo azpikate bat dagoen bila daiteke. Bilatu nahi den karakterea edo katea izango da parametroa eta aurkitutakoaren indizea emaitza; aurkitzen ez bada bilatutakoa, emaitza -1 balioa izango da.
- *replace* metodoa *find*-en antzekoa da, baina aurkitutako azpikatea ordezkatzeko du.
- *split* metodoaren bidez kate bateko osagaiak banatzen dira eta zerrenda bat osatuko da zati horiekin. Emaitza zerrenda bat da eta parametro gisa karaktere banatzaile bat zehatz daiteke, baina zehazten ez bada, ohiko banatzaileak (zuriunea, tabulazioa eta lerro-jauzia) erabiliko dira.
- *join* aurreko metodoaren alderantzizkoa da, parametroa den zerrenda baten elementuak biltzen ditu emaitza den katean. Apur bat bitxia izan arren, oinarri gisa banatzailea erabiltzen da. Oso erabilia da emaitzak pantailartzeko.

6.1. programaren bidez metodo horien erabilera eta emaitzak azaltzen dira. Egiaztatu ohar moduan agertzen diren emaitzak exekuzioan agertzen direnak direla.

```

st = "PYTHON Lengoia"
#      012345678901234
st2 = "2,555,000"
print(st.lower())           # "python lengoia"
print(len(st))             # 15
print(st.count('a'))       # 2
print(st.find('a'))        # 12 (lehen a-ren indizea, 0tik hasita)
print(st2.replace(',', '.')) # 2.555.000
print(st.split())          # ["PYTHON", "Lengoia"] (zerrenda)
print(st2.split(','))      # ['2', '555', '000'] (zerrenda)
print(''.join(st.split())) # "PYTHON,Lengoia" (karaktere-katea)

```

6.1. programa. Oinarrizko metodoak karaktere-kateetarako.

Horrez gain, letra larria/xeha kudeatzeko metodo gehiago daude: *capitalize* (lehen karakterea larriz), *title* (hitz bakoitzeko lehen karakterea larriz) eta edukiaren nolakotasuna egiaztatzeak (*isalnum*, *isalpha*, *isdecimal*, *isdigit*, *islower*, *isnumeric*, *isprintable*, *isspace*, *istitle*, *isupper*).

Esan bezala karaktere-kateen zati bat identifika daiteke indizeekin (0tik hasita): zatiaren hasierako posizioa eta bukaerakoa zehatz daitezke makoen artean eta ':' karaktereaz banatuta. Hasiera ez bada zehazten, 0 posizioa hartzen da, eta, bukaera-posizioaren faltan, katearen bukaeraraino hartuko da. Horrela, *st[2:4]* espresioak 3. eta 4. karaktereek osatutako azpikatea adierazten du, *st[:2]* espresioak lehen bi karaktereak, eta *st[8:]* espresioak 9.etik bukaera arteko katea. Indize bakar bat erabiltzen bada, ':' karakterea zehazten ez bada, jakina, posizio horri dagokion karakterea adierazten egongo gara (azken hori 4.9. programan erabili zen).

Adibidez, 6.1. programan ondoko kodea gehituta:

```

pos=st.find('a')
print(st[:pos])

```

"PYTHON Lengo" agertuko da idatzita.

6.1.2. Zerrenden gaineko metodoak

Zerrenda datu-mota oso erabilia da eta dituen metodoei esker oso mota ahaltsua da. Zerrendak esleipen batez hasiera daitezke (adib. *aukerak* = ["bai", "ez"]) edo *list* funtzioaren bidez (*aukerak* = *list*("bai", "ez")). *Zerrenda* hutsa ere defini daiteke (adib. *aukerak* = []).

Zerrendako elementu zehatzak eta azpizerrendak ere erabil daitezke makoak eta ':' karakterea erabiliz. Horrela, *zerrenda[0]* lehen osagaia da eta *zerrenda[:2]* espresioaren bidez lehen bi osagaiak adierazten dira.

Bestalde, zerrendako elementu txikiena eta handiena hautatzeko *min()* eta *max()* funtzioak erabil daitezke; *len()* osagai kopurua lortzeko; eta '+' eragilea bi zerrenda kateatzeko.

Baldintzazko egituretan erabiliak izateko diseinatuta daude *in* eta *not in* eragiketak. Adibidez:

```
if hitza in hitzerrenda:
```

Zerrenden gainerako metodoei dagokienez, aipatutako *append* metodoa da interesgarrienetako bat, zerrendari osagaiak, banan banan, gehitzea ahalbidetzen baitu. Horrez gain, hainbat metodo erabil daitezke:

- *extend* metodoaren bidez zerrendak bil daitezke. *append* funtzioaren modukoa da, baina osagai bat gehitu beharrez beste zerrendako osagai guztiak gehitzen ditu⁷.
- *insert* metodoaren bidez zerrenda batean elementu bat txerta daiteke zehaztutako posizio batean. Beraz, elementua eta posizioa dira argumentuak.
- *count* metodoaren bidez zerrenda batean balio bat zenbatetan azaltzen den konta daiteke.
- *sort* metodoa zerrendako osagaiak ordenatzeko erabiltzen da. Ez du parametririk. Ordenazio konplexuetarako aurredefinitutako *sorted* funtzioa erabil daiteke.
- *reverse* metodoak zerrendaren osagaiak alderantzizko ordenan jartzen ditu. Ez du parametririk.
- *index* metodoak osagaien bilaketa egiten du, aurkituz gero indizea itzultzen du (0tik hasita).
- *pop* metodoa erabil daiteke elementu bat zerrendatik kentzeko

6.2. programan metodo horien erabileraren adibideak agertzen dira. Zerrenda osoa zuzenean ezin denez idatzi, edo egitura errepikakor baten bidez inprimatuko da, edo, adibidean egiten den bezala tarteak erabiliz (*[:]* espresioak tarte osoa adierazten du).

```
z1 = ["bai", "ez"]
z2 = [ 1, 2, 3, 5, 7, 11, 13 ]
z3 = [ -1, -3 ]
print(min(z3), max(z2)) # -3 13 Balio minimoa z3 eta maximoa z2
z1.append("zuri")      # osagaia erantsi z1-i
```

7. *z.extend(z2)* eta *z=z+z2* baliokideak dira.

```

print(z1)          # ['bai', 'ez', 'zuri']
print(len(z1))     # 3      Osagai kopurua
print(z2)          # [1, 2, 3, 5, 7, 11, 13]
z2.extend(z3)      # z3 erantsi z2-ri. Baliokidea: z2 = z2 + z3
print(z2)          # [1, 2, 3, 5, 7, 11, 13, -1, -3]
print(len(z2))     # 9      Osagai kopurua
print(z2.count(-1)) # 1      -1 osagaia behin azaltzen da z2 listan
z2.insert(0,0)     # txertatu 0 hasieran
print(z2)          # [0, 1, 2, 3, 5, 7, 11, 13, -1, -3]
z2.sort()          # Ordenatu
print(z2[:])       # [-3, -1, 0, 1, 2, 3, 5, 7, 11, 13]
z2.reverse()       # Alderantzizko ordena
print(z2[:])       # [13, 11, 7, 5, 3, 2, 1, 0, -1, -3]
print(*z2)         # 13 11 7 5 3 2 1 0 -1 -3 (inprimatzeko beste modu bat)
print(z2.index(7)) # 2      7 osagaiaren indizea z2 listan

```

6.2. programa. Oinarrizko metodoak zerrendetarako.

6.1.3. Zerrenden eta karaktere-kateen arteko bihurketa

Kasu batzuetan komenigarria da bi mota hauen artean bihurketak egitea. Adibidez, zerrendaren elementuak inprimatzeko karaktere-kate bihur ditzakegu; bestalde zerrenden gaineko metodo edo ezaugarri batzuk aplikatzeko kateak zerrenda bihur daitezke.

Lehen helbururako *join()* metodoa erabiliko da objektu gisa banatzailea zehaztuz (zuriunea, koma edo karaktere hutsa adib.). Bigarrenarako, berriz, *string*aren elementuak zuriuneen bidez berezituta badaude, *split()* metodoa erabiltzea besterik ez da egin behar zerrenda bat osatzeko. Banatzailea beste bat bada, orduan banatzailea zehaztu beharko da parametro gisa metodo horri deitzean. Adibidez datu-baseetatik inportatutako fitxategietan tabulazioa erabili ohi da banatzaile gisa, beraz, ondoko kodea erabiliko genuke eremuak zerrenda gisa eskuratzeko:

```
zeremuak = lerro.split('\t')
```

Banatzailearik ez baldin badago; alegia, karaktere bakoitza osagai bat izatea nahi badugu, orduan *list()* funtzio eraikitzailea erabil daiteke, katea parametro gisa pasaz.

Horrela, 4.9. programa (karaktere-kate bat irakurri teklatutik eta idatzi alderantziz) sinplifika daiteke 6.3. programan dagoen kodea erabiliz.

```

katea1 = input('Sakatu karaktere-katea: ')
z = list(katea1)          # katearen karaktereekin zerrenda bat osatu
z.reverse()              # atzekoz aurrera jarri zerrendako osagaiak
katea2 = ''.join(z)       # listako osagaiak kateatu banatzailerik gabe
print('Sarrera:', katea1)
print('Irteera:', katea2)  # beste aukera bat: katea1[::-1]

```

6.3. programa. *string* bat atzekoz aurrera, sinplifikatua.

Exekuzioaren adibidea:

```

>>>Sakatu karaktere-katea: abcdefg
Sarrera: abcdefg
Irteera: gfedcba

```

6.1.4. *List comprehension*

Zerrenda baten hasieraketa (edo esleipena) oso modu trinkoan egin daiteke *comprehension* izeneko aukeraz. Ondoko adibidean ikus daiteke nola hasieratu zerrenda balio kontsekutiboekin:

```
z = [ i for i in range(6)]
```

Eraitza honako hau litzateke: $[0, 1, 2, 3, 4, 5]$.

Ikusten denez, *for* egitura erabiltzen da esleipenaren eskuin aldera eta ':' karakterea ez da erabiltzen.

if ere aplika daiteke modu berean, horrela aurreko zerrendan balio bikoitiak bakarrik gordetzeko hau egin daiteke:

```
z2 = [ i for i in range(6) if (i%2==0)]
```

Eraitza honako hau litzateke: $[0, 2, 4]$.

Oso ahaltsua da ezaugarri hau eta ohituz gero kodea trinkotzeko lagungarria izango zaizu.

6.1.5. *Espresio erregularrak*

*String*ekin eta zerrendekin lotuta dagoen eta oso interesgarria den klase bat aipa daiteke hemen: espresio erregularrena. Testuetan bilaketak egiteko oso tresna ahaltsuak dira espresio erregularrak. Horien erabilera bideratzeko Pythonek eskaintzen duen baliabidea *re* modulua da (re: *regular expression*-en laburpena da). Ohikoa denez, eta Unixeko *grep* programaren antzera, eragile anitz eskaintzen ditu: '.' (edozein karaktere), '^' hasiera, '\$' bukaera, '*' (*kleene star*, 0 edo gehiago), '+' (bat edo gehiago), etab. Espresio erregularrak erabiltzeko orduan *match*(*patroia*, *st*)

prototipoa duen metodoa da oinarria. Patroia adierazteko r baten ondoan datorren *string* bat zehaztu behar da⁸. Horrela, ondoko kodearekin:

```
import re
bokal_segida = r"[aeiou]+"
m1 = re.findall(bokal_segida, "egitea eta saiatzea")
m2 = re.findall(r"\\(\\w+\\)", "(1) Udako Euskal Unib. (UEU), EH")
print(m1, m2)
```

Lehen bilaketan bokal-segidak (bat edo gehiago jarraian) esleitzen zaizkio $m1$ zerrendari. *findall* metodoari bigarrenaz deitzean, berriz, parentesien artean dauden hitzak bilatzen dira⁹ eta $m2$ zerrendari esleitu. Dagokion emaitza $['(1)', '(UEU)']$ izango da. Agerpen bakarra bilatu nahi bada, *search* metodoa erabiliko da *findall* ordez.

Hurrengo kapituluetan azalduko ditugu beste ezaugarri batzuk, erabili ahala. Informazio gehiagorako: <<https://docs.python.org/3/library/re.html>>.

6.1.6. Bektoreak eta matrizeak

Zerrenden bidez kudeatzen dira *Pythonen* beste lengoaietan taula, bektore, array edo taula izenarekin definitzen diren datu-egiturak. Baina zer gertatzen da matrizeekin edo dimentsio anitzeko bektoreekin?

Matrizeak zerrenden zerrendak dira, hau da, zerrenda bat non osagaiak zerrendak diren. Osagaiak adierazteko bi indize erabiltzen dira (makoen artean).

6.4. programan matrizeen batuketa kalkulatzeko funtzioa eta deia dugu.

```
def mat_batu(A, B, C):
    for i in range(len(A)):
        for j in range(len(A[0])):
            C[i][j]=A[i][j]+B[i][j]

X = [[1,2],[3,4],[5,6],[7,8]]
Y = [[11,12],[13,14],[15,16],[17,18]]
Z = [[0,0],[0,0],[0,0],[0,0]]
# Z = [[0]*2]*4
mat_batu(X,Y,Z)
print(Z)
```

6.4. programa. Matrizeen erabilera: batuketa.

8. Ez da derrigorrez beti r aurritzia jartzea, baina gomendatzen da bestelako ihes-karakterekin gatazkarik egon ez dadin, \backslash sinboloarekin hasitakoekin esaterako ($\backslash n$, $\backslash t...$). r aurritziaeren ordez R ere erabil daiteke.

9. $\backslash w$ espresioak identifikadoreetako ohiko karaktereak adierazten ditu. Horren baliokidea: $[a-zA-Z0-9_]$

Emaita honako hau litzateke:

```
>>>
[[12, 14], [16, 18], [20, 22], [24, 26]]
```

Adibidean ikus daitekeenez, *len* funtzioa erabiltzen da errenkada zein zutabe kopuruak kalkulatzeko. Zeroz hasieratzea, berriz, nekagarria izan ez dadin * eragilea erabil daiteke. Adibidean bertan ikus daitekeenez, $Z = [[0]*2]*4$ espresioan 0 balioa, [0], 2 aldiz errepikatzen da zerrenda batean (2 elementu errenkadan), eta zerrenda hori 4 aldiz (4 errenkada) errepikatzen da matrizea osatuz.

Comprehension ezaugarria erabiliz ere hasieraketa ahaltsuak egin daitezke. Honako kode honen bitartez:

```
m = [[x, y] for x in [1,2,3,8,9] for y in [9,3,1,4] if x == y]
```

bi tauletan dauden zenbakien bikoteak lortzen dira matrize batean, kasu honetarako:

```
[[1, 1], [3, 3], [9, 9]]
```

Dena den, kalkulu matematiko konplexuak programatzeko asmoa izanez gero *numpy* hedapena erabiltzea gomendatzen da, dimentsio anitzeko bektoreen erabilera asko errazten du-eta.

6.1.7. Tuplak eta heina (*range*)

Zerrendekin batera tuplak (*tuple*) eta heinak (*range*) dira Pythoneko sekuentziak. Zerrenda motari dagozkion ezaugarri asko dituzte (funtzioak, + eta *in* eragileak, indizeen bitartez erabiltzeko aukera...). Pythonen nomenklaturan *iterable*-ak direla esaten da (*string*ak eta zerrendak bezala), *for* .. *in* egituraren parte har dezaketelako.

Tuplak zerrenden moduan definitzen dira, baina makoen artean definitu beharrean parentesi artean definitzen dira (osagaiak komen bidez banatuta bietan). Tupletan zerrendetan ez bezala:

- osagaiak aldaezinak dira (konstanteak)
- ez dute metodorik onartzen (zerrenden ohiko funtzioak eta eragileak bai)
- atzipen azkarragoa bideratzen dute
- datu heterogeneoak barneratzeko pentsatuta daude

Ondoko adibideko tuplaren balioak urteko hilen egun kopuruak dira. Balio konstanteak direnez, interesgarria izan daiteke tupla bat definitzea.

```
tup = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
```

Baina kontuz, otsailari dagokion balioa 29ra aldatu nahi badugu derrigorrezkoa da zerrenda erabiltzea, bestela `tup[1]=29` esleipena egikaritzean errore bat sortuko da.

Bestalde *range*-ren oinarritzko erabilera 4. kapituluan azaldu da (ikus 4.5. programa): egitura errepikakorretan erabiltzea *for .. in* egituran. *range()* funtzioaren bidez *range* motako sekuentzia bat sortzen da, aldaezina. Parametro gisa hiru aukera daude:

- parametro bat: balio kopurua (n bada 0 eta n-1 izango dira sekuentziaren hasierako eta bukaerako balioak)
- bi parametro: hasierako eta bukaerako balioak
- hiru parametro: hasiera, bukaera eta gehikuntza

Adibidez, *range(2, 100, 2)* 2 eta 98 arteko balio bikoitien multzoa adierazten du, biak barne.

6.1.8. Bestelakoak

Mota hauekin lotutako beste mota bat aipatu behar da: byte-taulak. Javaz ohikoak diren egitura hauek zerrendak balira bezala kudeatzen dira, baina osagaiak definitzean bi digitu hamaseitar (edo balio hamartar bat 0 eta 255 artean) erabili ohi dira. Adibidez, ondoko definizioa egin eta gero:

```
btaula = bytearray([0x01, 0x02, 0x04, 0x08])
```

btaula[i] bitartez osagaiak aipa daitezke eta *btaula* bitartez identifikadoreaz sekuentzia. *bytes* mota antzekoa da.

Horrez gain, *memoryview* klasea dugu memoriaren edukiak kudeatzeko, baina klase horren erabilera sistema eragilearekin lotua denez liburuaren esparrutik urrun samar geratzen da hori.

6.2. HIZTEGIAK

Hainbat lengoaiatan eta testuingurutan *hash taula* edo *elkartze-taula* izena erabiltzen den arren, Pythoneko terminologian egitura horiei *hiztegi* esaten zaie. Funtsean taulak edo zerrendak dira, baina osagaiak erreferentziatzeko gako bat erabiltzen da, ez zenbaki bat. Beraz, indize bat erabiliko da, makoen artean, baina indizea bilaketa-gako bat izango da.

Adibide batekin hasiko gara: benetako hiztegitxo elebidun bat egiteko, euskara-inglese hiztegi bat esaterako, Pythoneko hiztegi-egitura bat sortu nahi dugu, horrela euskarazko hitz bat emanda haren ingelesezko ordaina lortu ahal izateko. Adibidez 'bai' hitzaren ordain gisa 'yes' lortu nahi da. Hiztegiei esker Python lengoaiari hori

erraz lortzen da: *hiztegia* egitura ondo osatuta baldin badago, ondoko bi espresioetako edozeinek lortuko dute helburua:

```
hiztegia['bai']      # konstante bat erabilita
hiztegia[hitza_eu]   # hitza_eu aldagaiaren balioa 'bai' bada
```

Orain falta zaiguna zera da: nola definitzen da datu-egitura hori modu egokian? Erraza da, *dict()* *built-in* funtzioa erabilita hiztegi bat sortzen da, 6.5. programan ikus daitezkeenez.

```
hiztegia = dict()      # hiztegia = {}    (baliokidea)
hiztegia['bai'] = 'yes'
hiztegia['ez'] = 'no'
hitza_eu = input("Hitza euskaraz: ")
print ("Hitza ingelesez: ", hiztegia[hitza_eu])
```

6.5. programa. Oinarrizko hiztegi baten erabilera.

Hiztegiak zuzenean hasiera daitezke beste bi modutan ere:

- giltzen artean gako/balioa bikoteak zehazten dira, tartean ":" karakterea jarritz eta bikotetik bikotera pasatzeko "," banatzailea erabiliz.
- *dict()* funtzio eraikitzailea deitzean (kontuz, gakoak kateak direla suposatzen da, eta kasu honetan ez da komatxorik erabili behar berauek mugatzeko)

Horrela, aurreko hiztegia hasieratzeko, ondoko bi espresioetako bat erabil daiteke (lehen hiru sententziak ordeztuz):

```
hiztegia = {'bai':'yes', 'ez':'no'}
hiztegia = dict(bai='yes', ez='no')
```

Hiztegien gainean hainbat funtzio eta eragile erabil daitezke, osagai (bikote) kopurua itzultzen duen *len()* funtzioa adibidez. Ohikoa da *for .. in* egitura aplikatzea ere (*iterable*-ak dira), osagai guztiak inprimatzeko adibidez (6.6. programa):

```
hiztegia = {'bai':'yes', 'ez':'no'}
print(len(hiztegia))
for h_eu in hiztegia:
    print(h_eu, hiztegia[h_eu])
```

6.6. programa. Hiztegia inprimatzen.

6.2.1. Hiztegien gaineko metodoak

Aurrekoa oso ahaltsua izan arren, hainbat metodo ere erabil daitezke hiztegiengan. 6.3. taulan daude erabilgarrienak.

PROTOTIPOA	EMAITZA
<code>clear()</code>	Hiztegia deuseztatzea
<code>copy()</code>	Hiztegia bikoiztea
<code>keys()</code>	Gakoen zerrenda
<code>values()</code>	Balioen zerrenda
<code>items()</code>	Bikoteen zerrenda (<i>for</i> zerrendetan erabiltzeko)
<code>update(hizt2)</code>	Bi hiztegien bildura

6.3. taula. Hiztegien gaineko funtzioak eta eragileak.

Adibidez, 6.6. programaren azken zatia *items* metodoa erabiliz ere idatz daiteke:

```
for h_eu, b_eu in hiztegia.items():
    print(h_eu, b_eu)
```

Bestalde, *enumerate()* *built-in* funtzioaren bitartez zerrenda batetik hiztegi bat lor daiteke. Gakoak zenbakizkoak izango dira. Horrela, hurrengo espresioaren bidez:

```
dict(enumerate(['bai', 'ez', 'zuri'], 1))
```

Ondoko hiztegia lortzen da:

```
{1: 'bai', 2: 'ez', 3: 'zuri'}
```

Bigarren parametroa zehazten ez bada, 0tik hasiko dira gakoak.

Gainera, gakoak eta balioak zerrenda banatan badaude, *zip* funtzioaz biltzen dira bikotetan. Aurreko hiztegiaren hasieraketa honela ere egin daiteke:

```
eu=['bai','ez']
en=['yes','no']
hiztegia=dict(zip(eu,en))
```

Hiztegi klasea oso erabilgarria da, baina oro har datuak fitxategietatik edo saretik kargatu beharko direnez, benetako etekina ateratzeko aukera hurrengo kapituluan izango dugu, datuak kargatzeko aukerak aurkeztu ondoren.

Badaude hiztegiekin oso antz handia duten beste bi klase: *set* (multzoak) eta *frozenset* (multzo aldaezinak) klaseak. Klase bi horietan gako bidezko bilaketak ere egin daitezke, baina gakoek ez dute balio bat esleituta.

6.3. ERROREAK ETA SALBUESPENEN KUDEAKETA

Pythonen erroreen tratamendua salbuespenen bidez egin ohi da. Horretarako programazioan *try* eta *except* sententziak erabiliko dira, *try* adarrean egin beharrekoa zehazten da, eta *except* adarrean erroreen tratamenduari dagokion kodea da.

Adibide batekin hasiko gara berriro ere. Zenbakien arteko eragiketetan "zati zero" eragiketa salbuespena da, tratatzea komeni da. Salbuespen hori sistemak sortzen du, *ZeroDivisionError* izena duena. 6.7a. programan tratamenduaren adibide bat ikus daiteke.

```
a = int(input("Sakatu zatikizuna: "))
b = int(input("Sakatu zatitzailea: "))
try:
    print(a/b)
except ZeroDivisionError:
    print("Errorea: zati 0")
```

6.7a. programa. Salbuespenen tratamendua.

Aurdefinitutako salbuespenak asko dira, 6.2. taulan erabilienetako batzuk agertzen dira:

KODEA	ESANAHIA
Exception	Orokorra, edozein salbuespen barneratzen duena
ArithmeticError	Edozein salbuespen aritmetiko (zati 0, gainezkatzea...)
OSError	Sistema eragilearen deiekin lotutako salbuespena, fitxategia aurkitzen ez denean adibidez
SystemError	Sistemaren kernelak sortzen duen barne-errorea
TypeError	Mota bateragarritasunarekin arazoak daudenean
IOError	Sarrera edo irteerarako errorea. Hurrengo kapituluan sakonduko da
MemoryError	Memoriaren gaineko errorea
KeyboardInterrupt	Etena teklatutik (<i>Ctrl</i> teklen bidez sortzen dira)

6.2. taula. Ohiko salbuespenak.

Taulan agertzen diren salbuespenak nahiko orokorrak direnez, eta batzuetan zehatzagoa izatea komeni denez, kode zehatzagoak ere eskaintzen ditu Pythonek:

- Aritmetikoetarako: *OverflowError* (gainezkatzea), *ZeroDivisionError* (zati zero), *FloatingPointError* (koma higikorreko errorea).
- Sistemetakoak: *BrokenPipeError* (arazoak *pipe* batean), *ConnectionError* (arazoak Internet edo sareko konexioan), *FileNotFoundError* (fitxategia ez da aurkitzen), *PermissionError* (arazoak baimenekin), *NotADirectoryError* (katalogo bat bilatu eta ez aurkitu)...

Salbuespen posibleak asko dira eta <<https://docs.python.org/3/library/exceptions.html#IndexError>> gunean zehaztasunak aurki daitezke.

Salbuespenarekin batera askotan zenbait informazio gehigarri ere jasotzen da, eta horrelakoetan *except* adarrean *as* sententzia bat gehitu behar da aldagai batekin, informazio gehigarria jasoko duena. Horrela, sistemaren errore bat jaso eta inprimatzeko ondoko kodea erabil daiteke:

```
except OSError as akatsa:
    print("Sistemaren errorea:", akatsa)
```

Edozein kasutan, goian aipatutakoak sistemak definitutako salbuespenak dira, baina posible da salbuespen berriak sortzea (eta komenigarria software-proiektu handietan zein gertaeretara zuzendutakoetan), erroreak mekanismo estandarraren bitartez tratatu ahal izateko. Horretarako *raise* sententzia erabiliko da. Adibidez, osoko negatiboen faktoriala ez dago definituta, eta 5.2. programan egiten zena baino dotoreagoa da ondokoa:

```
# n faktoriala (3)
from funtz import faktorial

m=int(input('Sakatu zenbaki bat, faktoriala kalkulatzeko: '))
if m <= 0:
    raise NameError('FaktoNeg')
try:
    print(faktorial(m))
except NameError:
    print("Errorea")
    raise
```

6.7b. programa. *Faktorial* funtzioa salbuespenekin definituta.

NameError erabilita salbuespena sortzen da eta interpretatzaileak abisua emango du. Exekuzioaren adibidea:

```
Sakatu zenbaki bat, faktoriala kalkulatzeko: -3
Traceback (most recent call last):
  File "/home/PYTHON_UEU/code/kap6/p6-7b_Faktorial_salbuespenekin.py", line 6, in
<module>
    raise NameError('FaktoNeg')
NameError: FaktoNeg
>>>
```

Tratamendu berezi bat egin nahi bada, aplikazio konplexuetan ohikoa dena, kontua konplexuagoa da (kontsultatu <<https://docs.python.org/3/tutorial/errors.html#user-defined-exceptions>>).

6.4. ADIBIDEAK

6.4.1. Enuntziatua: *Palindromoak detektatu*

6.3. programan oinarrituta, egin programa bat hitz-zerrendako hitz palindromoak lortzeko. Palindromoak diren hitzetan atzekoz aurrerako irakurketa hitz berarena da.

```
def atzekoz_aurrera(h):
    z = list(h)
    z.reverse()
    atzekoz = ''.join(z) # osagaiak kateatu banatzailerik gabe
    return atzekoz

hitzak = ["bai", "ez", "inoiz", "eta", "ama", "aita", "amama", "ate"]

for hitz in hitzak:
    ztih = atzekoz_aurrera(hitz)
    if ztih == hitz:
        print(hitz, ztih)
```

6.8. programa. Palindromoak detektatzen.

Exekuzioaren adibidea:

```
>>>
ama ama
amama amama
```

6.3. programaren kodea erabiliz funtzio bat definitu da hitz bat emanda haren karaktereak atzekoz aurrera jarrita beste hitz bat lortzeko, eta funtzio hori erabiliz kodea oso sinplea da: zerrendako hitz gutiak banan-banan tratatu, bakoitzari buelta eman eta hitz berarekin konparatu egiten da; biak berdinak badira palindromoa da.

6.4.2. Enuntziatua: *Bokal gehieneko hitzak*

Espresio erregularren atala kontsultatuta, egin programa bat hitz-zerrenda baten barruan bokal gehien dauka(te)n hitza(k) inprimatzeko.

```
import re

hitzak = ["bai", "ez", "berehalakoa", "leihoetako", "lehoietako", "saiatzeta"]
bokal = r"[aeiou]"
nmax = 0 # Kopuru maximoa metatzeko
zluze = [] # Bilatutako hitzen zerrenda

for hitz in hitzak:
    zatiak=re.findall(bokal, hitz)
    n=len(zatiak)
    if n>nmax:
        nmax = n # Aurrekoek baino gehiago
        zluze = [] # Aurrekoek ez dute balio
        zluze.append(hitz)
    elif n==nmax:
        zluze.append(hitz)
print("Max:", nmax, zluze)
```

6.9. programa. Bokak gehien duten hitzak.

Exekuzioaren emaitza:

```
>>>
Max: 6 ['berehalakoa', 'leihoetako', 'lehoietako']
```

re.findall metodoarekin bokalak banan-banan lortzen dira zatiak izeneko zerrenda batean, eta *len* funtzioaren bidez kontatzen dira zerrendako elementuak

Ariketa gisa zera proposatzen da: bokal kopurua kontatu beharrean, bokal-segida luzeena kontuan hartzea.

6.4.3. Enuntziatua: Matrize-biderketa

Matrizeen batuketara egiten zuen programan oinarrituta (6.4. programa), egin matrizeen arteko biderketa kalkulatzeko programa.

```
def mat_bider(A, B):
    C = [[0,0,0],[0,0,0],[0,0,0],[0,0,0]]    # dim: 4*3
    for i in range(len(A)):                  # Aren errenkadak (len(A)=4)
        for j in range(len(B[0])):          # Bren zutabeak (len(B[0])=3)
            for k in range(len(B)):         # Bren errenkadak (len(B)=2)
                C[i][j] += A[i][k] * B[k][j] # Kontuz, metaketa (+= eragilea)
    return C

X = [[1,2],[3,4],[5,6],[7,8]]              # dim: 4*2
Y = [[0,1,2],[3,4,5]]                     # dim: 2*3
print(mat_bider(X,Y))
```

6.10. programa. Matrizeen arteko biderketa.

Emaitza: [[6, 9, 12], [12, 19, 26], [18, 29, 40], [24, 39, 54]]

6.4.4. Enuntziatua: Hitza atzekoz aurrera jarrita ere badago listan

Hiztegi bat erabiliz aldatu 6.8. programa, mota honetako hitzak ere lortzeko: hitza dago listan eta atzekoz aurrera jarrita lortzen den hitza ere bai. Adibidez, *ate* hitzari buelta emanda *eta* lortzen da, eta biak daude zerrendan.

6.11. programan hiztegi bat erabiltzen da jakiteko hitz bat erabiltzen den (bat balioa esleitzen da kasu horretan) edo ez. Ideia hau zabal daiteke hitzen maiztasuna kalkulatzeko, ikusi geroago proposatuko diren ariketak.

```
def atzekoz_aurrera(h):
    z = list(h)
    z.reverse()
    atzekoz = ''.join(z) # osagaiak kateatu banatzailerik gabe
    return atzekoz
```

```

hitzak = ["bai", "ez", "noiz", "eta", "ama", "zion", "amama", "ate"]

hiztegi = dict()

for hitz in hitzak:
    hiztegi[hitz]=1          # hitza dagoela adierazteko

for hitz in hitzak:
    ztih = atzekoz_aurrera(hitz)
    try:
        if hiztegi[ztih]==1:
            print(hitz, ztih)
    except:
        None

```

6.11. programa. Hitza eta bere alderantzizkoa zerrendan. Hiztegi bidez.

Errorea tratatu da, hiztegian ez dagoen hitz bat kontsultatzean errorea gertatzen baita. Hori saihestu daiteke 6.11b. programan erabili den *defaultdict* funtzioa erabiliz hiztegia definitzeko.

Egia esan, hiztegirik erabili gabe eta listaren gainean *in* eragilea erabilia, kodea askoz sinpleagoa izan zitekeen (6.11c. programa):

```

for hitz in hitzak:
    ztih = atzekoz_aurrera(hitz)
    if ztih in hitzak:
        print(hitz, ztih)

```

6.5. PROPOSATUTAKO ARIKETAK

- 1) 10 zenbaki dituen zerrenda bat osatu teklatutik irakurritz, gero ordenatu eta alderantzizko ordenan inprimatu.
- 2) Espresio erregularrak erabiliz, egin programa bat testu bateko bokal-segidarik luzeena bilatzeko.
- 3) Osatu hiztegi bat hilen euskarazko izenekin eta haien egun kopuruarekin. Gero, egin funtzio bat hau lortzeko: hil baten izena eta eguna emanda, urteko zein eguna den (1-366 arteko balioa) itzuli beharko du. Hainbat proba egin ondo dabilela ziurtatzeko.
- 4) Hiztegi bat erabiliz, kalkulatu zerrenda batean dauden hitzen maiztasuna.

Adibidez, zerrenda honako hau bada: *["da", "du", "dute", "da", "dute", "dira", "da", "da", "dute"]*

orduan irteerak honako hau izan beharko du: *"da":4; "dute":3; "du":1; "dira":1.*

- 5) Hiztegi bat erabiliz, egin programa bat kontatzeko zenbat aldiz agertzen den karaktere bakoitza *string* batean. Adibidez, *"lengoaia"* katean emaitzak hau izan behar du: *a:2, e:1, g:1, i:1, l:1, n:1, o:1.*

7. Sarrera/Irteera. Internet

Aurreko kapituluetan erabili ditugun *input* eta *print* funtzioek osatzen dute sarrera/irteera egiteko oinarri-oinarrizko bidea, baina programa konplexuagoak egin ahala ahalmen handiagoa behar dugu Sarrera/Irteerari (S/I) begira: mezuetan formatu aberatsagoak erabiltzea, diskoko fitxategiekin lan egitea, zuzeneko atzipena eta datu-baseak erabiltzea, Interneteko datuak prozesatzea...

Beraz, kapitulu honetan horri helduko diogu ordena honetan: mezuei formatua ematea, fitxategien kudeaketa eta atzipena, eta Interneteko datuen maneia.

7.1. MEZUEN FORMATUA

Programek idatzi behar dituzten mezuetan informazioa era egituratuago eta ikusgarriagoan erakutsi ahal izateko bi bide nagusi daude: % sinboloan oinarritzen den *print* funtzioa (modu zaharra), eta *format* metodoa. Bigarrena da lehenesten dena, baina lehenengoa da bertsio zaharragoetatik datorrena (eta beste zenbait lengoaietatik datozen programatzaileei errazen egiten zaiena).

7.1.1. Modu zaharra

Beste lengoaietako *printf* funtzioaren antzekoa da modu hau. Lehen eremuan konstanteak eta aldagaien zein espresioen formatuak zehazten dira, gero % karakterea eta azkenik aldagaiak zein espresioak parentesi artean.

7.1. taulan ikus daitekeenez, formatua adierazteko % karakterea eta letra bat zehazten da. Letraren aurretik zenbaki bat gehi daiteke luzera adierazteko, eta zenbaki errealeen kasuan bi zenbaki puntu hamartarrez bereizita.

ADIBIDEA	ESANAHIA
%s	<i>string</i>
%d	zenbaki hamartarra
%f	zenbaki erreala (puntu dezimala agertuko da)
%b	formatu bitarra
%X	formatu hamaseitarra
%E	Formatu esponentziala
%4d	zenbaki hamartarra 4 digiturekin
%.2f	zenbaki erreala, bi zifra puntu ondoren
%5.1f	zenbaki erreala, bost zifra puntu aurretik eta bat puntu ondoren

7.1. taula. *print* funtzioan formatuak adierazteko hainbat kode.

Modu hau erabiltzen duen eta 2.3. programaren aldaera den kodea agertzen da 7.1. programan.

```
PI = 3.14159
# irakurketa eta bihurketa
r = float(input("sakatu erradioaren neurria: "))
perimetroa = 2 * PI * r
azalera = PI * r * r
# idazketa
formatua = "%d erradioari dagokion azalera %.2f da eta perimetroa %.2f."
print (formatua % (r, azalera, perimetroa))
```

7.1a. programa. Inprimaketa formatudunaren adibidea.

7.1.2. *format* metodoa

Edozein karaktere-kateren gainean aplikatzen da, baina S/Irekin oso lotuta dago.

Adibide batekin hasiko gara, 2.2. programaren bilakaera dena:

```
izena = input("Sartu zure izena: ")          # sarrera
print("Kaixo {}".format(izena))             # irteera format erabilita
```

2.2. programarekin konparatuta, kasu honetan giltzak erabili dira *string*ean eta aldagaiaren izena *format* metodoaren parametro gisa. Adibide honetarako aldrebesa ematen badu ere, *format* metodoa erabiltzea oso ahaltsua da aldagai anitzen balioa jaso behar duten mezuetarako. Adibidez 2.3. programaren aldaera 7.1b. programan agertzen da, idazketa lerro bakar batera ekarriz eta formatua dotoretuz.

```

PI = 3.14159
# irakurketa eta bihurketa
r = float(input("sakatu erradioaren neurria: "))
perimetroa = 2 * PI * r
azalera = PI * r * r
# idazketa
mezu = "{} erradioari dagokion azalera {} da eta perimetroa {}."
print (mezu.format(r, azalera, perimetroa))

```

7.1b. programa. Inprimaketa formatudunaren adibidea.

Kasu honetan, sententziak laburtu eta irakurgarritasuna handitze aldera, mezu izeneko katea definitu da tarteko aldagai gisa.

Beraz, formatua adierazten duen karaktere-kate baten gainean aplikatzen da *format* metodoa, katean agertzen diren giltzak ordezkatzeko bere parametroen balioekin (emandako ordena berean).

Dena den, oraindik hobekuntzak egin daitezke, adibidez balio errealeen zifra kopurua mugatzea. Programa probatu baldin baduzu, azalera edo perimetroa dezimal askorekin agertzen dira (erradioa 6 denean 113.09723999999999 agertzen da adibidez). Hori mugatzeko, bi dezimalera adibidez, giltzen artean *:.2f* zehatz daiteke, azken aurreko lerroa honela geratuko litzakeela:

```

mezu = " {:.2f} erradioari dagokion azalera {:.2f} da eta perimetroa {:.2f}."

```

Kasu horretan *:.2f* espresioak adierazten du zenbaki errealari bi zifra dezimal dagozkiola. Bi puntu karakterea formatua adierazteko eranstean da, eta formatuaren atal asko zehatz daitezke: doiketa (ezkerrean, eskuinean, erdian), zeinua eta dezimalak... 7.2. taulan ikus daitezke bihurketa-karaktere ohikoenak. Kodea beti ':' karakterearen ondoan agertuko da eta hainbat kasutan zenbakiak (aurreko programan 2 adib.) edo konstanteak (% portzentajeak adierazteko esaterako) txerta daitezke.

KODEA	ESANAHIA
s	<i>string</i> (ezer ez jartzearen baliokidea)
d	zenbaki hamartarra (zenbaki osoetarako)
n	zenbaki errealak (puntu dezimala agertuko da)
b	formatu bitarra
X	formatu hamaseitarra
<	doituta ezkerrean
>	doituta eskuinean
^	doituta erdian
,	zenbakietan milakoen mugak txertatzeko

7.2. taula. Formatuak adierazteko hainbat kode.

7.2. programan beste adibide bat azter daiteke, 1 eta 15 zenbakien arteko balioak hamartarrez, bitarrez eta hamaseitarrez inprimatzen duen programa.

```
mezu="zenb: {:2d} bitarrez: {:5b} hamaseitarrez {:2X}"
for n in range(1,16):
    print(mezu.format(n, n, n))
```

7.2. programa. Inprimaketa formatudunaren bigarren adibidea.

<<https://docs.python.org/3/library/string.html>> webgunean formateatzearen inguruko informazio osatuagoa aurki daiteke.

locals funtzioa¹⁰ ere erabil daiteke formatua emateko. Hiztegietarako pentsatuta dago, baina aldagaien izenak ere hiztegi moduan erregistratzen direnez, aldagaien izenak giltzen artean ere jar daitezke. 7.3. programan 7.1. programaren aldaera bat agertzen da. Ohar zaitez '\ ' karakterearen erabileraz ere, *stringa* hurrengo lerroan jarraitzen duela adierazteko erabiltzen baita.

```
PI = 3.14159
# Irakurketa eta bihurteta
r = float(input("sakatu erradioaren neurria: "))
perimetroa = 2 * PI * r
azalera = PI * r * r
# Idazketa
mezu = "{r:.2f} erradioari dagokion azalera {azalera:.2f} \
da eta perimetroa {perimetroa:.2f}."
print (mezu.format(**locals()))
```

7.3. programa. Inprimaketa formatudunaren *locals* funtzioa erabiliz.

Agian haxe intuitiboagoa da, baina gustukoen duzuna erabil dezakezu.

7.2. FITXATEGIAK

Fitxategiak erabiltzeko hainbat funtzio eta metodo erabiliko dira. Ohi bezala, adibide simple batekin hasiko gara, 7.4. programarekin, hain zuzen ere.

```
# programaren fitxategia inprimatzea (lerroz lerro)
for lerro in open('p7-4.py', 'r').readlines():
    print(lerro)
```

7.4. programa. Fitxategi baten irakurketa lerroz lerro.

open funtzioa erabili da eta itzultzen duen objektuaren gainean *readlines* metodoa¹¹. Oso kode trinkoa eta irakurterraza lortzen da. *open* funtzioak itzultzen duen objektuaren gainean (*file*), geroago aipatuko diren metodo guztiak izango dira aplikagarri.

10. ***locals()* erabili behar da.

11. Ez da derrigorrezkoa *readlines* metodoa erabiltzea, baina lerroz lerro irakurri nahi badugu, erabiliko da.

Aurreko soluzioak eragozpen txiki bat du, fitxategia itxi gabe geratzen dela. Ez da larria, baina fitxategiak ixtea komenigarria da, baliabideak ondo kudeatzeko. Kudeaketa ona bideratzeko *with .. as* egitura sortu zuten (fitxategietarako eta beste egituretarako). Haren bitartez objektu bat sortzen da, kudeaketa errazteaz gain automatikoki itxiko dena. Haren erabilera 7.4b. programan ikus daiteke.

```
with open('p7-4b_fitxategia_lerroka.py','r') as fitx:
    for lerro in fitx:
        print(lerro)
```

7.4b. programa. Fitxategi baten irakurketa lerroz lerro.

io izeneko moduluan daude fitxategiekin lotutako funtzioa eta metodoak. Modulu honek oinarritzko funtzionalitate guztiak eskaintzen ditu, fitxategiak *Unix* munduan ohikoa den datu-fluxu bihurtuta *file* objektuaren bitartez. Irakurtzen edo idazten den karaktere-kate bat bezala ikusten baita sarrera/irteerako edozein objektu.

Beste lengoaietan bezala, fitxategi bat erabiltzeko fitxategi hori ireki egin behar da, dagokion kontrol-informazioa memorian karga dadin eta sistema eragilearen sarrera/irteerako oinarritzko funtzioek kontsulta dezaten. Kontrol-informazio hori *file* motako egitura batean metatzen da fitxategia irekitzen denean, bere gainean sarrera/irteerako metodoak aplikatu ahal izateko.

Beraz, objektua eskuratzeko *open* funtzioa erabiliko da. Prototipoa ondokoa da:

```
f = open(fitxategia, modua, encoding="utf-8")
```

Kodeketa (*encoding*) hautazkoa da, baina testu-fitxategien kasuan garrantzitsua da ondo zehaztea. Bestalde, beste parametro batzuk ere zehatz daitezke: *errors* (erroreen maneiturako), *newline* (lerro-bukaeraren maneiturako) eta *closefd* (itxiera kontrolatzeko).

Fitxategien artean bi mota nagusi bereizten dira: testukoak eta bitarrak. Bigarren motakoetan b letra adieraziko da moduaren barruan. Fitxategiak bitarrak definituz gero, sistema eragileak ez du aipatutako byteen eta lerroen interpretazio berezirik egingo. Bitak irakurri edo idatziko dira azaltzen diren eran. Fitxategi bitarrak testuak ez diren datuak gordetzeko eta fitxategiaren edukia dagoen bezala mantentzeko erabiliko dira. Motaren arabera, metodoen semantika apur bat alda daiteke edo metodo berriak agertu, baina ezaugarri nagusiak komunak dira eta horiek izango dira hemen azalduko direnak. Metodoaren *line* edo *lines* agertzen bada testu-fitxategietan aplikatzeko direla ulertu behar da.

Irekitzeko ohiko moduak 7.2. taulan jasota daude.

KODEA	ESANAHIA
r	Testu-fitxategi bat irakurtzeko
w	Testu-fitxategi batean idazteko
a	Testu-fitxategi batean idazteko baina gehiketak eginez
+	Testu-fitxategi bat eguneratzeko (irakurri zein idatzi)
b	Formatu bitarra (aurrekoekin konbina daiteke). rb, wb, ab...

7.2. taula. Irekitzeko moduak.

Normalean, eraginkortasunari begira, fitxategiak memorian islatzen dira tarte batez (*buffering*), baina *open* funtzioan *buffering=0* aukeraz galaraz daiteke hori.

S/I estandarra (teklatura eta pantaila normalean, komando-lerrotik birbideratzeko aukera ahaztu gabe) erabiltzeko *sys.stdin*, *sys.stdout* eta *sys.stderr* fluxuak erabil daitezke (*sys* modulua inportatuz) irekitzeko beharrik gabe. 7.4. programan oinarrituta oihartzuna lortzeko programa idaztea erraza da, beraz (ikus 7.5. programa).

```
import sys
for lerro in sys.stdin:
    print(lerro)
```

7.5. programa. Oihartzun funtzioa.

7.6. programan sarrera/irteerako funtzioak eta metodoak erabiltzen dira, UNIXen erabiltzen den *cat* izeneko programa inplementatzeko. Programa horren bidez parametro gisa zehazten diren fitxategiak idazten dira bata bestearen atzean irteera estandarrean.

```
import sys

def erantsi_fitxat (fitx):
    for sarr in fitx.read():          # for sarr in fitx*: (baliokidea)
        sys.stdout.writelines(sarr)

for arg in sys.argv[1:]:
    f = open(arg, 'r')
    erantsi_fitxat (f)
```

7.6. programa. Hainbat fitxategi kateatzea.

Proba ezazu parametro gisa bi programen izenak zehaztuta.

Fitxategietako datuak bata bestearen ondoren jarraian tratatu ohi dira, atzipen sekuentziala eginez, eta horixe da orain arte azaldu dena. Datuak bata bestearen ondoren tratatu nahi ez direnean, zuzeneko atzipena beharko da, *seek* eta *tell* metodoetan oinarrituta burutzen dena. Geroxeago azalduko ditugu.

*. *read()* metodoa da metodo lehenetsia, metodorik zehazten ez bada.

7.2.1. Ohiko metodoak

Ondoren *io* moduluaren ohiko metodoak azalduko dira, metodo bakoitzaren prototipoa eta deskribapena zehaztuz:

```
close()
flush()
```

objektuari dagokion fitxategia itxi eta objektua desagertzen da. Aurretik memorian zerbait balego oraindik idatzi gabe, fitxategian idatziko du itxi baino lehenago. *flush()* metodoak azken zeregin hori du.

```
read(tam=-1)
readall()
```

read metodoak zehaztutako tamaina irakurtzen du, edo fitxategi osoa tamaina zehazten ez bada (-1). Irakurritakoa itzultzen du emaitza gisa. *readall* metodoak horixe egiten du, fitxategi osoa irakurri eta memoriara ekarri. Kontuz fitxategi handiekin!

```
readlines(kop=-1)
readline()
```

readlines metodoak aipatutako lerro kopurua (*kop*) irakurtzen du, edo fitxategi osoa, kopurua zehazten ez bada (-1). *readline* metodoak lerro bat irakurtzen du. Testu-fitxategietarako oso egokiak dira. Irakurritakoa itzultzen du karaktere-kate formatuan. Kontuan hartu behar da *read/readline* metodoak erabiliz fitxategien posizioa eguneratzen dela eta fitxategia berriro hasieratik atzitzeko berriro ireki beharko dela.

```
write(buf)
writelines(lerroak)
```

buf parametroak adierazten duen katea fitxategiaren barneko bufferrean idazten du *write* metodoak, geroxeago idatz dadin. *writelines* metodoak antzera lan egiten du, baina lerroz lerro lan eginez.

C lengoaiako *scanf* funtzioaren baliokiderik ez dago; horren ordez *read* eta aurreko kapituluaren azalduetako *split* metodoa konbinatzea proposatzen da. Idazketa formatudunetan, berriz, aurretik azalduetako *format* metodoa eta *write* edo *writelines* konbinatzea da ohikoena.

Fitxategiaren bukaera (EOF) detektatzea ohikoa da fitxategi sekuentzialak erabiltzean. Pythonen ez da behar izango, fitxategi osoak eskuratzeko metodoak erabiltzen badira (*readall*, *getlines*, edo *read* metodoak parametrorik gabe), baina zatika edo lerroka irakurtzean (*readline* edo *read* luzera batekin) detektatu beharko

da. Buffer hutsa jasotzen denean, fitxategi-bukaera antzeman dela suposatuko da halakoetan, beraz, ez dago metodo berezirik horretarako.

7.7. programan testu-editoreek duten bilaketa-laguntza nola programatzen den ikus daiteke. Lerroka irakurtzen da eta *find* metodoaren bidez (edo *in* eragilearen bidez) aztertzen da ea karaktere-katea lerroan dagoen ala ez (gogoratu *find* metodoak posizioa itzultzen duela edo -1 aurkitzen ez badu)

```
import sys

fitxat = sys.argv[1]
st = sys.argv[2]
n = 0
f = open(fitxat, 'r')
for lerro in f.readlines():
    if st in lerro:          # edo pos=lerro.find(st) / if pos>0:
        print(lerro)
        n = n+1
print ("agerpen kop.:", n)
```

7.7. programa. *string* baten bilaketa fitxategi batean.

7.2.2. Erroreen maneia

Sarrera/irteerako metodo bakoitzak balio berezi bat itzuliko du, errore bat gertatzen bada. Erroreen tratamendurako *io* objektuaren hainbat eremu eta salbuespen erabil daitezke. S/Irako orokorra *IOError* izeneko salbuespena erabiltzea da *try* eta *except* sententziekin batera (ikus aurreko kapituluan salbuespenei dagokien atala erabilera gogoratzeko). Fitxategiekin lotutako erroreak *OSError* multzoaren barruan sartzen dira, adibidez, fitxategiak irekitzean sortzen direnak (ohikoa da fitxategia ez aurkitzea, izena ez delako ondo teklatu edo beste katalogo batean dagoelako)

Horrela, 7.6. programan erroreen tratamendua egin nahi bada, badaezpada fitxategien izenak ondo zehaztu diren ziurtatzeko, honela aldatu beharko lirateke azken lerroak:

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
        erantsi_fitxat (f)
    except OSError:
        print('Sistemaren errorea (open)', arg)
    except IOError:
        print('S/Iko errorea', arg)
```

Probatu programa hori parametro gisa existitzen ez den fitxategi bat pasatuz.

7.2.3. Zuzeneko atzipena

Adibideetan ikusi dugunez, oinarritzko sarrera/irteerako metodoek (*read*, *write*, *readline*, *writeline*...) fitxategiko datuen atzipen sekuentziala bideratzen dute, eta horretarako fitxategia irekitzean hartzen duen uneko posizioa erabili eta eguneratzen dute (0 posizioa gehienetan), "a" moduaren kasuan (*open* funtzioan) izan ezik, non fitxategiaren luzeraren balioa (bukaerako posizioa) hartuko baitu uneko posizio gisa, eransketak egin ahal izateko.

Zuzeneko atzipena burutzeko aipatutako funtzioak *seek* metodoarekin konbinatu beharko dira S/Iko beste metodoak (*tell* metodoarekin ere konbinatuta kasu batzuetan). *seek* metodoak posizio finko edo erlatibo batean kokatzeko ahalmena eskaintzen du, posizioa eta jatorria zehaztu beharreko parametroak izanik.

```
seek(pos, jatorri)
```

Hiru modu edo *jatorri* bereizten dira, bakoitzak posizioa/desplazamendua aplikatzeko puntu desberdina aukeratzen duela, 0 fitxategiaren hasiera aukeratzen du (modu lehenetsia denez, nahi izanez gero, ez da zehaztu behar), 1 uneko posizioa eta 2 fitxategiaren bukaera.

tell metodoarekin, berriz, uneko posizioa ezagut daiteke, emaitza gisa itzultzen baitu. Hori interesgarria izaten da zuzeneko atzipena atzipen sekuentzialarekin konbinatu nahi denean, posizioarekiko atzipen erlatiboak egiteko adibidez, nahiz eta *seek* metodoaren 1 jatorriarekin horixe ere egin daitekeen.

7.8. programan *zuzen_atzi* funtzioa definitzen da, zeinak, parametro baten arabera, posizio jakin batean dagoen luzera finkoko eremu bat irakurri edo idazten duen.

```
import sys

def zuzen_atzi(fitx, pos, buffer, eragik):
    try:
        fitx.seek(pos,0)
    except IOError:
        print("seek-a ez da ondo joan")
        exit

    try:
        if(eragik=="R"):
            return(fitx.read(1))
        else:
            fitx.write(buffer)
    except IOError:
        print("zuzeneko atzipena ez da ondo joan")
        exit
```

```

try:
    buf = ""
    fitxat = sys.argv[1]
    f = open(fitxat, 'r+')
    buf = zuzen_atzi(f, 2, buf, "R")
    print(buf)
    zuzen_atzi(f, 8, buf, "W")
    f.seek(0)
    print(f.read())
except OSError:
    print("Sistemaren errorea (open)", fitxat)
except IOError:
    print("S/Iko errorea", fitxat)

```

7.8. programa. Zuzeneko atzipenaren erabilera.

Programa horretan *zuzen_atzi* funtzioa definitzen da, fitxategi batean kokatu, *pos* posizioan, eta idatzi edo irakurri (karaktere bat) egiten da *eragik* parametroaren arabera. Erroreen tratamendua ere egiten da: programa nagusian funtzioa erabiltzen da 2 posizioan dagoen karakterea (3.a) karakterea irakurtzeko eta 8 posizioan kopiaitzeko. Bukaeran fitxategia irakurri (hasieran kokatu eta gero) eta inprimatzen du.

Adibidez, fitxategiaren edukia "*abcdefghijkl*" bada, emaitza hauxe izango da: "*abcde**fg**h**c**ijkl*", 2 posizioko 'c' karakterea 8 posizioan idatzi baita, zegoen 'i' karakterearen ordeztu. Proba ezazu programa fitxategi labur batez (*proba_seek* fitxategia dago programekin batera) eta posizioak aldatuz programan.

7.2.4. Katalogoen maneia

Fitxategiak katalogoetan antolatzen dira eta batzuetan beharrezkoa da katalogoen maneia egitea programatik.

Adibidez, 7.7. programaren kateatzea katalogo bateko fitxategi guztiekin egin nahi bada, eta katalogoa parametro gisa pasatzen bazaio, programa nagusiaren kodea honela eratutako litzateke: horretarako hainbat funtzio eta metodo interesgarri daude.

```

from os import listdir
from os.path import isfile, join
for izenf in listdir(sys.argv[1]):
    print("FITX:", sys.argv[1], izenf)
    if isfile(izenf):
        f = open(izenf, 'r')
        erantsi_fitxat(f)

```

Ikusten denez, *listdir* eta *isfile* metodoak inportatzen dira *os* eta *os.path* moduluetatik. Lehenengoak egiten du nahi genuena, baina azpikatalogoak eta bestelako fitxategi motak (*socketak*, *pipeak*...) egon daitezkeenez, *isfile* funtzioa erabiltzen da datu-fitxategiak hautatzeko (kontuz probatzean fitxategi bitarrekin)

Metodo asko dago modulu horietan, horietako batzuk 7.3. taulan islatu dira. Dena den, lagungarria izan daiteke jakitea modulu horietan komando eta sistema-dei ia guztiei dagozkien metodoak daudela.

PROTOTIPOA	HELBURUA
os MODULUA	
<code>listdir(path='.')</code>	Katalogoaren osagaiak (parametrorik gabe uneko katalogokoak)
<code>getcwd()</code>	Uneko katalogoa
<code>lchmod(path, modua)</code>	Baimenak aldatzea modua-rekin
<code>remove(path, *, dir_fd=None)</code>	Fitxategiak ezabatzea
os.path MODULUA	
<code>abspath(path)</code>	Fitxategi edo katalogo baten bide absolutua itzultzen du
<code>isfile(path)</code>	Datu-fitxategia den ala ez
<code>isdir(path)</code>	Katalogoa den ala ez
<code>getsize(path)</code>	Dagokion tamaina
<code>getmtime(path)</code>	Azkenez aldatu zeneko denbora

7.3. taula. Katalogoekin lan egiteko hainbat funtzio eta metodo.

Informazio gehiagorako kontsultatu <<https://docs.python.org/3/library/os.html>> eta <<https://docs.python.org/3/library/filesys.html>>.

7.2.5. Bestelakoak

Proiektu handietan fitxategiez gain datu-baseak ere kudeatzen dira. Horrelakoetan datu-baseen hornitzaileek eskaintzen dituzten moduluak, dagozkien metodoekin, erabili beharko dira. Adibidez, MySQL datu-baseak erabiltzeko *MySQLdb* modulua dago eta hainbat metodo (*connect*, *execute*...). Informazioa zabaltzeko 8.5. atalera jo dezakezu.

7.3. INTERNET

Interneteko konexioak eta Webeko edukiak modu errazean kudeatzea da Pythonen ezaugarri garrantzitsuenetako bat. Horretan sakontzeko liburuan kapitulu oso bat dagoen arren, oinarritzkoena aurreratuko dugu hemen, lehenbailehen ariketa interesgarriak egin ahal izateko.

Ohi bezala adibide batekin hasiko gara, 7.9. programa. Bertan *urllib.request* modulua *urlopen* metodoa erabiltzen da web orri bat (parametro gisa eman behar da URLa) irakurtzeko.

```
import sys
from urllib.request import urlopen
weborri = urlopen(sys.argv[1])
for lerro in weborri:
    lerro = lerro.decode('utf-8') # kodeketa-arazoengatik
    print(lerro)
```

7.9. programa. Web orri baten irakurketa.

Probatzeko testu asko eta HTML kode gutxi duen gune bat zehaztea komeni da, adibidez <<https://www.berria.eus>>.

Klase eta metodo asko eta asko daude Interneten aritzeko, baina Webeko edukiak modu errazean erabiltzeko duen ahalmena frogatzeko bigarren adibide bat azalduko dugu: *smtplib* modulua mezu bat bidaltzeko. *SMTP*, *sendmail* eta *quit* metodoak erabiltzen dira 7.10. programan. Programa martxan jarri ahal izateko posta-zerbitzari bat eta bezero bat behar dira lokalean (Linuxen *postfix* eta *mail* instalatzea gomendatzen da). Ohiko posta-zerbitzariekin konektatzea konplexuagoa da (kontuak, baimenak eta konfigurazioa direla-eta).

```
import smtplib
zerb = smtplib.SMTP('localhost')
mezu = """
    To: ixa@localhost
    From: ixa@localhost
    Subject: Proba
    Kaixo. """
zerb.sendmail('ixa@localhost', 'ixa@localhost', mezu)
zerb.quit()
```

7.10. programa. Mezu bat bidaltzea.

7.4. ADIBIDEAK

7.4.1. Enuntziatua: *Hitzen maiztasuna*

Fitxategi edo web orri baten gainean hitzen maiztasuna kalkulatzeko. 10 aldiz edo gehiago agertzen diren hitzak inprimatuko dira. Bi parametro pasatzen dira: aukera ("fitx"/"URL") eta izena (fitxategiarena edo URL osoa). Hiztegi bat erabiltzen da maiztasunak metatzeko eta bertan gakoak hitzak izango dira, eta lotutako balioa agerpen kopurua. 7.11. programan dago kodea. Gogoratu aurreko adibideetan hainbat kode-zati berrerabiltzen direla.

```
# klaseen inportazioa
import re
import sys
from urllib.request import urlopen

#maiz_inkre: hitzen taula batetik (hak) maiztasunak eguneratu m hiztegian
def maizt_inkre(hak,m):
    for h in hak:
        if(h in maiz):
            maiz[h] += 1      # inkrem
        else:
            maiz[h] = 1      # 1 balioa
```

```

# argumentuak eta
aukera = sys.argv[1]
iz = sys.argv[2]
maiz = dict()

if (aukera=="URL"):
    weborri = urlopen(iz)
    for lerro in weborri:
        lerro = lerro.decode('utf-8')
        hitzak = re.findall(r"\w+", lerro)
        maizt_inkre(hitzak,maiz)
else:
    f = open(iz,'r')
    for lerro in f.readlines():
        hitzak = re.findall(r"\w+", lerro)
        maizt_inkre(hitzak,maiz)

# inprimatu 10 aldiz edo gehiago daudenak
for j in maiz:
    if (maiz[j]>10):
        print(j, maiz[j])

```

7.11. programa. Fitxategi edo web orri baten gainean hitzen maiztasuna kalkulatzea.

7.4.2. Enuntziatua: Zuhaitz bitarra idatzi

7.12. programan zuhaitz bitar bat definitzeko erabiltzen den egitura eta metodoak azaltzen dira. Funtzioak arbola ezkerretik eskuinera idazten du, algoritmo errekursiboa erabiliz.

```

class ZuhBitar():

# zuhaitza erro-adabegiaren (nodoaren) bidez ezagutzen da
def __init__(self,id):
    self.ezk = None
    self.esk = None
    self.id = id

#gehikuntzak goian. Behean egiteko egitura errepikakorra beharko da
def gehitu_esk(self,adabegi):
    if self.esk == None:
        self.esk = ZuhBitar(adabegi)
    else:
        zuh = ZuhBitar(adabegi)
        zuh.esk = self.esk
        self.esk = zuh
def gehitu_ezk(self,adabegi):
    if self.ezk == None:
        self.ezk = ZuhBitar(adabegi)
    else:
        zuh = ZuhBitar(adabegi)
        self.ezk = zuh
        zuh.ezk = self.ezk

```

```
# inprimaketa errekurtsiboa ezkerretik eskuinera
def print_zuh2(self):
    if self.ezk != None:
        self.ezk.print_zuh2()
    print(self.id)
    if self.esk != None:
        self.esk.print_zuh2()
```

7.12. programa. Egitura errekurtsibo baten adibidea.

7.5. PROPOSATUTAKO ARIKETAK

1) *more* programaren egitekoa burutzen duen programa egin behar duzu. Testu-fitxategi bat parametro gisa jasota, pantailaz pantaila idazten joango dira lerroak. Pantaila osatu eta gero, teklatutik irakurri, eta, 'q' bada, amaitu egingo da, bestela, beste edozein karaktere jasota, hurrengo pantaila idaztera pasako da. Sinplifikatzearen, pantaila baten edukia 40 lerrotan ezarriko da. Bigarren bertsio batean gehitu errorearen tratamendua.

2) *bilatu_eta_ordezkatu* izeneko programa bat egin behar duzu, zeinek fitxategi baten gainean karaktere-kate bat bilatu eta beste batez ordezkatuko duen. Hiru parametro emango zaizkio programari, beraz, fitxategiaren izena, bilatzeko katea eta ordezkatzekoa. Bigarren bertsio bat egin, kate bat bilatu beharrean espresio erregular bat bilatuko duena.

3) 7.9. programa osatu web orrian beltzez (dagokion HTML etiketa bilatu beharko da) idatzitako zatiak inprimatzeko.

4) Hitzen maiztasunak kalkulatzeko 7.11. programa ondo dago, baina hainbat hobekuntza egin nahi ditugu:

- Maiztasunak kalkulatzeko hiztegia klase moduan definitu behar da.
- Erroreen tratamendua egin behar da: parametro desagokiak, fitxategia ez egotea, URL-a ez aurkitzea...
- Maiztasunak ematean ordena garrantzitsua da, handienetik txikienera.

Aldaketak egin, hori guztia lortzeko

5) Aldatu aurreko programa, hitzen ez baizik eta n-gramen maiztasuna kalkula dezan. Parametro moduan pasako da n zenbakia. n-gramak n karakterezko multzo teilakatuak dira; adibidez "*Python lengoaia*" katean 3-gramak hauek dira: " Py", "Pyt", "yth", "tho", "hon", "on ", "n l", " le"...

8. Klase berriak eta ohikoenak

Kapitulu honetan aurreko kapituluetan landu ezin izan diren hainbat kontzeptu eta aukera jartzen dira praktikan, azalpen laburrekin lagunduta. Bi multzotan banatu dira landuko diren gaiak: klaseetan sakontzea eta eskuragarri dauden modulu interesgarrienak (matematikoak, grafikoak, CSV, XML eta JSON formatuei dagozkienak, datu-baseetakoak...) azaltzea. Gaiak oso zabalak dira eta hemen sarrera bat baino ez dugu egingo, beti bezala adibideekin lagunduta. Hurrengo kapituluetan gai horietako batzuetan sakontzen da, baina kapitulu bakoitzari dagokion gaiaren barruan.

8.1. KLASEETAN SAKONTZEN

Hasieran esan den bezala, liburu hau ez dago zuzenduta algoritmika ikastera edota objektuei orientatutako programazio eta metodologiak ikastera. Neurri batean edo bestean algoritmika jakinda, objektuei orientatutakoa barne, hori aplikatzea Python lengoaian da liburu honen helburua. Atal honi dagokionez, objektuei orientatutako metodologian ohikoak diren kontzeptuak eta praktikak azaletik aztertuko dira erabilera bultzatuz.

5. kapitulan objektuei orientatutako programazioaren (OOP) inguruko ezaugarri nagusiak eta oinarrizko adibideak azaldu dira. Kapitulu honetan objektuen erabilera aurreratuari ekingo diogu, herentzia eta poliformismoa esaterako. Gai honetan gehiago sakontzeko webgune hau da gomendagarriena: <<https://docs.python.org/3/tutorial/classes.html>>.

8.1.1. Aldagaien eta funtzioen esparrua

Aldagaien eta funtzioen izenak Pythonen izen-sistema osatzen dute eta izen horien esparrua askotarikoa izan daiteke:

- Lokala, funtzioarena edo metodoarena, bertan definitzen bada.
- Klaseko globala, klasearen metodoetatik kanpo definitzen denean. Klaseetako metodoak ere esparru honetakoak dira.
- Globala, programa osoarena, klaseetatik eta funtzioetatik kanpo definitzen bada.
- Erabatekoa, *built-in* funtzioen kasua (*builtins* izeneko moduluan daude horiek).

Interpretatzaileak aipatutako ordenan bilatzen ditu izen bat ebatzi behar duenean, lokaletik globalera alegia, eta, beraz, anbiguotasuna dagoenean lokalena hautatuko da.

Dena den, kontuan hartu behar da funtzioak defini daitezkeela beste funtzioen esparruan, beraz, halakoetan, lokal-global kontzeptua lausoa da, inkremental bihurtuz; eta esparru estuagoa/zabalagoa esatea da egokiena.

Horrez gain, aipatutako esparruak alda daitezke bi gako erabilita: *global* eta *nonlocal*. Lehenarekin, hainbat lengoaiatan bezala, esparru lokaletik globalera zabaltzen da esparrua, bertako aldagaiaren izena modulu osora zabalduz. Bestalde, aldagai globalak edo esparru zabalagokoak beste esparru batean erabiltzen direnean ezin dira aldatu, irakurtzeko bakarrik erabil daitezke; aldatu ahal izateko *nonlocal* gisa zehaztu beharko dira (bestela lokal bat sortuko da balio berria barneratzeko).

```
def esparru_test():
    def f_lokal():
        mezu = "mezu lokala"
    def f_ezlokal():
        nonlocal mezu
        mezu = "mezu ezlokala"
    def f_global():
        global mezu
        mezu = "mezu globala"
    mezu = "mezu hasiera"
    f_lokal()
    print("Esleipen lokalaren ondoren:", mezu)
    f_ezlokal()
    print("Esleipen ez-lokalaren ondoren:", mezu)
    f_global()
    print("Esleipen globalaren ondoren:", mezu)

    esparru_test()
    print("Esparru globalean:", mezu)
```

8.1. programa. Izenen esparrua.

8.1. programan esparruen inguruko adibide bat azter daiteke. Exekuzioaren irteera hauxe da:

```
Esleipen lokalaren ondoren: mezu hasiera
Esleipen ez-lokalaren ondoren: mezu ezlokala
Esleipen globalaren ondoren: mezu ezlokala
Esparru globalean: mezu globala
```

Hasiera batean portaera harrigarria egiten bada ere, urratsez urrats uler daiteke, beti kontuan hartuta *mezu* izeneko aldagai bat baino gehiago dagoela: bat lokala *f_lokal* funtzioaren barruan; beste bat *f_global* funtzioaren barruan definituta, baina globala dena; eta hirugarrena *esparru_test* funtzio osorako, hasierako balioa hartzen duena. Hori kontuan hartuta

- *f_lokal* exekutatzean bertako aldagai lokalari aldatzen zaio balioa, baina inprimatzean programa esparru horretatik kanpo dago, beraz, funtzio osoko esparrua duen aldagaia inprimatzen da.
- Bigarren eta hirugarren inprimaketetan aldagai bera inprimatzen da, baina *nonlocal* deklarazioa dela-eta *f_ezlokal* funtzioan aldatu zaio balioa.
- *f_global* funtzioan bertako aldagaia aldatzen da, baina globala denez, azken inprimaketan, *esparru_test* funtziotik kanpo, haren balioa inprimatzen da.

8.1.2. Herentzia eta polimorfismoa

OOPn erabat normala da dauden objektuetatik abiatuz beste objektu berri bat definitzea. Herentzia hitza erabiltzen da hori adierazteko, klase batek bere ezaugarriak beste klase batetik heredatzen dituela pentsa dezakegulako. Bide horretatik klaseen hierarkia bat finkatzen da, klase umea klase gurasoengandik heredatzen dela.

Polimorfismoa forma bat baino gehiago izateko tasuna da eta Pythonek ere integratzen du. Eragile berak objektu desberdinekin modu desberdinetan eragiteko ahalmena izendatzen du polimorfismoak OOPn, hau da, objektu desberdinek modu diferentean erantzuten diotela metodo berberari.

Ezaugarri horiek lantzeko ohiko adibidea dugu: poligonoak eta dagozkien azalerak kalkulatzeko moduak. 8.2. programan ikus daitekeenez, *Poligonoa* klasea definitzen da hasieran, hiru metodo dituela: eraikitzailea (*init*), datuak sartzekoa (*SartuAldeak*) eta alde neurriak inprimatzekoa (*InpriAldeak*). Gero *Triangelua* eta *Laukizuzena* klase eratorriak (herentzia bidez) definitzen dira, *class* espezifikazioan *Poligonoa* klasea zehaztuz eta metodo eraikitzailean ere aipatuz. *Azalera* metodoa bakoitzean definitzen da modu desberdinean eratorritako klase bakoitzerako, eta gainera *SartuAldeak* metodoa birdefinitzen da *Laukizuzena* klasean (bi alde emanda nahikoa da-eta). Polimorfismoaren adibideak dira azken bi metodoak, baina bigarrenetan heredatutako metodoa aldatu egin da.

```

class Poligonoa:
    def __init__(self, alde_kop):
        self.n = alde_kop
        self.aldeak = [0 for i in range(alde_kop)]

    def SartuAldeak(self):
        for i in range(self.n):
            self.aldeak[i] = float(input("Sartu aldea("+str(i+1)+") : "))

    def Inprialdeak(self):
        for i in range(self.n):
            print(i+1, "aldea", self.aldeak[i])

class Triangelua(Poligonoa):
    def __init__(self):
        Poligonoa.__init__(self,3)

    def Azalera(self):
        a, b, c = self.aldeak
        s = (a + b + c) / 2
        azal = (s*(s-a)*(s-b)*(s-c)) ** 0.5
        print('Triangeluaren azalera %0.2f' %azal)

class Laukizuzena(Poligonoa):
    def __init__(self):
        Poligonoa.__init__(self,4)

    def SartuAldeak(self):
        for i in range(2):
            self.aldeak[i] = float(input("Sartu aldea("+str(i+1)+") : "))
            self.aldeak[2] = self.aldeak[0]
            self.aldeak[3] = self.aldeak[1]

    def Azalera(self):
        a, b, c, d = self.aldeak
        azal = a*b
        print('Errektanguluaren azalera %0.2f' %azal)

t = Triangelua()
t.SartuAldeak()
t.Inprialdeak()
t.Azalera()

l = Laukizuzena()
l.SartuAldeak()
l.Inprialdeak()
l.Azalera()

```

8.2. programa. Objektuen arteko herentziaren adibidea.

Objektuek klase desberdinetatik heredatutako ezaugarriak eduki ditzakete. Horri askotariko herentzia esaten zaio, eta Pythonen erabil daiteke. Horretarako *class* espezifikazioan eta metodo eraikitzailean klase gurasoak zehaztu beharko dira.

8.2. MODULUAK INSTALATZEN

Sarean modulu asko eta asko daude eta, funtsezkoenak kenduta, gerta daiteke erabili nahi dugun modulu ez egotea instalatuta gure konputagailuan. *import* agindua ondo ibil dadin, dagokion moduluak instalatuta eta indexatuta egon behar du¹².

Moduluak kudeatzeko hainbat programa dago eta *pip* da horien artean ezagunena. *pip* erabiltzeko aurretik instalatu behar da. Linuxen, eta Python3.4-tik aurrera, honako hau egin behar da instalatzeko:

```
sudo apt-get install python3-pip
```

Beste bertsioetarako, mesedez kontsultatu URL hau: <<https://pip.pypa.io/en/stable/installing/>>.

Behin *pip* kudeatzailea edukita, paketeak instalatzea erraza da. Adib. *html2text* moduluak instalatzeko ondoko komandoetako bat nahikoa da¹³:

```
sudo python3 -m pip install html2text
sudo pip3 install html2text
```

Liburuaren sarreran esan den bezala, gaur egungo programazioan funtsezkoa da kodea berrerabiltzea. Horretarako, aurretik aipatutako moduluak inportatuko dira. Ondoren, eta hurrengo ataletan, eta kapituluetan, ohiko programetarako erabiltzen diren moduluak aipatu eta landuko dira.

Azalduko ez diren baina luburutegi estandarrean dauden metodoak eta klaseak hemen kontsulta daitezke: <<https://docs.python.org/3/library/>>.

8.3. KALKULURAKO MODULUAK

math modulu asko erabiltzen da, baina horrekin batera beste hauek ere: *random* zorizko zenbakietarako eta *statistics* estatistika-arlorako (<https://docs.python.org/3/library/numeric.html>). Oinarrizko eragiketetatik at, kalkuluak egin behar direnean modulu hauetako metodoak erabiltzea da ohikoena.

12. Paketeak edukitzen dituzten katalogoetan `__init__.py` izeneko fitxategi batek egon behar du, bestela ez da bertan bilatuko. Aipatutako fitxategi hutsa izan daiteke, baina batzuetan erabiltzen da hainbat aldagai hasieratzeko (<https://docs.python.org/3/tutorial/modules.html#packages>).

13. Kapitulu honetan modulu hori erabiliko dugu, beraz, instalatzea gomendatzen da.

8.3.1. Funtzio matematikoak

Has gaitzen *math* moduluaren metodo ohikoenekin (asko daude):

```
math.fabs(t)
```

t zenbakiaren osoko balio absolutua itzuliko du.

```
math.pow(o, b)
```

o ber b balioa itzuliko du.

```
math.sqrt(t)
```

t zenbakiaren erro karratu positiboa itzuliko du. t negatiboa bada, errorea gertatuko da.

```
math.exp(t)
```

t -ren esponentziala itzuliko du.

```
math.log(t)
```

t -ren logaritmo nepertarra itzuliko du. Bigarren parametro gisa oinarria zehatz daiteke

```
math.log10(t)
```

t -ren logaritmo hamartarra itzuliko du.

Trigonometria ere *math* moduluaren ardurapean dago:

```
math.sin(t)
```

t -ren sinua itzuliko du.

```
math.cos(t)
```

t -ren kosinua itzuliko du.

```
math.tan(t)
```

t -ren tangentea itzuliko du.

Hiru metodo trigonometriko horiez gain beste asko daude: *asin* (arku sinua), *acos* (arku kosinua), *atan* (arku tangentea), *sinh* (sinu hiperbolikoa), *cosh* (kosinu hiperbolikoa), *tanh* (tangente hiperbolikoa)...

Neurri gehienak radianetan ematen dira, baina radian/gradu bihurketetarako *radians* eta *degrees* metodoak erabil daitezke.

Balio singularrak ere lor daitezke modulu hau erabiliz:

```
math.pi
```

π balioa (3.141592...)

```
math.e
```

e balioa (2.718281...)

8.3.2. Zorizko zenbakiak eta estatistikak

Zorizko zenbakietarako, berriz, *random* modulua erabili behar da. Aukera handia dago, baina hauek dira erabilienak:

```
random.random()
```

Zoriz sortutako zenbaki erreal bat (0 eta 1 artean) itzuliko du.

```
random.seed(h)
```

Zorizko zenbakien sorrera hasieratzeko hazi (h) batekin.

```
random.randrange(n)
```

0 eta n-1 arteko zenbaki osokoak sortzeko. Adibidez 0 eta 9 arteko bat lortzeko 10 zehaztuko da.

```
random.sample(zerr,n)
```

Zerrendako elementuen artean n aukeratzeko zoriz. Zerrenda bat itzuliko du.

```
random.choice(s)
```

Sekuentzia baten barruko elementu bat aukeratzeko zoriz. Karaktere-kateak ere zehatz daitezke.

Adibide sinple batzuk (*idle* bitartez):

```
>>> import random
>>> random.random()
0.17970987693706186
>>> random.randrange(6)      # balio bat 0 eta 5 artean
4
>>> random.choice(['gora', 'behera', 'berdin'])
'gora'
>>> random.sample(range(100), 5)  # 5 balio 0 eta 99 artean
[30, 83, 16, 4, 8]
```

Zorizko zenbakiak asko erabiltzen dira. Adibidez katalogo bateko mp3 motako fitxategien artean bat aukeratzeko 8.3. programa erabil daiteke.

```
import os
import sys
import random

list = []
n = 0

for f in os.listdir(sys.argv[1]): # katalogoko fitxategi bakoitzeko
    if(f.endswith('.mp3')):      # bukaera egiaztatzeko
        n += 1
        list.append(f)

i=random.randrange(n)           # zorizko bat 0 eta n-1 artea
print(list[i])
```

8.3. programa. Abesti bat aukeratzea zoriz.

Estatistikan ohikoak diren neurriak ere erabil daitezke *statistics* moduluaz:

```
statistics.mean(zerr)
```

Zerrendako elementuen arteko batezbestekoa itzuliko du.

```
statistics.median(zerr)
```

Zerrendako elementuen arteko mediana itzuliko du.

```
statistics.mode(zerr)
```

Zerrendako elementuen arteko moda itzuliko du.

```
statistics.variance(zerr)
```

Zerrendako elementuen arteko bariantza itzuliko du.

```
statistics.stdev(zerr)
```

Zerrendako elementuen arteko desbiderapen tipikoa itzuliko du.

Kalkulu matematiko konplexuetarako eta estatistika sofistikatuak egiteko modulu interesgarriak daude, *NumPy* (www.numpy.org) edo *StatsModels* (statsmodels.sourceforge.net) esaterako, baina hori liburu honen esparrutik kanpo geratzen da.

8.4. DATUEN FORMATUAK

7. kapituluaren landutako ohiko testu-fitxategiaz gain gero eta gehiago erabiltzen dira trukerako formatuetan oinarritutako fitxategiak. XML da markaketa-lengoaia orokorra (hedagarria den neurria), baina CSV eta JSON lengoaien erabilera ere oso hedatuta dago. HTML ere formatu interesgarria da.

Serializatzeko edo markatzeko formatuekin hasi baino lehen datuen formatuekin lotutako hainbat modulu azpimarragarri aipatu behar dira:

- Unicode kodeketarekin lotutako moduluak: *unicodedata* (<https://docs.python.org/3/library/unicodedata.html>).
- Datu-trinkoketarekin lotutako moduluak: *glib*, *gzip*, *bz2*, *zipfile*, *tarfile* (<https://docs.python.org/3/library/archiving.html>).
- Zifraketarekin lotutako moduluak: *crypt*, *hashlib*, *hmac* (<https://docs.python.org/3/library/crypto.html>).

8.4.1. CSV

Estatistikako lanak batzuetan kalkulu-orrietatik edo datu-baseetatik ekarritako fitxategien gainean egiten dira. CSV (*Comma Separated Values*, komaz banatutako balioak) da esportazioetarako formatu erabilienetako bat.

Formatu horren gainean lan egiteko *csv* modulua dago. Barneratzen dituen metodoen artean honako hauek dira erabilienak:

```
csv.reader(f)
```

CSV fitxategia interpretatzeko metodoa. Parametro gisa irekitako fitxategia (*f*) behar du, eta, eremu-banatzailea koma ez bada, *delimiter* gakoa eta haren balioa (adibidez *delimiter=';'*). CSV motako objektu bat itzultzen du, iteragarria dena, fitxategien modura erabil daitekeena. Horrela lortutako objektuaren gainean errenkadak lor daitezke *in* eragilearen bidez, eta zutabeak indizearen bidez (0tik hasita).

```
csv.writer(f)
```

CSV formatura bihurtzeko metodoa. Parametro gisa idazteko irekitako fitxategia behar du. CSV motako objektu bat itzultzen du.

```
csv.writerow(z)
```

CSV motako objektu baten gainean errenkada bat idazteko balio du. Parametro gisa zutabe guztiak dituen zerrenda bat eman ohi zaio.

8.4a. programan erabileraren adibide bat dugu, hauteskundeen emaitzak dituen CSV fitxategi batetik hautagai baten emaitzak inprimatzen dituen. Erabiltzeko bi parametro pasa behar zaizkio: fitxategia eta hautagaiari dagokion zutabea¹⁴.

```
import csv
import sys

csvfitx = csv.reader(open(sys.argv[1]))
zut=int(sys.argv[2])
for errenk in csvfitx:
    print(errenk[0], errenk[1], errenk[zut])
```

8.4a. programa. Hauteskundeen fitxategitik hautagai baten emaitzak.

Aurreko metodoez gain CSV fitxategiak hiztegi moduan tratatzeko bi klase daude: *csv.DictReader* eta *csv.DictWriter*. Ondoko adibidean ikusi ahal izango denez, izenburu gisa dauden etiketak erabil daitezke hiztegiaren gako gisa programa egitean.

8.4b. programan sarrera CSV formatuan dagoen fitxategi bat da, non hainbat operaren datuak dauden: urtea, izena, egilea eta hizkuntza. Lehen lerroan izenburuak agertzen dira. Programaren helburua *Trivial* moduko joko baterako galderak automatikoki sortzea da. Sortutako galderak, erantzun zuzenarekin eta bi erantzun okerrekin idatziko dira CSV fitxategi batean.

14. Probak egiteko <http://www.euskadi.net/emaitzak/republica/result_1936/csv/mun_bizkaia_capital_1936_feb_e.csv> fitxategia hartu dugu eta *adibidea.csv* fitxategian gorde. Probatzeko adib. *python3 p8-4a.py adibidea.csv 7*

```
#!/usr/bin/python3
# -*- kodeketa: utf-8 -*-

# CSVak kudeatzeko:
import csv
# urteak zoriz sortzeko
import random

sar_fitx = "operak-sar.csv"
irt_fitx = "operak-irt.csv"

# sarrera fitxategia ireki
with open(sar_fitx, newline='') as csv_sarrera:
    # sarrera fitxategia hiztegi batean kargatu
    firak = csv.DictReader(csv_sarrera)

    # irteera fitxategia ireki
    with open(irt_fitx, 'w') as csv_irteera:
        # irteera fitxategian idazteko aldagaia prestatu (hiztegia)
        # (CSV formatua izanik, dena kudeatzeko objektua sortu dugu)
        eremuak = ["Galdera", "Zuzena", "Oker1", "Oker2"]
        fidaz = csv.DictWriter(csv_irteera, fieldnames=eremuak)
        fidaz.writeheader()

        # aurrena fitxategiko idazle guztiak jasoko ditugu
        # gero distraigarriak sortu ahal izateko
        jatorrizko_fitxategia = list(firak)
        idazleguztiak = set()
        for elementu in jatorrizko_fitxategia.copy():
            idazleguztiak.add(elementu["egilea"].strip()) # strip*

        # galderak sortuko ditugu eta fitxategian idatzi
        for errenkada in jatorrizko_fitxategia:
            galdera = "Nork idatzia da "+errenkada["izena"].strip()+" opera?"
            zuzena = errenkada["egilea"].strip()
            # okerrak sortu (3, badaezpada bat zuzena den)
            okerrak = random.sample(idazleguztiak, 3)
            # okerren artean zuzena badago, kendu
            if zuzena in okerrak:
                okerrak.remove(zuzena)
            # lehenengo bi elementuak jaso okerren zerrendatik
            oker1, oker2 = okerrak[0:2]
            fidaz.writerow({"Galdera": galdera, "Zuzena": zuzena,
                           "Oker1": oker1, "Oker2": oker2})
```

8.4b. programa. CSV formatua, hiztegi moduan.

Probatu ezazu programa. Ez du parametririk behar.

Honetaz gehiago sakontzeko gune hau gomendatzen da: <<https://docs.python.org/3/library/csv.html>>.

*. *strip* metodoa erabiltzen da karaktere-kateak garbitzeko.

8.4.2. JSON

JSON gero eta gehiago erabiltzen den formatua dugu. Interneteko hainbat APIk JSON (*JavaScript Object Notation*, *JavaScript* objektuen idazkera) formatuan itzultzen dituzte emaitzak. Zerbitzu eraginkorrak eskaintzeari zuzenduta dago. Programetan erabilitako datuak serializatzea ere du helburu, beraz, programen arteko komunikazio eraginkorrerako da egokia.

Formatu horren gainean lan egiteko *json* modulua dago. Barneratzen dituen metodoen artean, honako hauek dira erabilienak:

```
json.dumps
```

JSON formatura bihurtzeko metodoa. *separators* izeneko parametroaren bidez defini daitezke banatzaileak.

```
json.loads
```

JSON formatuko segidak interpretatzeko metodoa. Fitxategi batean badago, fitxategia ireki (*open*) eta lerro bakoitzari aplikatu egiten zaio. *json.load* metodoa erabil daiteke fitxategi osoa bihurtzeko, baina ez da ohikoa.

JSON formatuan informazioa bikoteetan (gakoa/balioa) egon ohi da eta aurreko metodoaren bidez Python hiztegi bihurtzen dira. Ohikoa da balioa hiztegia ere izatea, kasu horretan hiztegia indexatzeko bi indize (edo gehiago) erabiliko dira.

8.5. programan erabileraren adibide bat dugu, non Twitter API bidez jasotako JSON formatuko txioak eskuratu eta inprimatzen diren (identifikadorea, hizkuntza eta mezua). Erabiltzeko parametro bat pasa behar zaio: JSON formatuko fitxategia¹⁵.

```
import json, sys

f = open(sys.argv[1], 'r')
# json erregistro bakoitza lerro batean
for lerro in f:
    txio = json.loads(lerro)
    try:
        testu = txio['text']
        id = txio['id_str']
        hizk = txio['user']['lang'] # hiztegia hiztegiaren barruan
        print(id, hizk, testu)
    except:
        continue # ezer ez egitea
```

8.5. programa. JSON formatuko txioen tratamendua.

Honetaz gehiago sakontzeko gune hau gomendatzen da: <<https://docs.python.org/3/library/json.html>>.

15. Probak egiteko <<https://gist.github.com/hrp/900964>> fitxategia hartu dugu eta *twitter.json* fitxategian gorde. Probatzeko adib. *python3 p8-5.py datu_twitter.json*

8.4.3. XML gaineko metodoak

XML, *eXtensible Markup Language*, dokumentuak *markatzeko* edo etiketatzeko estandarra da. Markatze-etiketak eta edukia bera testua dira, eta hortaz testu-fitxategiak dira, non etiketak eta datuak txertatzen diren. Metalengoaia da XML, eta bertatik eratorriko lengoaia asko dago¹⁶. Horrela, HTML bera XMLtik eratorritako lengoaia gisa ikus daiteke eta ondo eratutako HTML dokumentuak (XHTML) XML dokumentuak dira¹⁷.

Edozein kasutan, programazio-lengoaia gehienetan liburutegi ugari daude XML egiturak sortzeko, irakurtzeko edota aldatzeko. Eta Python ez da salbuespena.

Pythonek hainbat *built-in* modulu eskaintzen ditu XML fitxategiak prozesatzeko, *ElementTree*, DOM, edota SAX adibidez. SAX gauza sinpleak egiteko pentsatuta dago eta DOMek azkartasunari begiratzen dio, XML fitxategi osoa memorian kargatuz. *ElementTree*-k, berriz, XML zuhaitz gisa errepresentatzen du XML fitxategia, eta modu erraz eta eraginkorrean XML zuhaitzen gainean eragiketak egiteko baliagarria da. Azken hau izango da adibideetan erabiliko duguna.

Badaude ahaltsuagoak diren liburutegiak ere, *lxml* adibidez, baina funtzio eta atributu ugari partekatzen dituzte *ElementTree* moduluarekin, hortaz atal honetan Pythonen berezko moduluarekin lan egingo dugu.

Honako XML dokumentua erabiliko dugu hurrengo programetan (*herrialdeak.xml*):

```
<?xml version="1.0"?>
<datuak>
  <herrialdea izena="Portugal">
    <posizioa>111</posizioa>
    <hizkuntza>portuges</hizkuntza>
    <azalera>92212</azalera>
  </herrialdea>
  <herrialdea izena="Argentina">
    <posizioa>8</posizioa>
    <hizkuntza>espainola</hizkuntza>
    <azalera>2780400</azalera>
  </herrialdea>
```

16. Hona hemen zerrenda luze bat: <https://en.wikipedia.org/wiki/List_of_XML_markup_languages>.

17. Hemen ez gara gaiaz luzatuko, baina honetaz sakondu nahi duenak dokumentu hau irakurtzea gomendagarria da: <<http://www.unibertsitatea.net/otarra/ingeniaritza-eta-teknologia-1/informatika/xml-oinarriak>>.

```
<herrialdea izena="Mongolia">
  <posizioa>19</posizioa>
  <hizkuntza>mongoliera</hizkuntza>
  <azalera>1564115</azalera>
</herrialdea>
</datuak>
```

ElementTree moduluak XML dokumentua zuhaitz moduan errepresentatzeko aukera ematen du. XML dokumentuaren erroa *Element* motako objektu batean kokatuta, zuhaitzean zehar mugitzeko eta bilaketak egiteko aukera ematen du modu errazean¹⁸.

Adibidez, aurreko irudiko XML egituraren, herrialdea da objektuaren adabegi edo nodo nagusia, gero, hiruak maila berean, posizioa, hizkuntza eta azalera.

Informazioa aurkitzeko metodo oso sinple zein ahaltsuak eskaintzen ditu: *findall*, *find* eta *findtext*. Metodo horiek parametro gisa pasatako elementua edota elementuaren testua aurkitzeko erabiltzen dira. *findall* metodoak parametroaren adierazpena betetzen duten ume guztiak itzultzen ditu zerrenda batean, *find* metodoak adierazpena betetzen duen lehenengo elementua, eta *findtext* metodoak elementuaren testua (datua). Testua eskuratzeko *attrib* metodoa ere erabil daiteke elementuaren gainean. Parametro gisa *Xpath* espresioak ere erabil daitezke.

8.6. programan, halako aipatutako XML dokumentua irakurri, eta hiztegi bat sortuko dugu herrialdeen izenekin eta dagozkien hizkuntzekin.

```
import xml.etree.ElementTree as ET

#XML dokumentua irakurri eta zuhaitz gisa jaso Element objektu batean
tree = ET.parse('herrialdeak.xml') #fitxategitik XML jaso zuhaitz moduan
root = tree.getroot() #Erro elementua jaso Element motako objektuan

hiztegia = dict()
#erro elementuaren umeak diren 'herrialdea' elementuak
for herrialde in root.findall('herrialdea'):
    # herrialdearen "izena" atributuko balioa jaso
    izena = herrialde.attrib['izena']
    # hizkuntza umearen testu balioa jaso
    hizkuntza = herrialde.findtext('hizkuntza')
    hiztegia[izena] = hizkuntza

print(hiztegia)
```

8.6. programa. XML fitxategiaren erabilera.

18. *ElementTree* moduluari buruz gehiago ikasteko Pythonen dokumentazioa ikus daiteke <<https://docs.python.org/3/library/xml.etree.elementtree.html>>.

HTML fitxategiekin lan egiteko ere berdin-berdin balio du modulu honek, beti ere ondo eratuta baldin badaude, hau da, XHTML fitxategia bada. Web orriekin lan egiteko adibide gehiago azalduko dira 10. kapituluan.

8.5. KONEXIOA DATU-BASEEKIN

Datu-base mota ugari daude gaur egunean erabilgarri, *Oracle*, *MySql*, *Postgres*, *SqlServer*, *SQLite*¹⁹... Erabileraren arabera datu-base mota bat erabiliko du garatzaileak, weberako *MySql*, enpresa munduko gauzetarako *Oracle* edo *Postgres*... Pythonek horiek guztiak kudeatzeko moduluak eskaintzen ditu: *MySQLdb*, *cx_Oracle*...

Liburu honetan baina, *SQLite* datu-baseak erabiltzen ikasiko dugu. *Lite* hitzak txikia, ez oso pisutsua, esan nahi du ingelesez, eta horregatik *SQLite*-ri datu-base txertatua ere esaten zaio. *MySQL* edo *Oracle* aparteko softwareak dira, eta horrela kanpotik konektatu beharra dago beraiekin lan egiteko. *SQLite* aldiz ez, garatzen ari garen aplikazioaren barruan gordetzen da. Gaur egunean oso erabiliak dira musika-jotzaileetan, mugikor adimendunetan, autoetan...

Pythonek *built-in* modulu bat dauka *SQLite* motako datu-baseekin lan egiteko: *sqlite3*. Hortaz, aparteko ezer instalatu gabe erabil ditzakegu modulu horren metodoak gure programetan.

Atal honetan datu-base batera konektatzen ikasiko dugu, baita taula berri bat sortzen eta tauletan eragiketak egiten ere. Azkenik, datuak modu eraginkorrean nola jaso ditzakegun ikusiko dugu.

8.7. programan datu-base batekin konexioa ezarri eta taula berri bat sortzen da. Ostean, sortutako taula berrian datuak sartuko dira eta, bukatzeko, taulatik datuak jaso eta pantailaratuko dira.

Emandako urratsak honakoak dira:

- Datu-basera konexioa oso modu errazean burutzen da *connect* metodoaren bitartez. Datu-basea ez bada aurretik sortu, momentu horretan sortuko da uneko katalogoan.
- Hainbat SQL espresio egikarituko dira datu-basearen gainean *execute* metodoa erabiliz: taula berria sortu ("*CREATE TABLE*"), errenkadak gehitu ("*INSERT*"), eta datuak jaso ("*SELECT*"). Edozein SQL espresio egikaritu daiteke metodo honi esker.
- Egindako aldaketak datu-basean gorde daitezen *commit* metodoa erabiltzen da.
- Datu-basearekin konexioa ixten da *close* metodoaz.

19. Guztiak SQL datu-baseak dira. Gaur egun NoSQL datu-baseak ere erabiltzen dira (MongoDB esaterako), baina liburu honen esparrutik kanpo geratzen dira.

```

import sqlite3

conn = sqlite3.connect('proba.db') #Datu basearekin konexioa ireki
#Datu-baseari taula bat gehitu:
conn.execute('''CREATE TABLE Herrialdea          #Taularen izena
               (ID INT PRIMARY KEY             NOT NULL,   #Taularen gakoak
                Izena          TEXT              NOT NULL,   #Testu motako atributua
                Azalera        INT               NOT NULL);''')#Zenbaki oso motako atributua

#Herrialdea taulari errenkadak txertatu:
conn.execute("INSERT INTO Herrialdea (ID,Izena,Azalera) \
              VALUES (1,'Argentina',2780400)")
conn.execute("INSERT INTO Herrialdea (ID,Izena,Azalera) \
              VALUES (2,'Portugal',92212)")
conn.execute("INSERT INTO Herrialdea (ID,Izena,Azalera) \
              VALUES (3,'Mongolia',1564115)")

conn.commit() # Aldaketak gordetzeko
#Herrialdea taulako errenkadak pantailaratu:
kursore = conn.execute("SELECT ID, Izena, Azalera from Herrialdea")
for errenkada in kursore:
    print("ID =", errenkada[0])
    print("Izena =", errenkada[1])
    print("Azalera =", errenkada[2])

conn.close() #Datu basearekin konexioa itxi

```

8.7. programa. SQL datu-basearen gaineko eragiketak.

8.6. GRAFIKOAK ETA IRUDIAK

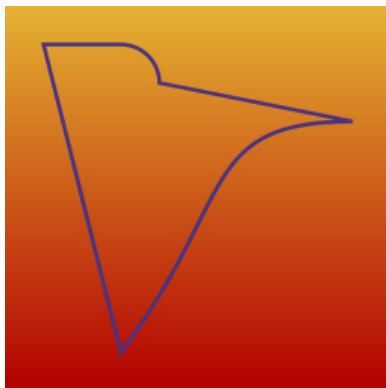
Grafikoak eta irudiak ez ohi dira sortu programazio-lengoaia orokorren bitartez. Dena den, programa batzuetan emaitzak modu grafikoan ematea oso ezaugarri garrantzitsua izan daiteke. Modulu asko daude lan horretan laguntzeko, horietako gutxi batzuk baino ez ditugu landuko.

8.6.1. Irudiak

Pythonen, beste programazio-lengoaietan bezala, oinarritzko tresnak erabiliz oso nekeza da irudiak sortzea, baina publikoak diren modulu eta klaseei esker lan hori asko sinplifika daiteke eta oso emaitza ikusgarriak lortu lan askorik egin gabe.

Irudietarako ezagunen diren Python moduluak hauek dira: *graphics.py* eta *pycairo*. Guk *pycairo* probatu dugu, oso ahaltsua delako eta beste lengoaietan ere (C++, Perl, PHP...) erabiltzen delako.

Ondoko adibidean *pycairo* moduluaren erabilera landuko dugu, ahalmenaren eta oinarritzko ezaugarrien adibide gisa. 8.1. irudian agertzen den irudia 8.8. programak sortu du.



8.1. irudia. *pycairo* bitartez lortutako grafiko baten adibidea.

Programa (helbide honetan <http://cairographics.org/pycairo/tutorial/>) aurkitutako adibide batetik eratorria da) hainbat bloketan dago banatuta:

- Moduluen (*math* eta *cairo*) kargatzea.
- Hondoaren definizioa: *cairo* objektuaren bidez lantzen da hasieran (*ImageSurface* eta *Context* metodoak erabiliz). Azken metodoak beste objektu bat eskaintzen digu (testuingurua: *testuing*); eta horren gainean hainbat metodo aplikatuz (*rectangle* eta *fill*) eta objektu laguntzaile baten laguntzaz (*bide*) hondoa ezarrita geratzen da.
- Marrazkia bera metodo geometrikoen bidez: *testuing* objektuaren gainean marrazteko tipikoak diren metodoak erabiltzen dira: *arc* (arku zirkularretarako), *line_to* (lerroetarako), *curve_to* (kurbetarako)...
- Marrazkiaren zehaztasunak eta idazketa. Lerroen kolorea eta zabalera definitu ondoren, marrazkia finkatzen da eta *png* formatura bihurtzen da.

```
import math
import cairo

#neurriak 256 puntuko karratua
ZABAL, LUZ = 256, 256
# hondoa definitu
azalera = cairo.ImageSurface (cairo.FORMAT_ARGB32, ZABAL, LUZ)
testuing = cairo.Context (azalera)
testuing.scale (ZABAL, LUZ) # eskala-normalizazioa
# hondoko kolorea
# prestakuntza -> bide
bide = cairo.LinearGradient (0.0, 0.0, 0.0, 1.0)
# kolore degradatua bi zatitan
bide.add_color_stop_rgba (1, 0.7, 0, 0, 0.5)
bide.add_color_stop_rgba (0, 0.9, 0.7, 0.2, 1)
# hondoa marraztu eta koloreztatu
testuing.rectangle (0, 0, 1, 1) # errektangulua (x0, y0, x1, y1)
testuing.set_source (bide)
testuing.fill ()
```



```
# marrazketa
testuing.translate (0.1, 0.1) # Abiapuntu-aldaketa
testuing.move_to (0, 0)
# arku zirkular bat (cx, cy, erradioa, hasiera_angulua, bukaera_angulua)
testuing.arc (0.2, 0.1, 0.1, -math.pi/2, 0)
# lerro zuzen bat (x,y)-raino
testuing.line_to (0.8, 0.2)
# kurba bat 3 puntuz (x1, y1, x2, y2, x3, y3)
testuing.curve_to (0.4, 0.2, 0.5, 0.4, 0.2, 0.8)
# hasierara itzuli
testuing.close_path ()

# arkatzaren kolorea eta lodiera
testuing.set_source_rgb (0.3, 0.2, 0.5)
testuing.set_line_width (0.01)
# finkapena
testuing.stroke ()
# fitxategira
azalera.write_to_png ("adib-marrazki.png") # irteera png formatoan
```

8.8. programa. Marrazketaren adibidea *cairo* modulua erabiliz.

Kontuan hartu behar da metodoak objektu desberdinen gainean aplikatzen direla: *cairo* gainean hasieran (*cairo.ImageSurface* eta *cairo.Context* esaterako), baina baita metodo horiek itzultitako objektuen gainean ere (*testuing.arc*, *testuing.line* edo *azalera.write_to_png* esaterako).

8.6.2. Grafikoak

Bisualizazio-tresnak gero eta garrantzia handiagoa hartzen ari dira informazioa azaleratzeko orduan. Datu numerikoak aurkezteko tresnak gero eta zabaldutako daude, eta atal honetan Pythoneko aukera interesgarrienak aipatu eta adibide bat landuko dugu.

Matplotlib da modulu zahar eta ezagun bat, baina oso oinarritzkoa da. Liburutegi horretan oinarrituta beste batzuk sortu dira lana errazteko moduan: *Pandas*, *Seaborn*, eta *ggplot*. *Plotly* ere aipa daiteke, hodeian zerbitzu bat delako, baina Pythonetik atzitzeko API bat daukana. Informazio gehiago kontsulta daiteke web orri honetan: <<http://pbpython.com/visualization-tools-1.html>>. Bertatik ere hartu ditugu adibidearen ideia nagusiak.

Pandas izango da adibidean erabiliko duguna. *Pandas* (<https://pypi.python.org/pypi/pandas>) ahaltsua eta malgua da, *NumPy* moduluan oinarritzen da kalkulu numerikorako, eta R lengoaiarekin lan egiten dutenentzat gertuko ezaugarriak ditu.

Erabili baino lehen, *matplotlib*, *numpy* eta *pandas* moduluak instalatu behar dira (denbora luze samar hartzen du):

```

sudo apt-get install libfreetype6-dev      # gerta daiteke ez izatea
sudo python3 -m pip install numpy
sudo python3 -m pip install matplotlib
sudo python3 -m pip install pandas

```

Bisualizatzeko datu-fitxategia, berriz, CSV ataleko adibidean erabiliko dugu gure adibidean, CSV izan ohi baita ohiko formatu bat datu-analisietako atazetan.

8.9. programan dugu *pandas* erabiltzeko erraztasuna adierazten duen adibide bat. Bertan barren bidezko grafiko bat idazten da (*png* fitxategi batera) 6 urratsetan:

- moduluen karga *import* bitartez
- *csv* fitxategia irakurtzea *read_csv* metodoaz
- boto-emaleen araberrako sailkapena eta erregistroen selekzioa (4 handienak) *sort_values* metodoaren bitartez
- grafikoa sortzea *plot* metodoaz
- irudi bihurtzea, *get_figure* metodoaz
- *png* formatuan gordetzea *savefig* metodoaz

```

import pandas

bozk = pandas.read_csv("adibidea.csv")
# boto-emaleez sailkatu eta handiena (totalak) kendu eta lehen 4ak hartu
bozk = bozk.sort_values('Boto-emaleak',ascending=False)[2:6]

# graf mota (bar), x ardatzean Herria
bozk_plot = bozk.plot(kind="bar",x=bozk["Herria"],
                      title="Parte hartzea (lehen 4ak)",
                      legend=False)

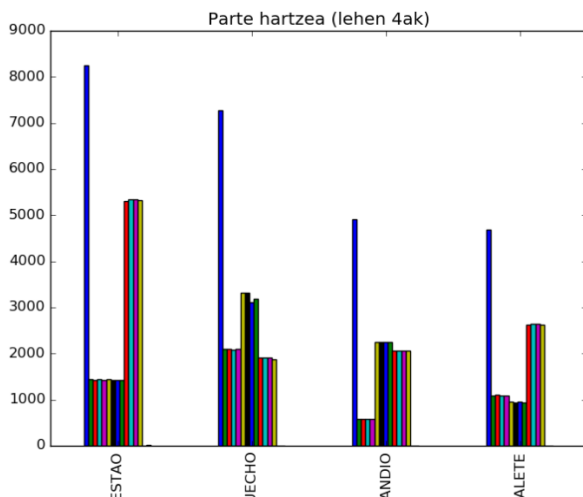
fig = bozk_plot.get_figure()
fig.savefig("adibidea-plot.png")

```

8.9. programa. Datuak modu grafikoan *pandas* moduluaren bitartez.

Aurreko adibidean bezala, metodoak objektu desberdinen gainean aplikatzen dira: *pandas* gainean hasieran, honek itzulitako *bozk* gainean gero, eta prozesuan sortutako *bozk_plot* eta *fig* objektuen gainean bukaeran.

Lortutako irudia 8.2. irudian ikus daiteke.



8.2. irudia. Datuetatik lortutako grafikoa.

8.7. BESTELAKOAK

Luze joko luke Pythoneko liburutegi osoa, bere modulu guztiekin, azaltzeak. Kapituluak bukatu baino lehen multzo bat bakarrik: internazionalizazioari dagokiona.

8.7.1. Internazionalizazioa (*i18n*)

Hainbat hizkuntzataraz egokitutako programak egiteko internazionalizazio deritzon prozesua funtsezkoa da. Mezuak, zenbakien zein daten idazkera eta abar egokitu behar dira hizkuntzen arabera. Hizkuntza bakoitzaren egokitzapen horri *locale* izena ematen zaio eta prozesuari *lokalizazioa*. Internalizazio/lokalizazio prozesu horietarako Pythonen ezinbestekoa da *gettext* eta *locale* moduluak, beste lengoia askotan ere erabiltzen direnak, ulertzea eta erabiltzea.

Testuen itzulpenari dagokionez, *gettext* moduluak *.po* eta *.mo* motako fitxategietan oinarritzen da. Bost urratsetan oinarritzen da internazionalizatzeko egokitutako programa:

- Inprimatzeko mezuak hizkuntza lehenetsian idazten dira (ingelesez gure adibidean) programan, baina *print(mezu_aldag)* edo *print("mezu")* zehaztu beharrean *print(_(mezu_aldag))* edo *print(_("mezu"))* zehaztuz. *u* hori garrantzitsua da Unicode aplikatzen denean.
- Mezuak lortu jatorrizko programatik *pygettext* programaren bidez, eta *.pot* luzapena duen fitxategi batean gorde (*messages.pot* fitxategian geratzen dira besterik esaten ez bada).

- Prestatu mezuak hainbat hizkuntzatarako eta gorde *.po* eta *.mo* formatuan (bana hizkuntza bakoitzeko). Horretarako editore berezi bat erabil daiteke, adibidez *poedit*²⁰. *.mo* eta *.po* fitxategiak aplikazioari dagokion katalogoaren azpian: gure kasuan *locale/eu/LC_MESSAGES/proba-il8n.mo*
- Berriro programan, hasieran, *gettext* modulua barneratu *import* bitartez, eta *gettext.translation* metodoaz mezuak itzultzeko lotura egin (proiektuaren izena eta katalogoa zehaztuz). Lotura egin eta gero *install* metodoa ere zehaztuko da.

8.10. programan ikus daiteke adibide bat, aipatutako 1. eta 4. urratsak barneratzen dituen.

```
import gettext
lokal = gettext.translation('proba-il8n', localedir='locale', languages=['eu'])
lokal.install()

try:
    print(_(u"Hello world"))
    print(_(u"il8n for internazionalization"))
    print(_(u"l10n for localization"))
    print(_(u"Bye"))
except:
    print(_(u"Error in program"))
```

8.10. programa. Internazionalizaziorako proba.

Aipatutako 2. eta 3. urratsak egiten badituzu eta *p-il8n.py* programa probatzen baduzu, honako emaitza aterako da exekutatzean:

```
Kaixo mundua
il8n intenazionalizaziorako
l10n lokalizaziorako
Agur
```

Nahiz eta programaren mezuak ingelesez egon. Gauza bera gertatuko da *gettext.translation* metodoa zehazten ez baduzu eta sistema euskaraz baduzu²¹. Kasu horretan sistemak ezarrita duen hizkuntzan ariko da.

Oinarrizko informazioa hemen kontsulta daiteke: <<https://docs.python.org/3/library/i18n.html>>. Adibide osagarri bat hemen aurki daiteke: <<http://inventwithpython.com/blog/2014/12/20/translate-your-python-3-program-with-the-gettext-module/>>.

Bukatzeko, *locale* moduluen bidez Python kodean lokalizazioari dagozkion aldagaiak ezar daitezke, *LC_ALL* batez ere. *getlocale* eta *setlocale* metodoak dira

20. Instalatzeko: *sudo apt-get install poedit*

21. Euskaraz duzula egiaztatzeko komando hau erabil dezakezu: *echo \$LANG*

erabilienak. Adibidez aurreko programan ondoko bi lerroak gehitzen baditugu hasieran²²:

```
import locale
locale.setlocale(locale.LC_ALL, 'eu_ES.utf8')
```

8.8. PROPOSATUTAKO ARIKETAK

- 1) Sortu modulu bat liburutegi simple baterako. Oinarrizko objektua dokumentua izango da, eta bertatik eratorriko dira liburuak eta aldizkarietako artikulak. Dena paperean. Oinarrizko metodoak hauek izango dira: biltegitratzea, inprimatzea, maileguan uztea eta mailegutik itzultzea.
- 2) Moldatu 8.3. programa zure argazkien artean zoriz 5 hautatzeko.
- 3) Esportatu zure kalkulu-orri bat eta 8.4. programan oinarrituta inprimatu hainbat eremu. Kalkulatu zutabe edo errenkada baten batezbestekoa eta desbiderapen tipikoa. Sortu grafiko bat *Pandas* bitartez.
- 4) Datu-baseen gaineko metodoak erabiliz, 8.7. programaren kodea zabaltu datu-baseko datuak editatzeko. Probatzeko zuzendu Argentinaren azalera.
- 5) Datu-base berri bat sortu behar duzu *csv* fitxategi batetik. Horretarako *WorldData* info gunetan dagoen fitxategia erabiliko da (<https://www.worlddata.info/downloads/countries.csv>) eta bertan dauden informazioak barneratuko dira datu-basean. Bigarren urrats moduan herrialdeen hizkuntza nagusiak (gehienez 3) gehitzeko aukera programatu behar duzu.

22. Instalatuta dituzun *locale*-ak ezagutzeko komando bat dago: *locale-a*.

9. Sistemaren gaineko aukerak

Lau azpiataletan banatuko ditugu Pythonetik balia daitezkeen aukerak: sistemaren komandoak, administrazioa, sistema-deiak eta *multithreading* bidezko konkurrentzia.

Lehen bi ataletako kontzeptuak lantzeko on-line dagoen liburu hau gomendatzen dugu: *Linux: Sistemaren eta sarearen administrazioa* (<http://ueu.org/download/liburua/LINUXDebian.osoa.pdf>). Sistema-deien inguruan dokumentu hau kontsulta daiteke: <http://www.sc.ehu.es/acwlaalm/seo/teoria/2-gaia.pdf>.

Zazpigarren kapituluak aipatu den *os* moduluak aukera handiak eskaintzen ditu sistema eragilearekiko interakziorako. Nagusiki bi dira SEak eskaintzen dituen eragiketak: komandoak eta sistema-deiak. Komandoak programa exekutagarriak izan ohi dira eta sistema-deien bidez programatu ohi dira. Sistema-deiak, berriz, SEaren oinarritzko funtzioak dira eta sistemen programazioaren oinarria. Informazio gehigarria URL honetan aurki daiteke: <https://docs.python.org/3/library/os.html>.

Kontuan hartu behar da Unix/Linux sistemetan bakarrik direla erabilgarriak ondoren azalduko ditugun metodoak.

9.1. SISTEMAREN KOMANDOAK

Komandoak exekutatzeko biderik erosoena *os.system* metodoa erabiltzea da. Parametro gisa komando-lerroa adierazten duen karaktere-katea izango da.

Adibidez 9.1. programan honako hau egiten da: *probal* fitxategia bertan sortu ('l' karakterea izango da haren edukia), *berri* izeneko katalogo berri bat sortu, bertara aurreko fitxategia kopiatu (*laneko* izenarekin) eta hasierako fitxategia (*probal*) ezabatu.

```
import os
os.system("echo 'l' >probal")
os.system("mkdir berri")
os.system("cp probal berri/laneko")
os.system("rm probal")
```

9.1. programa. Komandoen erabilera Pythonetik.

Modu orokor horrez gain, komando espezifikoak adierazteko metodoak ere eskaintzen ditu modulu honek. Arazotxo bat dute metodo hauek, haien izena

batzuetan komando berarena da (*chown*, *chroot*, *chmod*, *mkdir*, *mkfifo*...), baina beste batzuetan beste luzeago bat (*link* eta ez *ln*, *remove* eta ez *rm*, *rename* eta ez *mv*...).

```
import os
os.system("echo '1' >probal")
os.mkdir("berri")
os.system("cp probal berri/laneko")
os.remove("probal")
```

9.2. programa. Komandoei dagozkien metodo espezifikoak.

9.2. programa aurrekoaren baliokidea da, baina kasu honetan *mkdir* eta *remove* metodo espezifikoak erabili dira *system* metodo orokorraren ordez.

9.3. programan, berriz, sistemaren kontrolerako komando batzuen erabilera ikus daiteke. *top* erabiltzen da prozesadorearen egoeraz jabetzeko, *vmstat* memoriarako, *df* disko-partizioetarako eta *ifconfig* sare-interfazeetarako.

```
import os
print("CPU:")
os.system("top -b -n 1 | head -2")
print("\nMemoria:")
os.system("vmstat")
print("\nPartizioak:")
os.system("df")
print("\nSare-interfazeak:")
os.system("/sbin/ifconfig")
```

9.3. programa. Komandoei dagozkien metodo espezifikoak.

Komandoen exekuzioak itzultzen duena errore-kodea izan ohi da, baina eragina normalean irteera estandarrean islatzen da (edo, adibideetan egin den bezala, fitxategiren batean irteera estandarra birbideratuz). Irteera hori aldagairen batean eskuratzeko *subprocess* moduluen *check_output* metodoa erabil daiteke. Adibidez:

```
emaitza = subprocess.check_output('df')
```

Bestalde, komandoen erabilera errazteko, eta batez ere sistemaren eraginkortasunaren kontrola errazteko, *psutil* modulua erabil daiteke (<https://github.com/giampaolo/psutil>). Bertako metodoen bidez sistemaren baliabideak (CPU, memoria, diskoa, sarea...) nola erabiltzen ari diren monitoriza daiteke. Aurretik instalatu behar da komando honen bidez:

```
sudo apt-get install python-psutil
```

9.4. adibidean erabileraren adibide bat agertzen da.


```
import psutil
print("CPU:\n", psutil.cpu_times())
print("\nMemoria:\n", psutil.virtual_memory())
print("\nPartizioak:\n", psutil.disk_partitions())
print("\nSare-interfazeak:\n", psutil.net_if_addrs())
```

9.4. programa. psutil moduluaren metodo espezifikoak.

9.2. ADMINISTRAZIOA

Administrazio-lanak burutzeko komandoak eta konfigurazioa-fitxategiak erabiltzen dira. Komando-fitxategiak osatu ohi dira lanak automatizatzeko *scripting* delakoaren bidez. Komando-fitxategi horietan komandoak eta *script*-lengoaia konbinatu ohi dira eta Python, gero eta gehiago, erabiltzen da *script*-lengoaia gisa. Adibide batzuen bitartez landuko ditugu²³.

Hasteko adibide bat: testatzeko script bat (wifi konexioarekin bakarrik proba daiteke). Linuxen '*sudo iw wlan0 scan*' komandoaren bidez inguruan dauden sareen informazioa jaso daiteke (baimena behar da; hori dela-eta pasahitza eskatuko digu sistemak). Ariketa honetarako lerro desberdinetan dauden eremu hauek interesatzen zaizkigu: identifikazioa (SSID) eta seinalearen kalitatea (*signal*). 9.5. programan inguruko wifi konexioen datu horiek selekzionatzen ditugu.

```
import os
os.system("sudo iw wlan0 scan >tmp")
for lerro in open('tmp', 'r').readlines():      # lerroka
    if "signal" in lerro:
        ahal = lerro
    if "SSID" in lerro:
        print(lerro, ahal)
os.system("rm tmp")
```

9.5. programa. Inguruko wifien informazio laburtua.

Gauza konplexuagoak egin daitezke, adibidez konexioa aktibatzea. Ikus adib. <<http://stackoverflow.com/questions/20470626/python-script-for-raspberrypi-to-connect-wifi-automatically>>.

Bigarren adibidean sistemaren erabileraz arituko gara. '*ps -aux*' komandoaren bidez lortzen den informazioa multzokatzen da erabiltzaileen arabera 9.6. adibidean, erabiltzaile bakoitzeko CPUren portzentaje metatuak lortuz.

23. URL honetan informazio gehiago horretaz: <<http://www.linuxjournal.com/content/python-scripts-replacement-bash-utility-scripts>>.

Hona hemen komandoaren lehen lerroen adibide bat:

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	33736	3044	?	Ss	16:59	0:01	/sbin/init
root	2	0.0	0.0	0	0	?	S	16:59	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	S	16:59	0:00	[ksoftirqd/0]
root	5	0.0	0.0	0	0	?	S<	16:59	0:00	[kworker/0:0H]
root	7	0.0	0.0	0	0	?	S	16:59	0:01	[rcu_sched

Lehen eta hirugarren zutabeak dira interesatzen zaizkigunak. Lehena erabiltzailea denez, hiztegia indexatzeko erabil dezakegu. Hiztegi batean CPUren balioak metatuko ditugu (lagungarria izango da 6. kapituluaren landutako hiztegiak).

```
import os

cpu = dict()

os.system("ps -aux >tmp")

for lerro in open('tmp','r').readlines(): # lerroka
    hitzak = lerro.split()
    if hitzak[0]!='USER':
        # lehen lerroa ez
        if hitzak[0] in cpu:
            # hiztegia?
            cpu[hitzak[0]] += float(hitzak[2])
        else:
            cpu[hitzak[0]] = float(hitzak[2])

for erab in cpu:
    # erabiltzaileka
    print(erab,cpu[erab])

os.system("rm tmp")
```

9.6. programa. CPUren erabilera erabiltzaileen arabera.

Beste aukera bat dago, Python programatik komandoen erreferentzia kendu eta komando-lerro hau idatzi:

```
ps -aux | python3 p9-6.py
```

Gainera, programaren sarrerak ez du *tmp* izan behar, sarrera estandarra baizik. Hori dela-eta, irakurketa errazten da, honela geratuz:

```
for lerro in sys.stdin:
```

Konplexutasuna igoz, 9.7. adibidean, azkenik, <http://askubuntu.com/questions/547437/find-delete-duplicated-files-on-multiple-harddisks-at-once> web orrian aurkitutako scriptaren moldaketa agertzen da. Bertan, parametro gisa jasotzen diren katalogoetan bilatzen dira izen bereko fitxategiak, eta izen errepikatuak pantailaratzen dira.

Horretarako erabilgarri gertatzen da *os.walk* metodoa, zeinaren bidez katalogo baten azpian dauden azpikatalogo bakoitzeko (bera barne) azpikatalogo eta fitxategi guztiak itzultzen dituen. Horrela hirukoteen zerrenda bat itzultzen du, hirukote bat azpikatalogo bakoitzeko, hirukotearen elementuak hauek izanik: (azpi)katalogoaren bidea, azpian dituen azpikatalogoak (zerrenda) eta azpian dituen fitxategiak (zerrenda).

```
import os
import sys

izenak = [] # izen sinpleak biltzeko
fitxategiak = [] # izen osoak biltzeko

def bilatu_fitx(dir):
    # katalogoaren fitx-ak os.walk bidez
    l = []; l2 = []
    for erro, dir_ak, fitx_ak in os.walk(dir):
        for fitx in fitx_ak:
            l.append(fitx) # izen sinpleak
            l2.append(erro+"/"+fitx) # izen osoak
    return (l, l2)
# itzuli izen sinpleak eta osoak

for dir_i in sys.argv: # argumentuak
    print("Fitxategien zerrenda sortzen...", dir_i)
    izenak = izenak+bilatu_fitx(dir_i)[0] # 0-sinpleak
    fitxategiak = fitxategiak+bilatu_fitx(dir_i)[1] # 1-osok

print("Bikoiztuak detektatzen (" +str(len(izenak)) +",fitx)...")
for izen in izenak:
    n = izenak.count(izen)
    if n > 1:
        print("-"*60, "\n> bikoiztuta:", izen, n, "\n")
        for item in fitxategiak:
            if item.endswith("/"+izen): # endswith metodoa bukaerarako
                print(item)
```

9.7. programa. Fitxategi bikoiztuak bilatzeko scripta.

Programa probatzeko onena da bi azpikatalogo sortzea (*dir1* eta *dir2*) eta bakoitzean hainbat fitxategi kopiatzea. Deitzean bi katalogoen izenak parametro gisa pasako dira, eta fitxategiren bat bi katalogoetan baldin badago, detektatuko da.

9.3. SISTEMA-DEIAK

Esan bezala sistema-deiak SEaren oinarrizko funtzioak dira. Python sistemen programazioari eta administrazioari begira lengoaia egokia da, beste gauzen artean, sistema-deiak zuzenean erabiltzea bideratzen duelako.

Sistema-deien bidez sistemaren baliabideak zuzenean erabil daitezke. Adibidez:

- prozesuen informazioa eskuratzea eta aldatzea: *getpid*, *getuid*, *seteuid*...
- prozesuak sortzea, exekutatzea, itxoitea eta hiltzea: *fork*, *exec*-en aldaerak (*execl*, *execvp*...), *wait*, *kill*
- komunikazioak: *pipe*, *mkfifo*...
- fitxategiak (zuzenean kanal-zenbakiaren bidez): *open*, *read*, *write*, *lseek*, *dup*...

Hemen ez dugu tokirik sistema-deien inguruko kontzeptuetan sakontzeko, baina aipatutako gunean horren inguruko informazio ugari aurki daiteke. Oro har, sistema-dei bakoitzari metodo bat dagokio.

9.8. adibidean sistemen programazioan ohikoak diren bi eragiketa (metodoak Pythonez) erabiltzen dira: prozesu umea sortzea (*fork*) eta komunikazio-kanala sortzea (*pipe*). Adibidean sortzen den kanalaren bidez umeak idazten duena gurasoak irakurtzen du eta pantailaratzen du.

```
import os, sys, time

# izen gabe FIFOa (pipe) sortzea
ir,id=os.pipe() # bi kanal itzultzen ditu, irakurtzeko eta idazteko

# 2 fitxategi
blokel=1024
r=os.fdopen(ir,'r',blokel) # irakurketarako ireki
w=os.fdopen(id,'w',blokel) # idazketarako ireki

pid = os.fork()
if pid: # gurasoa (irakurlea)
    w.close()
    while 1:
        data=r.readline() # FIFOtik irakurri
        if not data: break
        print("gurasoa read: ", data.strip()) # inprimatu pantailan
else: # umea (idazlea)
    r.close()
    for i in range(10):
        w.write("lerroa: " + str(i) + "\n") # FIFOa idatz
        w.flush() # memoriatik S/Ira
        time.sleep(1) # segundo bat itxoin
```

9.8. programa. FIFO batez komunikatzea.

9.4. MULTITHREADING

Multiprozesadoreen aukeraz baliatzeko eta aplikazioak azkartzeko hari anitzeko edo *multithreading* izeneko mekanismoa erabiltzen da. Hariak (*thread*) exekuzio-fluxuak dira, prozesuaren memoria eta fitxategiak partekatzen dituztenak, baina prozesadoreari (edo prozesadoreei) begira lehiakideak direnak. Gaur egungo programazio konkurrente eta paraleloaren funtsa dira, eta programazio-lengoaiei esku dago haiei etekina ateratzea (beti azpiko sistema eragileak uzten duenean).

Pythonen kasuan alde handi samarra dago ezaugarri honi begira Python2.4 bertsiotik aurrera. Lehenago *thread* moduluari dagozkion metodoak erabiltzen ziren helburu horrekin. Ahalmen mugatua du modulu honek, adibidez prozesu umeen zain geratzeko. Python2.4 bertsiotik aurrera muga horiek gainditu dira *threading* modulua bidez, eta, gainera, objektuei orientatutako programazioari eman zaio lehentasuna. Hala ere, aurreko modulua metodoak ere erabil daitezke, baina Python3 erabiltzen bada, aldaketatxo bat gertatu da: *_thread* izena zehaztu behar da modulu hori erabiltzeko. Laburbilduz, Python3-n bi modulu erabili ahal izango dira, *_thread* izenekoa, sinplea baina mugatua; eta *threading* izenekoa, osatuagoa eta objektuei orientatua.

Programazioari dagokionez, hari bakoitzari funtzio bat edo metodo bat dago-kio. Programaren arabera hariak funtzio/metodo bera erabiliko dute (nagusi-morroiei erdua deitu ohi denean morroiekin egiten den bezala) edo hari bakoitzak metodo/funtzio berezi bat edukiko du (*pipe* erdua da horren adibidea). Bi kasuetan funtzioak/metodoak definitu eta gero, programa nagusia harien sorreraz arduratuko da.

Adibide sinple batekin hasiko gara. Bi hari sortuko ditugu eta bakoitzak ordua inprimatuko du hiru aldiz denbora-tarte bat itxaron eta gero. Hori programatzeko bi aukera ditugu: bertsio zaharretan erabiltzen zen *thread* klaseko *start_new_thread* metodoa (*p9-9a.py*); eta *threading* klaseko *Thread* metodoa (*p9-9b.py*).

Lehen kasuan, programa nagusia zain geratzen da etengabe, bestela bukatuko litzateke hariak amaiaraziz. Beraz, probatzean CTRL-C bidez amaiarazi beharko duzu.

```
import _thread as thread
import time

# hariari dagokion funtzioa
def inpri_denbora(haria, atzerapena):
    kontag = 0
    while kontag < 3:          # 3 aldiz
        time.sleep(atzerapena) # itxotea (segundotan)
        kontag = kontag + 1
    print(haria, "- ordua:", time.ctime(time.time()))
```

```
# programa nagusia: bi hari sortuko duena
try:
    thread.start_new_thread(inpri_denbora, ("1. haria", 2,))
    thread.start_new_thread(inpri_denbora, ("2. haria", 4,))
except:
    print("Errorea: ezin haria sortu")

# itxoitea harien bukaera arte
while 1:
    pass
```

9.9a. programa. Bi hari, bakoitzak bere ordua idazten.

Bigarren kasuan, gomendagarriena, programa nagusian ez da itxaron behar; baina hari bakoitzeko bi urrats egiten dira: hariaren definizioa *Thread* metodoaren bidez, eta haria martxan jartzea *start* metodoaz.

```
from threading import Thread
import time

# hariari dagokion funtzioa
def inpri_denbora(haria, atzerapena):
    kontag = 0
    while kontag < 3:
        time.sleep(atzerapena)    # itxoitea (segundotan)
        kontag = kontag + 1
        print(haria, "- ordua:", time.ctime(time.time()))

# programa nagusia: bi hari sortuko duena
try:
    t1=Thread(target=inpri_denbora, args=("1. haria", 2,))
    t1.start()
    t2=Thread(target=inpri_denbora, args=("2. haria", 4,))
    t2.start()
except:
    print("Errorea: ezin haria sortu")
```

9.9b. programa. Bi hari, bakoitzak bere ordua idazten.

threading modulua oso ahalsua da, baina konplexua ere. Oinarrizko metodoak hauek dira:

- *run()*: Hariaren sarrera-puntua definitzeko objektuetan.
- *start()*: Haria abiaraztea *run* metodoa erabiliz (objektuetan) edo funtzio batez.
- *join([time])*: Harien zain geratzea.
- *isAlive()*: Haria indarrean dagoen ala ez galdetzea.
- *getName()*: Hariaren izena lortzea.
- *setName()*: Hariaren izena ezartzea.

Aurreko moduluaren metodoez gain, beste batzuk ditugu eskuragarri *threading* moduluari:

- *threading.activeCount()*: Sortutako hari aktiboen kopurua itzultzen du.
- *threading.currentThread()*: Hari deitzaileari dagozkion harien kopurua itzultzen du.
- *threading.enumerate()*: Indarrean dauden harien izenak itzultzen ditu, zerrenda batean.

Gainera, klase-herentzia bidez klase berriak defini daitezke, objektuei orientatutako metodologiari jarraituz. Horrela 9.9b. adibidea eratorritako klase batez ebazten da 9.9c. programan. Bertan, *DenbHaria* klasea definitzen da, *threading.Thread* klasetik eratorrita, beraz, haren objektuei klasearen metodo guztiak aplikatu dakizkieke. Gure kasuan *run* metodoa birdefinitu da eta *start* metodoa aplikatzen da hura aktibatzeko.

```
import threading
import time

class DenbHaria (threading.Thread):
    def __init__(self, threadID, izen, kont):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.izen = izen
        self.kont = kont
    def run(self):
        inpri_denbora(self.izen, self.kont)

def inpri_denbora(haria, atzerapena):
    kontag = 0
    while kontag < 3:
        time.sleep(atzerapena)
        kontag = kontag + 1
        print(haria, "- ordua:", time.ctime(time.time()))

try:
    # Hariak sortzea
    t1 = DenbHaria(1, "Thread-1", 2)
    t2 = DenbHaria(2, "Thread-2", 4)
    # Hariak abiatzea
    t1.start()
    t2.start()
except:
    print("Errorea: ezin haria sortu")
```

9.9c. programa. Hariak klase eratorri gisa.

Aurreko programa nahiko sinplea da, baina ez da oso praktikoa. Praktikotasunaren bila, 9.10. programa harien bidez programatuko dugu (*threading* modulua erabiliz). Zazpigarren kapituluaren egindako 7.9. programaren egokitzapena da. Bertan hainbat URLtatik irakurtzen da, eta horretarako hari bat sortzen da URL bakoitzeko. Irakurritakoa inprimatzen da HTML etiketak kenduta, eta horretarako erabiltzen da instalatu dugun *html2text* modulua.

```
import sys
import threading
from urllib.request import urlopen
from html2text import html2text

# 7.9 programaren egokitzapena

# hariari dagokion funtzioa
def eskuratu_url(url):
    print(url)
    orri=urlopen(url)
    for lerro in orri:
        lerro = lerro.decode('utf-8')
        txt = html2text(lerro) # html etiketak kentzeko
        if len(txt)>20:        # 20 karaktere baino gehiago
            print(txt)

urlak = ["http://sustatu.eus", "http://ueu.eus"]

for u in urlak:
    t = threading.Thread(target=eskuratu_url, args = (u,))
    t.start()
```

9.10. programa. Webguneak irakurtzen hari desberdinetatik.

Liburu honetan ez dugu asko sakonduko gai honetan. Adibidez ez gara sartuko sinkronizazio eta komunikaziorako dauden klase eta metodoetan²⁴.

Adibide konplexuago bat helbide honetan kontsulta daiteke: <<http://www.toptal.com/python/beginners-guide-to-concurrency-and-parallelism-in-python>>.

9.5. ADIBIDEAK

9.5.1. Enuntziatua: *grep* paraleloa

Katalogo baten fitxategi guztietan testu baten agerpena bilatzen duen programa egin behar da, baina fitxategi bakoitzean bilaketa hari batez egingo da. Hari bakoitzaren kodea sinplea izango da, *grep* komandoa erabiltzea baino ez. Komando horrek 0 itzuliko du, karaktere-katea aurkitu badu. Komandoaren irteera

24. Edozein kasutan <http://www.tutorialspoint.com/python/python_multithreading.htm> helbidean informazioa osa daiteke (<<https://docs.python.org/3/library/threading.html>> Python3-ko zehaztasunetarako).

pantailatik ager ez dadin, */dev/null* dispositibo nulura birbideratuko da. Objektuen orientazioarekin egingo dugu, eta horretarako 9.9c. programan oinarrituko gara.

```
import threading
import os
import sys

class BilaFitxHaria (threading.Thread):
    def __init__(self, fitx, st):
        threading.Thread.__init__(self)
        self.fitx = fitx
        self.st = st
    def run(self):
        bila_fitx(self.fitx, self.st)

def bila_fitx(f, kate):
    em = os.system("grep " + kate + " " + f + " >/dev/null")
    if not em:
        print(f)

try:
    for f in os.listdir(sys.argv[1]): # katalogoko fitxategi bakoitzeko
        if(os.path.isfile(f)):      # fitxategiekin bakarrik
            t = BilaFitxHaria(f,sys.argv[2])
            t.start()
except:
    print("Errorea: ezin haria sortu")
```

9.11. programa. *grep* bidezko bilaketa konkurrentea.

Erabiltzeko bi parametro pasa behar dira: katalogoa eta bilatu nahi den karaktere-katea. Bilaketa katalogoko fitxategietan egiten da azpikatalogoetan sartu gabe. Azpikatalogoetan sakontzeko *os.walk* metodoa erabil daiteke.

9.5.2. Enuntziatua: *FIFO*ak harien bidez

9.8. programan azaldutako *FIFO*aren adibidea harien bidez ebatzi behar da, prozesuen bidez ebatzi beharrean. Horretarako, bi funtzio definituko dira: irakurlea bata, idazlea bestea. *Thread* metodoaren bidez abiatuko dira. Sistema-dei gehienak (prozesuak sortzearenak kenduta) berdin geratzen dira.

```
# FIFO batez komunikatzea hariak erabiliz

import os, sys, time
import threading

def irakurle(irf):
    print("...irakurtzen")
    while 1:
        data=irf.readline()    # FIFOtik irakurri
        print("irakurle read: ", data.strip()) # inprimatu pantailan
    return
```

```

def idazle(idf):
    global harikop
    print("...idazten")
    for i in range(10):
        idf.write("lerroa: " + str(i) + "\n") # FIFOa idatzi
        idf.flush()                          # memoriatik S/Ira
        time.sleep(1)                        # segundo bat itxoin
    return

# izen gabe FIFOa (pipe) sortzea
ir,id=os.pipe() # bi kanal, bat irakurtzeko eta bestea idazteko

# 2 fitxategi
blokel=1024
ird=os.fdopen(ir,'r',blokel) # irakurketarako ireki
idd=os.fdopen(id,'w',blokel) # idazketarako ireki

print("Hasiera")
t1=threading.Thread(target=irakurle,args=(ird,))
t1.start()
print("irakurlea martxan")
t2=threading.Thread(target=idazle,args=(idd,))
t2.start()
print("idazlea martxan")

```

9.12. programa. FIFO batez komunikatzea hariak erabiliz.

9.6. PROPOSATUTAKO ARIKETAK

1) Aldatu 9.6. programa CPUren portzentajeak metatzeaz gain, memoriakoak ere metatzeko. Gainera, emaitzek erabileraren arabera sailkatuta agertu behar dute (*sort* komandoa erabil daiteke horretarako, baina Pythonen klaseak ere).

2) Aldatu 9.10. programa, honako helburuak lortzeko:

- Irakurri behar diren web orrien helbideak fitxategi batean egotea.
- Hariak klase eratorri gisa definitzea.
- Irakurtzen diren orrietatik testua lortu ondoren, inprimatu beharrean hiztegia osatu eta maiztasunak kalkulatzeko. Horretarako hari guztien informazioa jaso beharko da.

3) Egin Python programa bat bi matrizeren arteko biderketa egiteko harien bitartez. Hari bakoitzaren funtzioa izango da errenkada eta zutabe bat biderkatzea eta emaitzaren elementu bat lortzea. Bukaeran emaitza inprimatuko da. Oinarri gisa 6.4. programa har daiteke.

10. Aplikazioak I: Internet

Kapitulu honetan Interneteko baliabideak lantzen ikasiko dugu. Alde batetik hiru ariketa proposatzen ditugu HTML orriekin lan egiten ikasteko, hiruretan *BeautifulSoup* liburutegia erabiliz. Estekak bilatuko ditugu, orrien artean ibiliko gara eta web orri berri bat sortuko dugu. Horrez gain, azken ariketan Twitterrekin lan egingo dugu. Kasu horretan, txioen zuzeneko jarraipena nola egiten den ikasiko dugu *tweepy* liburutegiaz. Ikusten denez, eta aurreko kapituluetan egindakoaren ildotik, atazetarako egokiak diren moduluak berrerabiltzea da gaur egungo programazioaren gakoetako bat.

10.1. ARIKETA: WIKINEWS-ETIK BERRIEN IZENBURUAK ERAUZTEA

Ariketa honetan *Wikinews*-eko Europako portadan (<https://en.wikinews.org/wiki/Portal:Europe>) agertzen diren berriak eta haien tituluak jasoko ditugu. Hasi baino lehen, web orriaren HTML kodea aztertzea ezinbestekoa da, informazioa non eta nola gordetzen den aztertzeko. Horretarako, web orriaren iturburu-kodea atzi daiteke nabigatzailetik²⁵. Kontuan izan behar da web orri horren iturburu-kodea etengabe aldatzen doala, eta hortaz, baliteke behin ondo funtzionatzen duen kodeak, hortik eta denbora batera funtzionatzeari uztea.

Ariketa honen kasuan, *Wikinews*-en iturburu-kodea aztertuko dugu, berrien estekak eta izenburuak non aurkitzen diren aztertzeko, gero izenburuak pantailaratzeko.

Web orriari gainetik begiratuta (ikus beheko kode-zatia), tituluak buletekin (** elementuak) agertzen direla ikus dezakegu erraz asko.

```
...
<td class="lead_normal upper_lead">
<div style="padding-top: 5px"></div>
<p>&#160; <b>Politics and conflicts</b></p>
<ul>
<li><a href="/wiki/Over_270_civilians_reported_killed_from_shelling_in_Syria"
title="Over 270 civilians reported killed from shelling in Syria">Over 270 civilians
reported killed from shelling in Syria</a></li>
```

25. Mozilla-Firefox nabigatzailearekin, adib, *Ctrl-u* tekla-konbinazioarekin ikus daiteke.

```

<li><a href="/wiki/Lord_Howard_and_Alistair_Darling_address_Confederation_of_British_Industry_on_EU_referendum" title="Lord Howard and Alistair Darling address Confederation of British Industry on EU referendum">Lord Howard and Alistair Darling address Confederation of British Industry on EU referendum</a></li>
<li><a href="/wiki/Telegraph_publishes_letter_from_300_business_leaders_who_back_UK_leaving_EU" title="Telegraph publishes letter from 300 business leaders who back UK leaving EU">Telegraph publishes letter from 300 business leaders who back UK leaving EU</a></li>
<li><a href="/wiki/IMF_says_UK_leaving_the_EU_will_lead_to_negative_economic_consequences" title="IMF says UK leaving the EU will lead to negative economic consequences">IMF says UK leaving the EU will lead to negative economic consequences</a></li>
</ul>
</td>
...

```

Gehiago sakonduz, estekaren elementua (<a> elementua), buleta-elementu baten azpitik agertzen dela baieztatu dezakegu. Gainera, esteka-elementuek *title* atributua definituta dute, berriaren izenburua barneratzen duena. Hori horrela izanik, script bat egingo dugu zeinak web orria atzitu eta bertatik *title* atributua duten esteka-elementu guztiak jasoko dituen, eta banan banan begiratuko dugu ea buleta elementu bati lotuta ote dagoen. Baiezko kasuetan, berri baten esteka dela baieztatuko dugu eta haren izenburua pantailaratuko dugu.

Ariketa honetarako 7.3. atalean ikusitako modulua berrerabiliko dugu web orria atzitzeko: *urllib.request*. Modulu horrez gain, *BeautifulSoup* modulua (*bs4* izena erabiltzen da inportazioan) erabiliko dugu HTML egituraren barruan elementuak identifikatzeko. Modulu horrek HTML orria zuhaitz bilakatzen du, informazioa erauztea asko errazten duena. *BeautifulSoup* oso ahaltsua da eta gaizki eratutako HTML orriekin ere lan egitea ahalbidetzen du.

Modulu hori erabiltzeko, aurrena instalatu beharra dago, ez baitator Pythonen aurreinstalatuta. Horretarako, nahikoa da *pip3* bidez instalatzea honako komando honen bidez.

```
$ sudo pip3 install bs4
```

10.1. programak *Wikinews*-eko web orritik berrietara doazen estekak bilatzen ditu, eta beraien izenburuak pantailaratzen ditu.

Horretarako, aurrena web orria irekitzen da eta zuhaitz bilakatzen du *BeautifulSoup* objektuan *lxml* analizatzailearen bitartez analizatu ondoren.

```

orria = urlopen(sys.argv[1])
soup = BeautifulSoup(orria, "lxml")

```

Ostean, esteka-elementuak diren elementu guztiak jasotzen dira *findall* metodoaren bidez (*lotura_guztiak* aldagaian), eta begizta baten bidez elementu horien gurasoa egiaztatzen da ea buleta-elementua den, baiezko kasuan pantailaratuz.

```
import sys
#weborria atzitzeko erabiliko den liburutegia inportatu
from urllib.request import urlopen
#informazioa analizatzeko BeautifulSoup funtzioak jaso
from bs4 import BeautifulSoup

#web-orria atzitu eta html-a orria aldagaian gorde
orria = urlopen(sys.argv[1])
#orria aldagaieko html-a parseatu eta BeautifulSoup formatuan gorde
soup = BeautifulSoup(orria,"xml")

#link guztiak aurkitu:
lotura_guztiak = soup.find_all("a",{ 'title':True})
#link bakoitzerako href aldagaiaren edukia jaso
for lotura in lotura_guztiak:
    if lotura.parent.name == 'li':
        print(lotura["title"])
```

10.1. programa. Web orri batetik estekak lortzeko adibidea *Beautiful Soap* modulua erabiliz.

Aurreko programa egikaritzean:

```
python3 p10-1.py https://en.wikinews.org/wiki/Portal:Europe
```

Modu honetako irteera jasotzen da:

```
Over 270 civilians reported killed from shelling in Syria
Lord Howard and Alistair Darling address Confederation of British Industry on EU
referendum
Telegraph publishes letter from 300 business leaders who back UK leaving EU
IMF says UK leaving the EU will lead to negative economic consequences
Lawsuit filed against Ed Sheeran for his single Photograph
...
```

BeautifulSoup moduluaren inguruan informazio zehatza eskuratzeko <<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>> gunea kontsulta daiteke.

10.2. ARIKETA: EUSKAL HERRIKO FUTBOL-SELEKZIOKO JOKALARIEN IBILBIDEA ERAUZTEA WIKIPEDIATIK

Aurreko ariketa oinarri harturik, Wikipediaren Euskal Herriko futbol-selekzioaren web orritik jokalariai jaso nahi ditugu oraingoan. Baina jokalarien izenak jasotzeaz gain, beraiei buruzko informazio gehiago jaso nahi dugu, beraien Wikipediako sarrerako informazio-kutxetatik (*info-box*).

Jaso nahi dugun informazioa futbolariaren ibilbidea izango da, hau da, futbolariak jokaturako talde guztiak aipatu nahi ditugu, bertan jokaturako urteak eta, informaziorik izanez gero, jokaturako partida kopurua eta sartutako golak. Informazio hori hiztegi batean jasoko dugu.

Web orri batetik informazioa erauzi nahi dugunez, aurrena web orriaren iturburu-kodea aztertu behar dugu, jaso nahi dugun informazioa non dagoen eta nola egituratzen den jakin ahal izateko. Jarraian <https://eu.wikipedia.org/wiki/Euskal_Herriko_futbol_selekzioa> orriaren iturburu-kodearen zati bat erakusten dugu azalduko duguna ulertzen laguntzeko.

```
...
<h2><span class="mw-headline" id="Jokalariaipagarriak">Jokalariaipagarriak</span>...</h2>
<div style=...>...</div>
<p>Hona hemen Euskal Herriko futbol selekzioan lau bider edo gehiagotan aritutako jokalarien zerrenda, <a href="/wiki/2011" title="2011">2011ko</a> <a href="/wiki/Abenduaren_28" title="Abenduaren 28">abenduaren 28an</a>:</p>
<table border="0" cellpadding="2">
<tr valign="top">
<td>
<p><b>12</b></p>
<ul>
<li><a href="/wiki/Julen_Guerrero" title="Julen Guerrero">Julen Guerrero</a></li>
<li><a href="/wiki/Igor_Gabilondo" title="Igor Gabilondo">Igor Gabilondo</a></li>
...

```

Kasu honetan, "jokalariaipagarriak" ataleko jokalariai jaso nahi ditugu. Iturburu-kodea aztertzen badugu, Wikipediako orrien atalak elementuaren klase batetik bereizten direla kontura gaitezke, mw-headline klasearekin hain zuzen ere, eta elementu horren bitartez interesatzen zaigun atala identifika dezakegu ("Jokalariaipagarriak").

Kodea aztertzen jarraituta, interesatzen zaizkigun esteka-elementu guztiak taula beraren barruan daudela ohar gaitezke, atalaren elementuaren jarraian dagoen taula alegia. Kontuan izan, ez dagoela elementuaren ume batean, hortaz dagokion maila egokia bilatu behar da. Dena dela, esteka guztiak taula berean daudela ikusteak asko errazten dizkigu gauzak.

"Jokalariaipagarriak" identifikadorea duen elementua bilatuko dugu aurrena, find metodoaren bidez, eta haren gurasoa den <h2> elementua jasoko dugu gero parent metodoaren bidez:

```
jokalaria_atala = soup.find("span", {"id": "Jokalariaipagarriak"})
gurasoa = jokalaria_atala.parent # h2 elementua
```

Azken horren maila berean dago `<table>` elementua, hortaz, begizta baten bidez banan-banan `<table>` elementua bilatuko dugu, eta behin aurkituta, haren azpian dauden esteka-elementu guztiak erauziko ditugu *nextSibling* metodoaren bidez.

```
while hurrengo and hurrengo.name != "table":
    hurrengo = hurrengo.nextSibling
if hurrengo:
    taula = hurrengo
```

Estekaren helbidea *href* atributuaren baitan gordetzen da HTML kodean, baina Wikipediaren kasuan, esteka partzialak dira (ez dute URL osoa aipatzen), eta euskarazko esteken kasuan `<https://eu.wikipedia.org>` helbidea atxikitu behar zaie hasieran.

```
for jokalaria_esteka in taula.find_all("a"):
    esteka = "https://eu.wikipedia.org"+jokalaria_esteka["href"]
```

Horrez gain, kontuan izan beharra dugu jokalaria batzuen Wikipedia orria sortu gabe dagoela. Hori esteka-elementuaren atributu berezi baten bidez adierazten da, *class="new"* atributu-balioaren bidez. Ondoko kodean ezaugarri hori duten jokalariekin ondoren ezer ez egitea (*continue*²⁶) adierazten da

```
if "class" in jokalaria_esteka:
    continue
```

Aurreko guztia kontuan izanik, eta kodea ulergarriagoa eta egituratuagoa izan dadin, funtzio berri bat definituko dugu esteka batetik beharrezko informazioa erauziko duena. Funtzio horri *futbolariaren_informazioa_erauzi* deituko diogu. Funtzioak beste web orri batetik informazioa erauziko duenez, aurreko kodea idazterakoan bezala, web orria ondo aztertu behar dugu. Ondoren interesgarria den zati bat azaltzen da:

```
<div id="mw-content-text" lang="eu" dir="ltr" class="mw-content-ltr">
<table class="toccolours" ...>
<caption><big><b>Julen Guerrero</b></big></caption>
...
<tr>
<th ...>Futbolari ibilbidea</th>
</tr> <tr>
<td><b>Urteak</b></td>
<td><b>Taldea</b></td>
<td><b>Part. (golak)</b></td>
```

26. *continue* bidez begizta batean iterazio baten bukaera adierazten da (ikus 4. kapitulua).

```

</tr> <tr>
<td>1991-1992<br /> 1992-2006</td>
<td><a href="/wiki/Bilbao_Athletic" title="Bilbao Athletic">Bilbao Athletic</a><br
/> <a href="/wiki/Bilboko_Athletic_Kluba" title="Bilboko Athletic Kluba">Athletic</
a></td>
<td>? (?)<br /> 372 (101)</td>
</tr> </table>

```

Kasu honetan, `<div>` elementu baten barruan *mw-content-text* identifikadorea duen elementuaren lehen taulan informazio-kutxako (*info-box*) informazioa agertzen da. Ariketa honetarako, informazio hori atzituko dugu *find* metodoaren bidez (bi urratsetan):

```

div = soup_jokalari.find("div", {"id": "mw-content-text"})
info_box = div.find("table")

```

Ondoren kutxa egituraren baitan dagoen "Futbolari ibilbidea" hiztegiratuko dugu. Horretarako, taulan bertan dauden zutabe-goiburukoetan (`<th>` elementua) bilatu beharko dugu "Futbolari ibilbidea", jarraian horren atzetik datorren errenkada jasotzeko.

```

jokalari_ibilbidea = info_box.find("th", text = "Futbolari ibilbidea")
gurasoa = jokalari_ibilbidea.parent
#Lau elementu aurrera egin behar ditugu egituraren arabera
taldeak = gurasoa.nextSibling.nextSibling.nextSibling.nextSibling

```

Hurrengo errenkada (`<th>` beste elementu bat) horretan zutabe ezberdinetan (`<td>` elementuak) datoz urteak, klubak eta partida eta gol kopuruak hurrenez hurren. Sarrera bakoitza taula baten errenkadetan bereizita etorri beharrean, lerro-jauziaz bereizita dator lehenengo errenkada horren barruan. Beraz, dagokion kodea hau da:

```

eremuak = taldeak.find_all("td")
urteak = eremuak[0].text.split("\n")
klubak = eremuak[1].text.split("\n")
kopurua = len(urteak)

```

Jokalari batzuen kasuan ez da partida eta gol kopuruaren errenkada aurkitzen, eta, ondorioz, hutsa dagoen kasuan, beste datuen elementu kopuru bera duen zerrenda huts bat sortuko dugu prozesaketa automatikoa errazteko:

```

kopurua = len(urteak)
if len(eremuak) == 2:
    partiduak = [""] * kopurua
else:
    partiduak = eremuak[2].text.split("\n")

```


Informazio hori guztia zerrenden zerrenda batean sartuko dugu, eta hori izango da funtzioak itzuliko duen balioa:

```
futbolariaren_informazioa = []
for i in range(0,kopurua):
    urtea = [urteak[i],klubak[i],partiduak[i]]
    futbolariaren_informazioa.append(urtea)
```

Jokalari batzuek, web orria izan arren, informazio-kutxarik ez daukate edota "Futbolariaren informazioa" atala hutsik daukate. Informazio falta duten web orriak baztertzeko, salbuespen bat kudeatzeko kodea sartuko dugu (*try/except* bidez) errorerik gerta ez dadin, eta salbuespenen kasuan hutsa (*None*) itzuliko dugu.

10.2. programan orain arte azalduko guztia bildu dugu.

```
import sys
from urllib.request import urlopen
from bs4 import BeautifulSoup

def informazioa_erauzi(esteka):
    jokalari_orria = urlopen(esteka)
    soup_jokalari = BeautifulSoup(jokalari_orria,"lxml")
    try:
        div = soup_jokalari.find("div",{ "class": "mw-content-ltr" })
        info_box = div.find("table")
        jokalari_ibilbidea = info_box.find("th",text = "Futbolari ibilbidea")
        gurasoa = jokalari_ibilbidea.parent
        #Lau elementu aurrera egin behar ditugu egituraren arabera
        taldeak = gurasoa.nextSibling.nextSibling.nextSibling.nextSibling
        eremuak = taldeak.find_all("td")
        urteak = eremuak[0].text.split("\n")
        klubak = eremuak[1].text.split("\n")
        kopurua = len(urteak)
        if len(eremuak) == 2:
            partiduak = [""]*kopurua
        else:
            partiduak = eremuak[2].text.split("\n")
        futbolariaren_informazioa = []
        for i in range(0,kopurua):
            urtea = [urteak[i],klubak[i],partiduak[i]]
            futbolariaren_informazioa.append(urtea)
        return futbolariaren_informazioa
    except AttributeError:
        return None

orria = urlopen(sys.argv[1])
soup = BeautifulSoup(orria,"lxml")
#Jokalari aipagarriak atala aurkitu
jokalari_atala = soup.find("span",{ "id": "Jokalari_aipagarriak" })
gurasoa = jokalari_atala.parent # h2 elementua
hurrengo = gurasoa.nextSibling

while hurrengo and hurrengo.name != "table":
    hurrengo = hurrengo.nextSibling
hiztegia = {}
```

```
import sys
from urllib.request import urlopen

if hurrengoa:
    taula = hurrengoa
    for jokalaria_esteka in taula.find_all("a"):
        if "class" in jokalaria_esteka:
            continue
        esteka = "https://eu.wikipedia.org"+jokalaria_esteka["href"]
        hiztegia[jokalaria_esteka["title"]] = informazioa_erauzi(esteka)
```

10.2. programa. Euskal Herriko futbol-selekzioko jokalarien ibilbidea eraztea Wikipediatik.

10.3. ARIKETA: JOKALARIEN IBILBIDEAREN WEB ORRIA SORTZEA

Aurreko ariketan Euskal Herriko futbol-selekzioko jokalarien ibilbidea hiztegi batean jaso dugu. Oraingoan hiztegi horretatik erauzitako informazioarekin web orri simple bat egingo dugu.

Ataza erraza da oraingoan, HTML orri bat sortuko dugu *BeautifulSoup* liburutegia erabilita, eta horretarako hiztegian gordeta ditugun jokalariekin eta beraien ibilbidearen informazioarekin taula bat sortuko dugu.

Kontuan hartzeko lehen zehaztasuna: Wikipediak UTF-8 kodeketari jarraitzen dio bere web orrietan, eta, horrela, gure web orri berria ondo ikusarazteko web orriaren burukoan (<head> elementuaren baitan) <meta> elementuan UTF-8 kodeketa zehaztu beharko dugu. Hori egingo ez bagenu, web orria karaktere arraroekin ikusiko genuke, tilde eta ñ letretan.

Adibidez, UTF kodeketarako honako lerro hau idatzi dugu:

```
meta = soup_irteera.new_tag("meta", charset="UTF-8")
```

Bestalde, *BeautifulSoup* bidez HTML orri bat sortzeko, elementuak banan-banan sortu behar ditugu *new_tag* metodoaren bitartez. Metodo horretan elementu berriaren izena zehaztuko dugu, eta atributurik jarri nahiko bagenu, atributuaren izenari bere balioa esleituko genioke.

HTML kodean ez dugu sakonduko, baina kodea ondo ulertzeko gogora dezagun taulen inguruko zenbait zehaztasun: errenkadak *"tr"* etiketaren bidez adierazten dira eta zutabeak *"td"* etiketaren bidez, *"th"* etiketa berezia dago taulen goiburuak adierazteko.

Beraz, aurrena errenkada batean (*"tr"*) goiburukoak idatziko ditugu (*"th"*): "Jokalaria", "Urteak", "Kluba" eta "Part. (Gol)". Ostean, jokalarien ibilbideko errenkada bakoitzerako *"tr"* elementu bat sortuko dugu, eta ostean *"td"* elementu bana jokalaria izenerako, urteetarako, klubetarako eta partida kopuruetarako.

10.3. programan objektu horiek sortuz web orria osatu dugu, aurretik lortutako hiztegiko edukia sartuz taula batean. Programak ondo funtzionatu ahal izateko, 10.2. programari bukaeran gehitu behar diogu kodea.

```
soup_irteera = BeautifulSoup("", "lxml")
html = soup_irteera.new_tag("html")
head = soup_irteera.new_tag("head")
meta = soup_irteera.new_tag("meta", charset="UTF-8")
table = soup_irteera.new_tag("table")
tr = soup_irteera.new_tag("tr")
soup_irteera.append(html)
html.append(head)
html.append(table)
table.append(tr)
head.append(meta)
for izenburu in ["Jokalaria", "Urteak", "Kluba", "Part. (Gol)"]:
    th = soup_irteera.new_tag("th")
    tr.append(th)
    th.append(izenburu)
for jokalaria, ibilbidea in hiztegia.items():
    if not ibilbidea:
        continue
    for errenkada in ibilbidea:
        tr = soup_irteera.new_tag("tr")
        th.append(tr)
        td_j = soup_irteera.new_tag("td")
        tr.append(td_j)
        td_j.append(jokalaria)
        td_u = soup_irteera.new_tag("td")
        tr.append(td_u)
        td_u.append(errenkada[0])
        td_k = soup_irteera.new_tag("td")
        tr.append(td_k)
        td_k.append(errenkada[1])
        td_p = soup_irteera.new_tag("td")
        tr.append(td_p)
        td_p.append(errenkada[2])
with open(sys.argv[2], 'w') as fout:
    fout.write(soup_irteera.prettify())
```

10.3. programa. HTML taula bat sortzea hiztegi batetik.

10.3. programa exekutatuz gero, HTML orri batean idatziko da HTML kodea *prettify* metodoaren bidez. Horrela, lerro-jauziak eta tabulazioak gehituko dizkiogu HTML kodeari, irakurterazagoa izan dadin, baina horrek ez du eraginik izango nabigatzailean ikusiko den web orrian. Azkenean HTML orri hori fitxategi batean idazten da (2. parametroak zehazten duen izenarekin).

Probatzeko honako komando hau erabiliko dugu:

```
python3 p10-3.py https://eu.wikipedia.org/wiki/Euskal_Herriko_futbol_selektzioa
jokalariaiak.html
```

Bertan bigarren parametro gisa sortuko den fitxategiaren izena agertzen da.

jokalariai.html fitxategia nabigatzaile batekin irekiz gero, 10.1. irudikoa ikus dezakegu. Kontuan izan, hiztegi baten ordena ez dela finkoa, eta programaren exekuziotik exekuziora ordena alda daitekeela.

Jokalaria	Urteak	Kluba	Part. (Gol)
Jon Pérez Bolo		Danok Bat	
Jon Pérez Bolo		Athletic (harrobia)	
Jon Pérez Bolo	1994-1996	Bilbao Athletic	52 (14)
Jon Pérez Bolo	1994-1998	Athletic Club	41 (4)
Jon Pérez Bolo	1997	Osasuna (utzita)	7 (0)
Jon Pérez Bolo	1997-1998	Hércules CF (utzita)	26 (6)
Jon Pérez Bolo	1998	Athletic Club	1 (0)
Jon Pérez Bolo	1998-2004	Rayo Vallecano	192 (43)
Jon Pérez Bolo	2004-2006	Gimnàstic Tarragona	79 (10)
Jon Pérez Bolo	2006-2008	CD Numancia	60 (11)
Jon Pérez Bolo	2008-	Barakaldo CF	- (-)
Gorka Iraizoz	1998-99	Txantrea KKE	
Gorka Iraizoz	1999-00	Baskonia	

10.1. irudia. 10.3. programak sortzen duen HTML orria jokalarien informazioarekin.

10.4. ARIKETA: TWITTERREKO TRAOLA BATEN JARRAIPENA EGITEA

Pythonen bidez Twitterrekin lan egiteko liburutegi asko daude garatuta²⁷. Liburu honetan *tweepy*²⁸ liburutegia erabiliko dugu, oso ahaltsua izateaz gain, oso komunitate aktiboa duelako eta kode irekia delako. Jarraian azaltzen dugun ariketa *pythoncentral* webguneko honako tutorial honetan dago oinarrituta: <http://pythoncentral.io/introduction-to-tweepy-twitter-for-python/>.

BeautifulSoup-ekin bezala, *tweepy* erabiltzeko aurrena instalatu beharko dugu. Modu errazena *pip3* erabiltzea da:

```
$ sudo pip3 install tweepy
```

Kodearekin hasi baino lehen, konexioa ezarri beharko dugu Twitterrekin. Twitterrek *OAuth*²⁹ metodoa erabiltzea eskatzen du konexioa ezarri eta kautotu ahal izateko. Metodo horrek kautotzeko bi modu eskaintzen ditu, "aplikazio-erabiltzaile" ohiko kautotzea, eta "aplikazioa bakarrik" (*application-only*) kautotzea.

Adibide honetarako, azkeneko kautotze modua erabiliko dugu: "aplikazioa bakarrik". Modu horretan, aplikazioa bere izenean kautotzen da, erabiltzaile zehatz baten beharrik izan gabe. Horrek, baina, murriztapen batzuk ekartzen ditu, ezin izango baitugu txorik idatzi, erabiltzaileak bilatu, etab.

27. <https://dev.twitter.com/overview/api/twitter-libraries>

28. <http://www.tweepy.org/>

29. <https://dev.twitter.com/oauth>

Beraz, kodearekin hasi baino lehen aplikazio bat sortu behar dugu Twitterren, aplikazioen kudeatzailearen web orria erabiliz: <<https://apps.twitter.com/>>.

Create an application

Application Details

Name *

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. Tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request. If your application from using callbacks, leave this field blank.

10.2. irudia. Twitter aplikazio bat sortzeko inprimakia.

Behin aplikazioa sortu dugula, "*Keys and Access Tokens*" fitxan kautotzerako beharrezkoak diren kodeak lortuko ditugu. Segurtasunaren izenean, *Consumer Secret* kodea ez da publiko egin behar, baina hemen aurkezten ditugun programak probatzea errazteko asmoz, beharrezko kode guztiak eskainiko ditugu programaren kodean bertan.

Aurretik esan bezala, *tweepy*-k Twitterren APIra sarbidea eskaintzen du. API horren inguruan dokumentazio zabala dago, oso ondo azaldua dagoena (adibidez hemen: <<https://dev.twitter.com/overview/api>>). Oinarrian lau objektu ditu: *Tweets* (txioak), *Users* (erabiltzaileak), *Entities* (entitateak) eta *Places* (tokiak). Horietako objekturen bat atzituta, horren informazioa JSON formatuan jasotzen da, eta 8. kapituluaren ikusi dugun bezala, JSON formatuarekin lan egitea oso erraza da Pythonen, eta are errazagoa *tweepy* erabilita.

Ariketa honetan, parametro gisa jasotzen den traola (*hashtag* ingelesez) baten jarraipena egiteko programatxoa idatziko dugu.

Txioak zuzenean (*streaming*) jarraitzeko *tweepyn StreamListener* objektuaren instantzia bat sortu behar da, eta bereziki *on_status* metodoa birdefinitu beharra dago objektuei orientatutako programazioa erabiliz, interesatzen zaigun jokabidea izan dezan. Kasu honetan, txioaren testua eta bertan aipatutako traola guztiak inprimatu nahi ditugu.

Erroreen aurrean inprimatuko duen mezua birdefinitzea ere komenigarria da, exekuzioan gertatzen diren akatsen berri jaso eta konponbidea bilatzen lagunduko digulako. 10.4. programan islatzen da kodea.

```
import tweepy
from tweepy import StreamListener, Stream
import sys

class KorronteEntzulea(StreamListener):

    def on_status(self, status):
        # Txioaren testua inprimatzen du
        print('Txioaren testua: ' + status.text)

        # Txioan aipatzen diren traola guztiak inprimatzen ditu
        for hashtag in status.entities['hashtags']:
            print("\t#", hashtag['text'])

        return True

    def on_error(self, status_code):
        print('Erroreren bat. Kodea: ' + str(status_code))
        return True # entzuten jarraitzeko

    def on_timeout(self):
        print('Denbora gehiegi...')
        return True # entzuten jarraitzeko

entzulea = KorronteEntzulea()

consumer_key = "s9nvz9uD1dIhmVpOU6HBy5obo"
consumer_secret = "YzMXfhwVhfcFxFeax5nxj7vnrDro2Hcc3pVVPz1qcR2H2keBBB"
access_token = "163090148-oEayPiNVFXoSOLXWWfVsITkiZcluSEjq29iG5Ewj"
access_token_secret = "dKtXJaX2f4LujJegFVDol8iDVTAbAgYS0ZMcBia2J3lOu"

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

korrontea = Stream(auth, entzulea)
korrontea.filter(track=['#'+sys.argv[1]])
```

10.4. programa. Traola baten jarraipena *tweety* bitartez.

Kodean ikus dezakegunez, *KorronteEntzulea* klasea definitzen dugu *StreamListener* objektuaren instantzia gisa.

Ondoren programa exekutatzeko komandoa eta bi txiori dagokien irteera ikus daiteke. Adibidean jarraitzen den traola *python* da, baina beste bat zehatz dezakezu. Gogoratu txio bakoitzeko testua eta traolen zerrenda inprimatzen direla eta zuzenean ari garela entzuten, beraz tarte batez itxoin beharko duzu irteeran zerbait lortzeko. Programaren bukaera eragiteko *Ctrl-c* erabili behar da.

```
>>>>
python3 p10-4.py python
Txioaren testua: Order your FREE #Arduino #OpenSourceHardware #Python Dev Stickers,
FREE Shipping! #FreeDevStickers https://t.co/orNyLafzAB
    #Arduino
    #OpenSourceHardware
    #Python
    #FreeDevStickers

Txioaren testua: #Hiring for SE III #TechJobs #networking #softwaredevelopment
#automation #python #switches #career | Apply Now https://t.co/R3ow7J3xKv
    #Hiring
    #TechJobs
    #networking
    #softwaredevelopment
    #automation
    #python
    #switches
    #Career
...

```

10.4. programaren irteera *#python* traolarentzat.

10.5. PROPOSATUTAKO ARIKETAK

- 1) 10.1. programaren kodea findu behar duzu, buletaren barruan dauden baina berrietara estekarik ez duten izenburuak kentzeko (hizkuntzak, adibidez). Horretaz gain, "*Europe headlines*" izenburupean dauden titularrak gehi ditzakezu.
- 2) 10.3. ariketa sinpleagoa litzateke beste liburutegiren batekin egin izan bagenu. Proba itzazu *HTML.py*³⁰ eta *prettytable*³¹ moduluak ariketa bera egiteko.
- 3) 10.4. ariketan informazio gehiago gehitu behar duzu, *status* objektuaren *entities* atribututik jaso, hala nola aipatzen diren erabiltzaileak edota estekak. Horrez gain, eman begirada bat *status* elementuari zenbateko informazioa duen ikusteko eta erabili informazio horietako bat.

30. <http://www.decalage.info/en/python/html>

31. <https://code.google.com/archive/p/prettytable/>

11. Aplikazioa II: argazkien tratamenduak eta jokoak

Kapitulu honetan programa luzeago batzuk landuko ditugu, gure ustez oso motibagarriak diren bi gaitan: jokoak eta irudien tratamendua. Kapitulu honetarako oso baliagarria gertatu zaigu webgune hau eta bertan agertzen diren liburuak: <<https://inventwithpython.com/>>.

Ariketa hauetan gauza ahaltsu samarrak lortu nahi dira, eta hori lortzeko aurreprogramatutako moduluak erabiliko dira. Modulu horiek instalatzea izango da lanaren parte bat. Bestalde, ezin gara sartu modulu horien zehaztasunak esplikatzen, bestela liburua izugarri luzatuko litzateke eta. Hori egin ordez, erabilera-adibideen bitartez ikasteko estrategiari helduko diogu.

11.1. ARGAZKIEN TRATAMENDUA

Argazkiei tratamendu bereziak egitea ohikoa da. Zerbait puntuala denean edo ukitu artistikoa bada banan-banan egin behar da GIMP bezalako programa bat erabiliz. Baina askotan lan errepikakorra egiten dugu, adibidez argazkien bereizmena eta tamaina jaisteko, formatua aldatzeko edo bereizgarriren bat txertatzeko.

Atal honetan lan errepikakor horiek automatizatzeko programazioa landuko dugu eta horretarako *Pillow* modulua erabiliko dugu. Modulu hori oso interesgarria da argazkien eta irudien tratamendua egiteko. Mota guztietako manipulazioak egin daitezke, argazki-zati bat lortzea, argazki bat kopiatzea, tamainaz edo formatuz aldatzea, biratzea, geruza desberdinak gainjartzea, filtroak aplikatzea...

Pillow modulu horren metodoak oso metodo sinpleak dira, erraz erraz ulertzen direnak. Informazio zehatzagoa hemen kontsulta daiteke: <<http://pillow.readthedocs.org/en/3.1.x/handbook/tutorial.html>>.

11.1.1. Instalazioa

Linux barruan modulua instalatu ahal izateko hurrengo urratsak egingo ditugu:

- *Pillow* paketearekin dauden mendeotasunak ebaztea. Guk erabilitako Ubuntu sisteman honela egiten da³²:

```
sudo apt-get install python3-dev python3-setuptools
```

32. <http://askubuntu.com/questions/427358/install-pillow-for-python-3>

```
sudo apt-get install libtiff4-dev libjpeg8-dev zlib1g-dev libfreetype6-dev
liblcms2-dev libwebp-dev tcl8.5-dev tk8.5-dev
```

- *Pillow* bera instalatzea:

```
sudo pip3 install Pillow
```

11.1.2. Ariketa: Argazkiei logo bat itsatsi nahian

Ariketa honetan *png* edota *jpg* formatuan dauden hainbat argazki ditugu karpeta baten barruan (azpikarpetetan ere egon daitezke) eta argazki horietan guztietan, tamaina batetik gorakoak baldin badira, logo bat itsatsi nahi dugu eta sortutako argazki berriak karpeta berri batean utziko ditugu. Logoa handi samarra denez, azalera txikiagoarekin itsatsiko dugu. Programak, beraz, hiru parametro izango ditu: argazkiak dauzkan karpeta, logoa duen fitxategia eta emaitzak uzteko karpeta (aurretik sortua)³³.

11.1. programan kode osoa dago oharrez lagunduta. Mesedez, prestatu datuak (argazki batzuk karpeta batean, fitxategi bat logo batekin edo QR kode batekin, eta emaitza uzteko karpeta huts bat), gero probatu programa eta, azkenik, kodea irakurri eta ulertzen saiatu.

Argazkien tamaina minimoa eta logoa txertatzeko tokia konstanteen bidez adierazi dira. Logoaren tamaina erdira jaisten da zabalera zein altueraz.

```
# argazkiak fitxategi-sisteman zehar bilatu, logoa jarri eta gorde
# 3 parametro: bilatzeko karpeta, logoa duen fitxategia
#          eta emaitzak uzteko karpeta

import sys
import os
import shutil          # shell utils - SEko batzuk errazago
from PIL import Image

erroa = sys.argv[1]    # argazkien bilaketarako katalogoa
logof = sys.argv[2]    # logoaren fitxategia
helb = sys.argv[3]     # emaitzak uzteko katalogoa

ZABMIN = ALTMIN = 150  # Zabalera eta altuera minimoa
TXERKOORD = (0,0)     # logoa txertatzeko koordinadak

ArgazkiKop = 0
IaArgazkiKop = 0
EzArgazkiKop = 0

# logoa-ren fitxategia ireki eta dimentsioak
logoa = Image.open(logof)
zab, alt = logoa.size
logotxikia = logoa.resize((int(zab/2),int(alt/2)))
```

33. Ariketa honen inspirazioa hemen aurki daiteke: <<https://automatetheboringstuff.com/chapter17/>>.

```

for karp, azpik, fitxategiak in os.walk(erroa):
    for fitx in fitxategiak:
        fitxabs = os.path.join(karp, fitx)
        # egiaztatu luzapena .png or .jpg.
        fitxoin, luzap = os.path.splitext(fitx)
        if luzap != ".png" and luzap != ".jpg":
            # ez da argazkia
            EzArgazkiKop += 1
            continue # hurrengo fitxategira (else ez jartzeko)

        # irudia ireki eta dimentsioak lortu
        irudia = Image.open(fitxabs)
        zab, alt = irudia.size
        # dimentsioak gainditzen direla ziurtatu
        if zab>ZABMIN and alt>ALTMIN:
            # nahikoa handia, argazkia
            ArgazkiKop += 1
            print("ARG", fitxabs, fitx)
            # fitxategia kopiatu
            irudia.paste(logotxikia, TXERKOORD)
            irudia.save(helb + '/' + 'logo_' + fitx)
            # helburuko katalogoan: 'logo_'izen
        else:
            # txikiegia da
            IaArgazkiKop += 1
            print("IA", fitxabs, fitx)

print("KOPURUAK - Argazkiak:", ArgazkiKop, "Ia-argazkiak:", IaArgazkiKop,
      "Ez-argazkiak:", EzArgazkiKop)

```

11.1. programa. Argazkietan logo bat txertatzea.

Kodean ikusten denez *Image* objektua erabiltzen da argazkia eta logoa lantzeko. *Image* objektuko *open* metodoarekin irekitzen dira fitxategi horiek. Metodo horrek itzultzen duen objektua gero beste hainbat metodorekin aplikatu ahal izango dugu: *size* tamaina lortzeko, *resize* tamaina aldatzeko, *paste* beste argazki edo irudi bat itsasteko eta *save* kopia bat gordetzeko.

Karpetaren barruko fitxategietara iristeko *os.walk* metodoa erabili da (kontuz! hirukoteak itzultzen ditu: karpetak, azpikarpetak eta fitxategiak). Lortutako fitxategietan izen osoak lortzen dira *join* metodoaren bitartez, baina baita luzapena eta oina *splitext* metodoaren bidez ere.

11.1.3. Proposatutako hobekuntzak

1) 11.1. programan honako hobekuntza hauek egin behar dira: tamaina minimo eta maximoa parametro gisa jaso behar ditu eta logoa goi-ekzerrean jarri beharrean behe-eskuinean jarri behar da.

2) Logoa gehitu beharrean, testu bat gehitu nahi da. Beste aukera da ur-marka (*watermark*) bat jartzea.

11.2. JOKOAK

Jokoak algorithmitika ikasteko zein lantzeko oso interesgarriak dira, baina konplexutasun handiegia izan ohi dute honelako liburu baterako, interfazea oso garrantzitsua delako, grafiko edo animazioen bidez eraiki behar izaten delako. Horrez gain, jokoak programatzeko modua ere askotan desberdina izaten da, gertaerei orientatutako metodologia erabili behar delako. Horrelakoetan programa nagusian bertan zailtasun handirik ez da egoten, une oro esna egongo diren hainbat funtzio/metodo martxan jarri baizik ez da egiten, gehienak gertaerei orientatuak direnak. Tekla bat sakatzea edo askatzea, denbora-tarte bat igarotzea... horrelakoak izaten dira ohiko gertaerak.

Bi joko landuko ditugu, bat oso interfaze sinplea duena eta bestea interfaze konplexuago batekin (liburutegi bat erabiliz).

11.2.1. Hiru lerroka edo artzain-jokoa (lehen hurbilpena)

Joko hau oso ezaguna da eta oso sinplea. Hala ere programazioa ez da ebidentea. 3x3 taula batean bi jokalaria saiatzen dira 3ko lerro bat osatzen (errenkada bat, zutabe bat edo diagonal bat). Lehen jokalariaren aukerak X sinboloarekin markatzen dira eta bigarrenarenak O sinboloarekin.

Programaren elementu nagusia partidaren egoera, taularen egoera irudikatzen duen datu-egitura izango da. 3 azpizerrenda (errenkadak) dituen zerrenda bat. Azpizerrenda bakoitzean errenkada horretako hiru osagaiak izango dira. Osagaien balioak hiru egoera islatzeko gai izan behar du: hutsik egotea, jokalaria baten marka eta bestearena. Gure programan hutsik egotea -1 zenbakiarekin adieraziko da, X jokalariaren marka 1 zenbakiarekin, eta O jokalariarena 0 zenbakiarekin. Hasierako egoeran balio guztiak -1 izango dira.

Bestalde, 9 gelaxka (3x3) horiek izendatzeko modu bat behar da, gure aukera 1-9 tartearen arabera izan da.

Lehen hurbilpenean bi jokalaria aurreikusten dira, baina biak teklatu beretik arituko dira. Nolabait jokalaria bere buruaren aurka arituko da. Programa, beraz, txandak banatuko ditu eta irabazlea erabakiko. Hau izango da jokalaria agertuko zaion lehen mezua:

Hauek dira erabili behar dituzun gelaxken zenbakiak:

```

1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 8 | 9
-----

Hasiera!

  |  |
-----
  |  |
-----
  |  |
-----

txanda: 1

```

Programaren diseinuan funtzio nagusi hauek aurreikusi dira:

- taularen hasierako egoera inprimatzea, gelaxken zenbakiekin: *idatzi_taula_has()*
- taularen egoera irudikatzea: *idatzi_taula(t)*
- jokalaria bati dagokion jokaldia irakurtzea: *irakurri_jok(jok)*. Gelaxkaren zenbakia zuzena dela (1-9) eta libre dagoela egiaztatuko du
- baten batek irabazi duen ala ez aztertzea: *irabazlea(t)*

11.2. programan kode osoa dago. Mesedez, probatu, kodea irakurri eta ulertzen saiatu.

```

# "hirurak lerroan" jokuaren inplementazioa

# BI JOKALARIAK TEKLATU BERETIK
# PROGRAMA: EPAILE LANA

# jatorrizko ideiak:
# http://code.activestate.com/recipes/578816-the-game-of-tic-tac-toe-in-python/
# https://inventwithpython.com/chapter10.html
# http://www.linuxhispano.net/2013/09/03/tres-en-raya-python/

# taula idatzi gelaxka bakoitzean dagokion zenbakiarekin
def idatzi_taula_has():
    print ("Hauek dira erabili behar dituzun gelaxken zenbakiak:")
    print()
    # 3 errenkada
    for i in range(3):
        print (" ", end=" ")
        # 3 zutabe

```

```

    for j in range(3):
        # balioa: indizearena+1
        print (i*3+j+1, end="")
        if j != 2:
            # bereizgarria
            print (" | ", end="")
        # lerro-tartea
        print("\n-----")
    print()

# partidaren egoera idatzi.
# t- zerrenda dago egoera: -1 hutsik, 1 - X, 0 - O
def idatzi_aula(t):
    print()
    # 3 errenkada
    for i in range(3):
        print (" ", end="")
        # 3 zutabe
        for j in range(3):
            if t[i*3+j] == 1:
                print ('X', end="")
            elif t[i*3+j] == 0:
                print ('O', end="")
            else:
                print (' ', end="")
        # bereizgarria
        if j != 2:
            print (" | ", end="")
        # lerro-tartea
        print("\n-----")
    print()

def irakurri_jok(jok):
    zuzen = False
    while not zuzen:
        try:
            pos = int(input(jok + "- aukeratu gelaxka: " + "(1-9)? "))
            if pos >= 1 and pos <= 9:
                return pos-1
            else:
                print ("Aukera desegokia! Beste bat?\n")
                idatzi_aula_has()
        # badaezpada: adib. teklatze okerra
    except Exception as e:
        print (user + " aukera desegokia! Beste bat?\n")

def irabazlea(t):
    konbin_guztiak = [[1,2,3], [4,5,6], [7,8,9], [1,4,7], [2,5,8], [3,6,9],
                      [1,5,9], [3,5,7]]
    for k in konbin_guztiak:
        if t[k[0]-1] == t[k[1]-1] and t[k[1]-1] == t[k[2]-1]:
            return t[k[0]-1]
    return -1

```

```

def idatzi_buk(t,m):
    idatzi_taula(t)
    print(m)
    print("\nAgur!\n")

# programa nagusia
# hasiera, jokalaria txandakatzea eta irabazlea erabakitzea

idatzi_taula_has()

taula = []
for i in range(9):
    taula.append(-1)

print ("Hasiera!")

buk = False
txanda = 0

while not buk:

    idatzi_taula(taula)
    print ("txanda: " + str(txanda+1))
    if txanda % 2 == 0:
        jokalaria = 'X'
    else:
        jokalaria = 'O'

    # sarrera irakurri
    p = irakurri_jok(jokalaria)
    while taula[p] != -1:
        print ("Aukera desegokia! Gelaxka hartuta. Beste bat?\n")
        p = irakurri_jok(jokalaria)
    if jokalaria == 'X':
        taula[p] = 1
    else:
        taula[p] = 0

    # aurreratu jokaldia eta testeatu bukaera
    txanda += 1
    if txanda > 4:
        irab_jok = irabazlea(taula)
        if irab_jok != -1:
            if irab_jok ==1:
                mezua = "Irabazlea: " + "X"
            else:
                mezua = "Irabazlea: " + "O"
            idatzi_buk(taula,mezua)
            buk=True
        elif txanda == 9:
            idatzi_buk(taula,"Irabazlerik ez!")
            buk=True

```

11.2. programa. Hiru lerroka edo artzain-jokoa, 1. bertsioa.

11.2.2. Hiru lerroka edo artzain-jokoa (bigarren hurbilpena)

Kasu honetan programa bera, antolatzaile/epailea izateaz gain, jokalaria izango da. Horretarako programa nagusia aldatu da, txanda bakoitietan jokolariari bere mugimendua eskatuko dio eta bakoitietan makinak bere jokaldia asmatuko du.

```
# programa nagusia
# makinaren aurka
# jokalaria (X, 1) hasten da
# makinak (O, 0) ez du galduko (baina galtzea aurreikusita dago kodean)

idatzi_taula_has()

taula = []
for i in range(9):
    taula.append(-1)

print ("Hasi zu!")

buk = False
txanda = 0

while not buk:
    idatzi_taula(taula)
    if txanda % 2 == 0:          #jokolariaren txanda
        jokalaria = 'X'
        print ("txanda: " + str(txanda+1))
        p = irakurri_jok(jokalaria)
        while taula[p] != -1:
            print ("Aukera desegokia! Gelaxka hartuta. Beste bat?\n")
            p = irakurri_jok(jokalaria)
        taula[p] = 1
    else:                        #makinaren txanda
        jokalaria = 'O'
        p = hurrengo_jok(taula)
        taula[p] = 0

    # aurreratu jokaldia eta testeatu bukaera
    txanda += 1
    if txanda > 4:
        irab_jok = irabazlea(taula)
        if irab_jok != -1:
            if irab_jok == 1:
                mezua = "Irabazi duzu! Zorionak!"
            else:
                mezua = "Etxea irabazle! Hurrengoan agian?"
            idatzi_buk(taula,mezua)
            buk=True
        elif txanda == 9:
            idatzi_buk(taula,"Irabazlerik ez!")
            buk=True
```

11.3. programa. Hiru lerroka edo artzain-jokoa, 2. bertsioaren programa nagusia.

Makinak zer jokaldi egin behar duen zehaztuko duen *hurrenko_jok* funtzioa definituta dagoela suposatu da programa nagusi horretan. Beheko kodean ikus daitekeenez inplementazio sinple bat egin da, lau urrats posible hauek kontuan hartzen dira:

1. zentroa libre badago, horixe aukeratu,
2. bestela saihestu beste jokalaria irabaztea,
3. arrisku hori ez badago, erpinetako gelaxka libre bat eskuratu horrelakorik egonez gero,
4. bestela libre dagoen beste edozeinetan jarri (kanpoko marra bateko erdigunean).

```
def hurrenko_jok(t):
    ZENTR=5
    ERPINAK=[1,3,7,9]
    ERDIGUNEAK=[2,4,6,8]
    if t[ZENTR-1]==-1:
        return ZENTR-1
    else:
        p=galzorian(t)
        if p!=-1:
            return p
        p=bilatu_pos(t,ERPINAK)
        if p!=-1:
            return p
        p=bilatu_pos(t,ERDIGUNEAK)
        return(p)
```

11.3. programaren *hurrenko_jok()* funtzioa.

2. urratsa da zailena (*galzorian* funtzioa) eta honela inplementatzen da (aurretik programatuta zegoen *irabazlea* funtzioan oinarrituta):

```
def galzorian(t):
    konbin_guztiak = [[1,2,3],[4,5,6],[7,8,9],[1,4,7],[2,5,8],[3,6,9],
                      [1,5,9],[3,5,7]]
    for k in konbin_guztiak:
        arrisku=0; pos=0;
        for j in range(3):
            if t[k[j]-1] == 1:
                arrisku += 1 # zenbat hiruen artean
            elif t[k[j]-1] == -1:
                pos=k[j] # lerro berean libre dagoena
        if arrisku == 2: # jokalaria 2 hartuta baditu
            return pos-1 # libre dagoena
    return -1
```

11.3. programaren *galzorian* funtzioa.

3. eta 4. urratsetarako funtzio bera erabiltzen da, *bilatu_pos* izenekoa. Zerrenda batean zehazten diren posizioen artean libre dagoen lehena itzultzen du.

```
def bilatu_pos(t,z):
    for pos in z:
        if t[pos-1]==-1:
            return pos-1
    return -1
```

11.3. programaren *bilatu_pos* funtzioa.

11.2.3. Mahai-tenisa (ping-pong)

Oraingo honetan interfazea konplexuagoa izango da. Ping-pong bideo-joko klasikoan teklatutik bi jokalaria kontrolatuko ditu erabiltzaileak, ezkerrez "w" pala igotzeko eta "s" jaisteko, eta eskuinez gorako eta beherako geziei dagozkien teklak erabiliz. Programa pilotaren eta palen mugimenduaz zein hormetako erreboteez arduratuko da, baita marrazketaz eta tantoak kontatzeaz eta islatzeaz.

Kasu honetan interfaze grafiko sofistikatuago bat erabiliko dugu eta gauzak erraztearren, Internet konexioa edukita, *codeskulptor.org* gunearen aukerak erabiliko ditugu. Aukeren artean hauek dira azpimarragarrienak:

- *simplegui* (*Simple GUI*) modulua erabil dezakegu interfazea programatzeko, besterik instalatzeko beharrik gabe. Oso interfaze sinplea da oso metodo ahaltsuak dituelako (luze joko luke hemen metodo guztien zehaztasunak emateak)
- exekuzioa nabigatzailean azalduko zaigu

11.4. programan dago proposatzen dugun soluzioa³⁴, baita Interneteko helbide honetan ere: <http://www.codeskulptor.org/#user41_vKzuepVh6F_27.py>.

Gertaerei orientatuta dago programa, eta oharren bidez nahiko ondo azalduta dagoen arren, komenigarria da bukaeran dagoen kodea (programa nagusia) azaltzea. Ondoko urratsak ematen dira:

- *frame* bat (irudia) sortzea *simplegui*-k duen metodoaren bidez:

```
frame = simplegui.create_frame("Pong", ZABALERA, ALTUERA)
```

- eguneraketari dagokion funtzioa deklaratzeko, kodearen atal nagusia dena, *frame* objektuari dagokion metodoarekin:

```
frame.set_draw_handler(draw)
```

34. Eskerrak UPV/EHuko irakaslea den Jesus Ibañezi bere soluzioa gurekin partekatzeagatik.

- tekla sakatzeari eta askatzeari dagozkien funtzioak deklaratzeko, dagozkien metodoen bidez:

```
frame.set_keydown_handler(keydown)
frame.set_keyup_handler(keyup)
```

- botoi lagungarri bat gehitzea jokia berriro berrabiatzeko

```
frame.add_button("Restart", init, 100)
```

- programa martxan jartzea: aldagai globalen balioa hasieratzea (*init*), hasierako marrazketa eta funtzioen aktibazioa (*frame.start*) eta pilota mugitzen hastea (*pilota_hasi*).

Kodea luzea denez, lehen probatu, eta gero arretaz irakurtzea gomendatzen da. Horretarako onena da codeskulptor.org gunean fitxategi huts batetik abiatzea, kodea kopiatzea (11.4. programa) eta probatzea.

```
# Jesus Ibañezen bertsiotik eratorria

import simplegui
import random

# Hasieraketa

ZABALERA = 600          #pantailaren dimentsioa
ALTUERA = 400
ERRADIOA = 10          # pilotaren erradioa

PALA_ZAB = 8           # palaren dimentsioak
PALA_ALT = 80
PALA_ZAB_ERDIA = PALA_ZAB / 2
PALA_ALT_ERDIA = PALA_ALT / 2
pala_has_abiadura = 7   # pala mugitzeko hasierako abiadura

# Partida irabazteko tanto kopurua
tanto_kop = 5

# pilota azaltzeko funtzioa (eskuin/ezker)
def pilota_hasi(eskuina): # hasieran edo tantoa bukatutakoan
# parametroa boolearra: egiazkoa-eskuinerantz, bestela ezkererantz
    global pilota_pos, pilota_abiad
    pilota_pos = [ZABALERA/2, ALTUERA/2] # pilota erditik irteten da
    # abiadura zoriz
    pilota_abiad = [random.randrange(120, 240)/60,
                    -random.randrange(60, 180)/60]
    if not eskuina:          # ezkererantz
        pilota_abiad[0] *= -1
```

```

# jokoaren hasiera eta bukaera
def init():
    # aldagai globalak
    global bukaera, pala1_pos, pala2_pos, pala1_abiad, pala2_abiad
    global MEZU1, MEZU2, tantoak1, tantoak2
    bukaera = False
    MEZU1 = MEZU2 = ""
    tantoak1 = tantoak2 = 0
    # palak hasieran zentratuta. bi hasieraketa batera
    pala1_pos = pala2_pos = ALTUERA / 2
    pala1_abiad = pala2_abiad = 0

def joko_buk(irab):
    # bukaera ezarri eta bukaerako mezua
    global bukaera, MEZU1, MEZU2
    bukaera = True
    MEZU1 = "AMAITU DA"
    MEZU2 = str(irab) + ". jokalaria irabazle!"

# gertaeren kudeatzailea: gertaerek idazketa dakarte
# programaren mamia
def draw(c):
    # aldagai globalak
    global tantoak1, tantoak2, pala1_pos, pala2_pos
    global pilota_pos, pilota_abiad

    # palaren posizio bertikala eguneratzea, pantailan mantenduta
    if not bukaera:
        pala1_pos += pala1_abiad
        pala1_pos = max(pala1_pos, PALA_ALT_ERDIA)
        pala1_pos = min(pala1_pos, ALTUERA-PALA_ALT_ERDIA)
        pala2_pos += pala2_abiad
        pala2_pos = max(pala2_pos, PALA_ALT_ERDIA)
        pala2_pos = min(pala2_pos, ALTUERA-PALA_ALT_ERDIA)

    # draw mid line and gutters
    c.draw_line([ZABALERA / 2, 0],[ZABALERA / 2, ALTUERA], 1, "White")
    c.draw_line([PALA_ZAB, 0],[PALA_ZAB, ALTUERA], 1, "White")
    c.draw_line([ZABALERA - PALA_ZAB, 0],[ZABALERA - PALA_ZAB, ALTUERA],
                1, "White")

    # palak marraztea
    c.draw_line((PALA_ZAB_ERDIA, pala1_pos-PALA_ALT_ERDIA),
                (PALA_ZAB_ERDIA, pala1_pos+PALA_ALT_ERDIA),
                PALA_ZAB, "White")
    c.draw_line((ZABALERA-PALA_ZAB_ERDIA, pala2_pos-PALA_ALT_ERDIA),
                (ZABALERA-PALA_ZAB_ERDIA, pala2_pos+PALA_ALT_ERDIA),
                PALA_ZAB, "White")

    # pilota-ren kokapena eguneratzea eta marraztea: bi koordinadatan
    pilota_pos[0] += pilota_abiad[0]
    pilota_pos[1] += pilota_abiad[1]
    c.draw_circle(pilota_pos, ERRADIOA, 1, "White", "White")

```

```

# pilota ezker-paretara iristen
if pilota_pos[0] <= ERRADIOA + PALA_ZAB:
    if (pilota_pos[1] >= palal_pos - PALA_ALT_ERDIA and
        pilota_pos[1] <= palal_pos + PALA_ALT_ERDIA):
        # Pala pilotari dagokion tokian dago -> abiadura handitzea
        pilota_abiad[0] *= -1.1
    else:
        # puntua 2. jokalariaarentzat
        if not bukaera:
            tantoak2 += 1
            if tantoak2 == tanto_kop:
                joko_buk(2)
        pilota_hasi(True)          # berriro hasi pilota, erditik

# pilota eskuin-paretara iristen
if pilota_pos[0] >= ZABALERA - 1 - ERRADIOA-PALA_ZAB:
    if (pilota_pos[1] >= pala2_pos - PALA_ALT_ERDIA and
        pilota_pos[1] <= pala2_pos + PALA_ALT_ERDIA):
        # Pala pilotari dagokion tokian dago -> abiadura handitzea
        pilota_abiad[0] *= -1.1
    else:
        # puntua 1. jokalariaarentzat
        if not bukaera:
            tantoak1 += 1
            if tantoak1 == tanto_kop:
                joko_buk(1)
        pilota_hasi(False)

if pilota_pos[1] <= ERRADIOA or pilota_pos[1] >= ALTUERA - 1 - ERRADIOA:
    # pilota jaurti egiten du goian edo behean talka egitean
    pilota_abiad[1] *= -1

# markagailuaren inprimaketa
c.draw_text(str(tantoak1), (ZABALERA/4 - 20, 75), 40, "White")
c.draw_text(str(tantoak2), (3*ZABALERA/4 - 20, 75), 40, "White")

# mezuaren inprimaketa (beti hutsa, bukaeran izan ezik)
c.draw_text(MEZU1, (ZABALERA/4, 3*ALTUERA/4), 40, "White")
c.draw_text(MEZU2, (65+ZABALERA/4, 6*ALTUERA/7), 25, "White")

def keydown(key):
    # tekla sakatzearen eragina: palaren abiadura konstantea
    # 1. jokalaria "w" tekla pala igotzeko eta "s" jaisteko
    # 2. jokalaria gezien teklekin
    global palal_abiad, pala2_abiad
    if key == simplegui.KEY_MAP["w"]:          # 1. pala gora
        palal_abiad = -palal_has_abiadura
    elif key == simplegui.KEY_MAP["s"]:        # 1. pala beheara
        palal_abiad = palal_has_abiadura
    elif key == simplegui.KEY_MAP["up"]:       # 2. pala gora
        pala2_abiad = -pala2_has_abiadura
    elif key == simplegui.KEY_MAP["down"]:     # 2. pala beheara
        pala2_abiad = pala2_has_abiadura

```

```

def keyup(key):
    # tekla askatzearen eragina: palaren mugimendua geratu egiten da
    global pala1_abiad, pala2_abiad
    if key == simplegui.KEY_MAP["w"] or key == simplegui.KEY_MAP["s"]:
        pala1_abiad = 0
    elif key == simplegui.KEY_MAP["up"] or key == simplegui.KEY_MAP["down"]:
        pala2_abiad = 0

# frame-a (irudia sortzea)
# irudia sortzeko metodoa
frame = simplegui.create_frame("Pong", ZABALERA, ALTUERA)
# eguneraketa
frame.set_draw_handler(draw)
# tekla sakatzean
frame.set_keydown_handler(keydown)
# tekla askatzean
frame.set_keyup_handler(keyup)
# botoi lagungarria
frame.add_button("Restart", init, 100)

# hasiera
init()
frame.start()
pilota_hasi(False)

```

11.4. programa. Mahai-tenisa *simplegui* bidez.

Lokalean (zure ordenagailuan) exekutatu nahi izanez gero, *codeskulptor.org* gunerako asmatutako *simplegui.py* instalatu beharko litzateke, baina ez da erraza, hainbat mendekotasun dituelako (*pillow*, *pygame* eta *matplotlib*) eta batzuetan arazoak daudelako Python3-rekin³⁵. Are gehiago, *codeskulptor.org* bera python2.7-rako dago egokituta.

11.2.4. Proposatutako hobekuntzak

- 1) 11.1. programaren kodea laburtzeko *idatzi_taula_has()* eta *idatzi_taula(t)* funtzioak bateratu funtzio bakar batean.
- 2) 11.2. programan defini ezazu zerrenden hasieraketa 6. kapituluan azaldutako *comprehension* ezaugarria baliatuta.
- 3) 11.3. programa alda ezazu konbinazio irabazleak programaz detekta ditzan eta irabazteko baliatu gero. *Galzorian* egoera detektatu den bezalaxe, *Irabazteko zorian* egoteko aukera ere detekta dezake programak, eta jokaldi hori baliatu irabazteko. *galzorian(t)* funtzioan oinarrituta definitzea erraza da.
- 4) 11.4. programan aldaketak egin: puntu kopurua, pilotaren eta palen tamaina, abiadura...

35. <https://pypi.python.org/pypi/SimpleGUITk> <http://stackoverflow.com/questions/16387770/how-to-integrate-simplegui-with-python-2-7-and-3-0-shell>

- 5) 11.4. programa sinplifikatu behar duzu eta ezkerreko jokalaria kendu. Jokalariaren ordeztu pareta utzi, eta geratzen den jokalariaren helburua ahalik eta gutxi galtzea izango da. Gero, programa hobetzeko hainbat gehikuntza egin daitezke: denbora kontrolatu eta azaldu, pilotaren abiadura handitzen joan...
- 6) 5. ariketa aldatu alde bakar batean bi jokalarik egon daitezkeen, frontoi batean bi jokalarik daudela simulatzeko.
- 7) [erronka handia] 11.4. programa aldatu behar duzu makinaren aurka jokatzeko. Alde bateko palaren mugimenduak programak berak mugitu beharko ditu. Makinak beti irabaztea lortzen duzunean, gehitu erloju bat eta erronka litzateke denbora-tarte batean partida kopuru bat baino gutxiago galtzea. Hori konbina liteke pilotaren abiadura hautatzeko aukera gehitzearekin.
- 8) <<https://inventwithpython.com/pygame/chapter3.html>> gure ariketa-bilduman ere kopia bat dago) proba ezazu memoria-joko hori. Ondo ibil dadin *pygame* instalatu behar da eta *python2.7* erabili. Konplexua da: irudiak daude, sagua erabiltzen da... Saiatu ulertzen eta euskarara ekartzen. Behin ulertuta saia zaitez aldaketak egiten gero; adibidez, irudiak aldatzea.

12. Aplikazioak III: testuak lantzen

Python programazio-lengoiaren erabilera oso zabalduta dago testuak prozesatzen dituen arloan. Lana asko errazten du arlo honetarako bereziki diseinatu zen NLTK liburutegiaren erabilerak.

Landuko dituzun testuak eskuratzeko, hainbat aukera dituzu gaur egun: i) zuk zeuk lortutako testua erabiltzea, ii) edukia Internetetik eskuratzea, eta iii) NLTK tresna multzoarekin eskaintzen diren testuak hartzea. Lehenengo bi kasuetan 7. eta 10. kapituluetan landu duguna oso baliagarria izango zaizu. Lortutako testu horiek NLTKko testuekin integratu ahal izango ditugu, gero bertako metodoekin erabili ahal izateko.

Kapitulu hau osatzeko, *TAPE Testu-analisirako PERL erremintak* liburua (<http://ueu.org/download/liburua/PERL.BERRIA.osoa.pdf>) kontsultatzea merezi du. Liburu horretan testuen gaineko ariketa asko daude, baina Python lengoia erabili beharrean Perl lengoia erabiltzen da hor.

Ariketa sinple batez hasiko gara.

Pentsa dezagun telebista-programa batean euskarazko hitzekin joko bat sortu behar dutela eta jakin nahi dutela euskaraz letra bakoitzak duen probabilitatea. Horretarako testu luze bat hartu behar da, eta letra bakoitzaren agerpen kopurua kontatu Pythoneko hiztegi-egitura batean. Bukaeran, maiztasun absolutuak jakinda eta letra guztiak kontatu baditugu, erraza izango da maiztasun erlatiboak (probabilitateak) ere kalkulatzeko, ordenatzeko eta inprimatzeko.

12.1. programan kodea azaltzen da. Erabiltzen den testu-fitxategi luzea (*wiki_eu_labur.txt*) geroxeago deskribatuko dugu kapitulu honetan.

```

hizt = dict()
alfa = "abcdefghijklmnopqrstuvwxyz"
kont=0

fitx = 'wiki_eu_labur.txt'
for lerro in open(fitx,'rU').readlines():
    lerrox = lerro.lower()
    for k in lerrox:
        if(k in alfa):
            kont=kont+1
            if(k in hizt):
                hizt[k] += 1      # inkrem
            else:
                hizt[k] = 1      # 1 balioa

for j in sorted(hizt):
    print("%s %d %2.4f" % (j, hizt[j], hizt[j]*100/kont))

```

12.1. programa.

Exekuzioaren emaitza hau da:

```

a 78778 15.8231
b 11142 2.2379
c 2681 0.5385
d 14587 2.9299
e 57703 11.5900
f 3095 0.6217
g 11182 2.2460
h 8339 1.6749
i 45784 9.1960
j 2913 0.5851
k 24661 4.9533
l 18872 3.7906
m 8542 1.7157
n 34554 6.9404
o 29830 5.9916
p 6059 1.2170
q 141 0.0283
r 41902 8.4163
s 15150 3.0430
t 36044 7.2397
u 22583 4.5360
v 1315 0.2641
w 1509 0.3031
x 1642 0.3298
y 1065 0.2139
z 17715 3.5582
ñ 79 0.0159

```

Probabilitate horiek nahiko fidagarriak dira, baina testuetatik kalkulatu direnez, hitz deklinatuetako atzizkietan sarri azaltzen diren letrek ("a" atzizkia adibidez) behar baino agerpen gehiago jasoko dituzte. Probabilitate zehatzagoak behar izanez gero, hobe litzateke hiztegi batetik abiatzea eta ez testu-fitxategi batetik.

Ondoren testuak lantzeko teknikak eta adibideak hiru ataletan banatu ditugu: Interneteko erabilera-testuak eskuratzeko, NLTK paketearen erabilera bertako testuekin, eta NLTKren erabilera gure testuekin.

12.1. TESTUAK INTERNETETIK ESKURATZEA

Aurreko kapituluetan azaldutako hainbat modulu eta metodo erabilia, adibide sinple bat burutuko dugu, hainbat webguneren edukia arakatu eta parametro gisa pasatzen den hitzaren agerpen kopurua kalkulatu nahi dugu. 12.2. programan azaltzen da. Bertan ikus daitekeenez, *re* (espresio erregularrak) eta *urllib* moduluak erabiltzen dira lana errazteko. Lehenaren bitartez, bilaketak espresio erregularren bitartez egin daitezke; bigarrenaren bitartez, berriz, sareko edukiak eskura ditzakegu³⁶.

Webguneen URLak konstante moduan definituta daude zerrenda batean, baina hitza parametro gisa sartuko dugu.

```
import re
import sys
from urllib.request import urlopen

# webgune baten testua irakurtzeko eta testua bilatzeko
def kontatu_url(web, hitz):
    maiz = 0
    web_testu = urlopen(web)
    for lerro in web_testu:
        lerro_deskod = lerro.decode('latin-1')
        espres = re.compile(r"("+hitz+".*)")
        maiz = maiz + len(re.findall(espres, lerro_deskod))
    return (maiz)

# webguneen zerrendako emaitzak lortu eta hiztegi batean gordetzeko
def kontatu_n_url(web_zerrenda, hitz):
    emaitzak = {}
    for i in web_zerrenda:
        emaitzak[i] = kontatu_url(i, hitz)
    return(emaitzak)

# hiztegiaren datuak inprimatzeko
def idatz_hizt (hizt):
    for gako, balio in hizt.items():
        print ("%s:\t %d" % (gako, balio))

# programa nagusia
# aztertzeako kanalak
kanalak = ["http://www.ueu.eus",
            "http://www.sustatu.eus",
            "http://www.berria.eus"]

#aztertzeako hitza edo hitz-hasiera
h = sys.argv[1]
print(h, "hitzaren agerpenak: ")
idatz_hizt(kontatu_n_url(kanalak, h))
```

12.2a. programa. Hitz baten kontaketa zenbait webgunetan.

36. Gogoratu behar da 7. kapituluaren adibideetan antzeko ariketa bat aztertu dugula: webgune baten testuan oinarrituta hiztegi bat osatzea maiztasunak kontatuz (ik. 7.11. programa).

Programa horren exekuzioan agertuko dena halako zerbait izango da:

```
>>>
python3 p12-2.py euskara
euskara hitzaren agerpenak:
http://www.berria.eus: 12
http://www.ueu.eus: 2
http://www.sustatu.eus: 10
```

Espresio erregularra erabiltzen da hitzaren deklinabidea kontuan hartzeko. "." jarri dugu parametroaren atzean ondoren edozein karaktere ager daitekeela adierazteko. Parametroan "euskara" zehaztu beharrean "euskaraz" zehaztuz gero, agerpen kopurua jaitsiko da, baita espresio erregularrean aipatutako "." atzizkia kenduta ere. Edozein kasutan, geroago azalduko dugunez, espresio erregular sinpleekin ezin da lematizazio zuzen eta oso bat egin. Adibidez 'egin.*' espresioarekin ez da 'egiten' harrapatuko.

Webguneen beste ezaugarri interesgarri bat esteken agerpena da. Estekak baliatuz testu gehiago eskura daitezke, eta modu errekurtsiboan jarraituko bagenie, sarea arakatzeko robot moduko bat programatuko genuke. 12.2b. ariketan webgune bakoitzaren barruan estekak bilatu ditugu, eta esteken bitartez orri berriak eskuratu ditugu kontaketa burutzeko. Estekak lortzeko funtzioa eta deitzen dion programa-zatia bakarrik aldatu ditugu liburura.

```
import re
import sys
from urllib.request import urlopen

def lortu_estekak(kanal):
    joateko = []
    espres_erreg = r"<a href=\"(http.+?)\"[>\\s]"
    web_oso = urlopen(kanal)
    eduki = str(web_oso.read())
    estekak = re.findall(espres_erreg, eduki, re.I)
    egiaz_domeinu = kanal+r".+"
    egiaz_mota = r"\\.?(?! (pdf|jpg|docx?|png|gif|bmp|tiff)))*$"
    for esteka in estekak:
        domeinu_egoki = re.search(egiaz_domeinu, esteka)
        mota_egoki = re.search(egiaz_mota, esteka)
        if domeinu_egoki and mota_egoki:
            joateko.append(esteka)
    # bikoiztuak ezabatu
    joateko_multzo = set(joateko)
    joateko_multzo.discard(kanal) # kendu hasierakoa
    joateko = list(joateko_multzo)
    joateko.insert(0, kanal) # gehitu hasierakoa hasieran
    return(joateko)
```

```

# webgune baten testua irakurtzeko eta testua bilatzeko
def kontatu_url(web, hitz):
    maiz = 0
    espres = re.compile(r"("+hitz+".*)")
    zerr = lortu_estekak(web) # estekak lortu (jatorria ere)
    for url in zerr:
        # print(url)
        try: # erroreak saihestu
            web_testu = urlopen(url)
            for lerro in web_testu:
                lerro_deskod = lerro.decode('latin-1')
                maiz = maiz + len(re.findall(espres, lerro_deskod))
        except:
            continue
    return (maiz)

# webguneen zerrendako emaitzak lortu eta hiztegi batean gordetzeko
def kontatu_n_url(web_zerrenda, hitz):
    emaitzak = {}
    for i in web_zerrenda:
        print(i, ".....")
        emaitzak[i] = kontatu_url(i, hitz)
    return(emaitzak)

# hiztegiaren datuak inprimatzeko
def idatz_hizt (hizt):
    for gako, balio in hizt.items():
        print ("%s:\t %d" % (gako, balio))

# programa nagusia
# aztertze kanalak
kanalak = ["http://www.ueu.eus",
            "http://www.sustatu.eus",
            "http://www.berria.eus"]

#aztertze hitza edo hitz-hasiera
h = sys.argv[1]
print(h, "hitzaren agerpenak: ")
idatz_hizt(kontatu_n_url(kanalak, h))

```

12.2b. programa. Estekak lortzeko funtzioa eta erabilera.

Programa horren exekuzioan honako hau agertuko da pantailan:

```

>>>>
python3 p12-2b.py euskara
euskara hitzaren agerpenak:
http://www.ueu.eus .....
http://www.sustatu.eus .....
http://www.berria.eus .....
http://www.berria.eus:      565
http://www.sustatu.eus:     10
http://www.ueu.eus:         112

```

12.2. NLTK TRESNAK

NLTK-k oso erraza egiten du testuarekin lan egitea, tokenizatzailerak, lematizatzailerak, entitate izendunen ezagutzaileak eta abar eskaintzen baititu erraz asko erabiltzeko. Tresna horietako batzuen erabilera ikusiko dugu atal honetako kasu praktikoen bidez.

Dena dela, luze joko luke aukera guztiak garatzeak. Adibidez, luzeegi joko luke grafikoak egiteko metodo guztiak lantzeak edo ikasketa automatikoaren bidez sailkatzaileak sortu ahal izateko metodo guztiak lantzeak. Izan ere, NLTK-k berak liburu oso bat eskaintzen du Interneten libreki eskuragarri helbide honetan: <<http://www.nltk.org/book/>>. Testuekin lan egin nahi izanez gero, eta are gehiago, hizkuntzaren prozesaketan (NLP, *Natural Language Processing* ingelesez) lan egin nahi izanez gero, oso gomendagarria da liburua irakurtzea, oso argi azalduta dagoen informazio asko baitauka.

Esan bezala, euskarazko antzeko beste liburu bat dago kontzeptu horiek lantzen dituen, <http://www.buruxkak.eus/liburuak_ikusi/478/tape_testu_analisirako_perl_erremintak.html>, baina kasu horretan Perl programazio-lengoiaren bitartez landuta.

NLTK-k barneratzen dituen modulu batzuk, hauek dira:

Eginkizuna	Moduluak	Funtzionaltasuna
Testu-bildumak	<i>corpus</i>	Testu-bildumak eta lexikoak eskuratzea
Testu-prozesaketa	<i>tokenize, stem</i>	Hitzetan banatzea (tokenizatzea), lematizazioa*
Agerkidetzen bilaketa	<i>collocations</i>	Neurri estatistikoak osagaien agerkidetzaren probabilitatea kalkulatzeko (<i>t-test, chi-squared, point-wise mutual information</i>)
POS etiketatzea	<i>tag</i>	Desanbiguazioa testuingurua kontuan hartuta. Hainbat metodo: <i>n-gram, backoff, Brill, HMM, TnT</i>
Ikasketa automatikoa	<i>classify, cluster, tbl</i>	Hainbat metodo: erabaki-zuhaitzak, entropia maximoa, <i>naive Bayes</i> , EM, <i>k-means</i>
Azaleko sintaxia	<i>chunk</i>	Esprezio erregularrak, n-gramak, izen propioak
Analisi sintaktikoa	<i>parse, ccg</i>	Hainbat aukera: <i>chart, feature-based</i> , baterakuntza, probabilitikoa, mendekotasunak...
Ebaluazio-metrikak	<i>metrics</i>	Doitasuna eta estaldura (<i>precision, recall</i>), Komuztadura-koefizienteak...

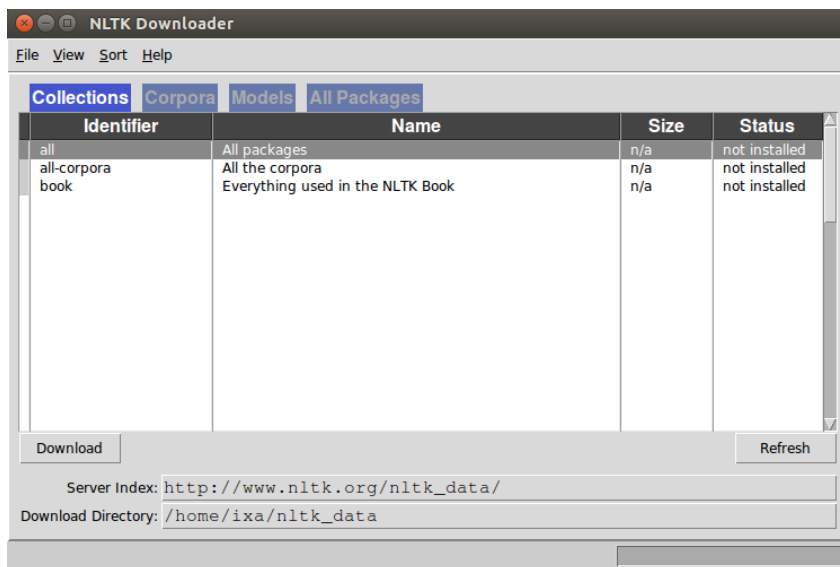
* *Stemming* sasilematizazioa da. Hitzari azken karaktereak kenduz lortzen da (sasi)lema, horrela *etxe*-ko-tik *etxe* lortzen da, baina *egiten*-etik egi (eta ez *egin*).

Liburu honetan modulu sinpleenak erabiliko ditugu, eta erabili ahala hobeto ulertu ahal izango duzu. Testu-bilduma gehienak ingelesez daudenez, hasierako adibideak ingelesezko testuekin egingo ditugu.

12.2.1. Instalazioa eta baliabideen deskarga

NLTK-k hainbat pakete eta corpus dauzka eskura, erraz asko erabil daitezkeenak. Erabili aurretik baina, baliabideok deskargatu behar ditugu. Horretarako lehenik eta behin NLTK instalatu behar da (*sudo python3 -m pip install -U nltk*) eta gero Python3-n bertan sartu eta bertako metodo batez baliabideak kargatu (*nltk.download()*).

12.1. irudian ikus dezakegu nola deskarga dezakegun baliabideren bat NLTKtik.



12.1. irudia. Baliabideak eta paketeak jaisteko aukera ematen duen interfazea.

Aukera egokia izan daiteke baliabide denak jaistea (denbora behar da, hobeto konexio azkarra duzunean), beste bat, behar ahala instalatzea. Jaisten duzuna (*nltk_data* karpeta) erro-katalogoan (edo beste katalogo zehatz batean: */usr/share* edo */usr/local/share* adibidez) jartzea gomendatzen da, geroago erraz aurkitu ahal izateko.

12.2.2. Ariketa: Corpusak, zenbaketak eta stopwords-ak

Berriki deskargatu dugun corpus bat erabiliko dugu hurrengo adibidean. *Brown Corpusa* ingelesezko milioi bat hitz bildu zuen lehen testu-corpusa izan zen, 1961. urtean Brown Unibertsitatean sortua. 500 jatorri ezberdinetatik jasotako testuak dira eta generoaren arabera (*news, fiction, humor, romance...*) sailkatuta daude.

Kasu praktiko honetan, argumentu gisa genero horietako bi landuko ditugu, bi genero horietako hitz ohikoenak lortuko ditugu.

```

import nltk,sys

#nltk-ren corpora inportatu: brown
from nltk.corpus import brown

#generoak jaso (ingelesez)
g1 = sys.argv[1]
g2 = sys.argv[2]

#lehen gaiaren corpora jaso hitz zerrenda gisa
corp1 = brown.words(categories=g1)
#bigarren generoaren corpora jaso hitz zerrenda gisa
corp2 = brown.words(categories=g2)

#corpusetan kontaketak egin minuskulak eta maiuskulak bereizi gabe
corp1_kontaketa = nltk.FreqDist(h.lower() for h in corp1)
corp2_kontaketa = nltk.FreqDist(h.lower() for h in corp2)

maiz1=corp1_kontaketa.most_common(10)
maiz2=corp2_kontaketa.most_common(10)

#kontaketak pantailaratu
print(g1+"-ren hitz ohikoenak:")
print(maiz1)
print(g2+"-ren hitz ohikoenak:")
print(maiz2)

```

12.3a. programa. Maiztasun handieneko hitzak.

Adibide gisa, "news" eta "science_fiction" generoak pasatzen baditugu, ikus dezakegunez, 10 hitz ohikoenak pantailaratzen ditugunean, puntuazio-markak, deklinabideak, artikulua eta preposizioak maizen agertzen diren hitzak direla ikus dezakegu:

```

>>>
python3 p12-3a.py news science_fiction
news-ren hitz ohikoenak:
[('the', 6386), ('', 5188), ('.', 4030), ('of', 2861), ('and', 2186), ('to',
2144), ('a', 2130), ('in', 2020), ('for', 969), ('that', 829)]
science_fiction-ren hitz ohikoenak:
[(',', 791), ('.', 786), ('the', 723), ('of', 329), ('to', 306), ('and', 294), ('a',
236), ('"', 235), (''', 235), ('was', 200)]

```

12.3a. programaren kodean hainbat kontu azpimarra dezakegu:

- *brown* objektua inportatu da *nltk.corpus* kasetik. Eta objektu horren generoetako (kategoriak) testuetan *words* metodoa aplikatu dugu azpicorpus horietan hitzak lortzeko, hitz-mailako lanketa egin ahal izateko gero.

- Moduluko *nlk.FreqDist()* metodoaren bidez hitz-mailako maiztasunak lortu ditugu (bikote-zerrenda bat) eta zerrenda horren gainean *most_common()* metodoa aplikatu dugu maiztasun handieneko 10 hitzak lortzeko.
- Kontaketetan 6. kapituluaren azaldu den *comprehension* moduko esleipenak (zerrenda bati esleipen iteratiboa) egin dira kodea trinkotzeko.

Horrekin ez ditugu genero bakoitzeko berezitasunak ikusten. Hain ohikoak diren eta normalean esanahi txikia duten hitz horiek baztertu ohi dira azterketa lexikografikoetan eta "*stopword*" zerrenda bat osatu ohi da baztertu nahi diren hitzekin. NLTK-k ere *stopword*-zerrenda bat eskaintzen du. Alda dezagun, beraz, aurreko adibideko kodea, hitz horiek eta puntuazio-markak kontuan har ez ditzan.

12.3b. programan dago bertsio berria. Bide batez zerrendaren idazketa dotoreagoa egin dugu eta 10 aukeratu beharrean 30 aukeratu dugu.

```
import nltk, sys

#nlk-ren corpora inportatu: brown
from nltk.corpus import brown

#generoak jaso (ingelesez)
g1 = sys.argv[1]
g2 = sys.argv[2]

#lehen gaiaren corpora jaso hitz zerrenda gisa
corp1 = brown.words(categories=g1)
#bigarren generoaren corpora jaso hitz zerrenda gisa
corp2 = brown.words(categories=g2)

#ingelesezko stopwords-en corpora jaso hitz zerrenda gisa
stopwords = nltk.corpus.stopwords.words('english')

#corpusetan kontaketak egin bi baldintzak: alfabetoko karaktereekin
# osatuta egotea eta stopwordsetan ez agertzea.
corp1_kontaketa = nltk.FreqDist(h.lower() for h in corp1 if
                                h.isalpha() and h.lower() not in stopwords)
corp2_kontaketa = nltk.FreqDist(h.lower() for h in corp2 if
                                h.isalpha() and h.lower() not in stopwords)

maiz1=corp1_kontaketa.most_common(30)
maiz2=corp2_kontaketa.most_common(30)

print(g1+"-ren hitz ohikoenak:")
print(' '.join(str(p[0]) for p in maiz1))
print(g2+"-ren hitz ohikoenak:")
print(' '.join(str(p[0]) for p in maiz2))
```

12.3b. programa. Maiztasun handieneko hitz esanguratsuak.

Oraingoan bai, hitz esanguratsuagoak jaso ditugu irteeran. Berrien hitzen artean *"state"* edo *"president"* bezalako hitzak ikusten ditugun bitartean, zientzia-fikzioan *"mercer"*, *"ekstrohm"* edo *"helva"* hitzak ditugu.

```
>>>
python3 pl2-3b.py news science_fiction
news-ren hitz ohikoenak:
said would new one last two first state president year home also made time years three
house week may city could school four day committee man members back government many
science_fiction-ren hitz ohikoenak:
would could said one time helva ekstrohm mercer know like long people hal mike ship
back jack man first make never years see head mind made take well earth course
```

12.2.3. Ariketa: Lemak eta kategoriak

Tagger edo *POS tagger*, etiketatzaile esango diogu euskaraz, ezagutzen den tresna da, erabilienetako bat hizkuntza-prozesaketan. Tresna horren bitartez hitzaren kategoria gramatikala modu automatikoan lortzen da hitzaren testuingurua kontuan hartuta. Horrela euskarazko *"zuen"* hitzak bi etiketa edo kategoria izan ditzake testuinguruaren arabera: aditz laguntzailea (adibidez *"esan zuen"*) edo izenordaina (*"zuen kontura"*).

Brown Corpora erabilita 12.4. programan kategoria eta lema ohikoenak inprimatuko ditugu lehenik, adjektibo ohikoenak gero, eta, azkenik, parametro gisa sartzen dugun izenarekin konbinatzen diren adjektiboak (ingeleseko etiketatzaile honen arabera izenak "NN" etiketa dute eta adjektiboek "JJ" etiketa).

```
import nltk, sys

#nltk-ren corpora inportatu: brown
from nltk.corpus import brown

#izena jaso (ingelesez)
iz = sys.argv[1]

#corpus etiketatua jaso
corp_etik = brown.tagged_words(tagset='universal')

stopwords = nltk.corpus.stopwords.words('english')

#etiketen eta lemen kontaketa
etik_kontaketa = nltk.FreqDist(et for (h,et) in corp_etik if
                                h.isalpha() and h.lower() not in stopwords)
lema_kontaketa = nltk.FreqDist(h for (h,et) in corp_etik if
                                h.isalpha() and h.lower() not in stopwords)

maiz_et=etik_kontaketa.most_common(10)
maiz_lem=lema_kontaketa.most_common(10)
```

```

print("etiketa eta lema ohikoenak:")
print(maiz_et)
print(maiz_lem)

# adjektibo ohikoenak
adj_kontaketa = nltk.FreqDist(h for (h,et) in corp_etik if et == 'JJ')
maiz_adj=adj_kontaketa.most_common(10)
print("ADJ ohikoenak:")
print(maiz_adj)

# ADJ-IZE bigramak

bigr=dict()
for esaldi_et in brown.tagged_sents():
    for (h1,et1), (h2,et2) in nltk.bigrams(esaldi_et):
        if (et2=='NN' and h2==iz and et1=='JJ'):
            print(h1, h2)

```

12.4. programa. Lemak eta etiketak lantzen.

Programa exekutatzen badugu, eta parametro gisa "*house*" izena sartu, hauxe da exekuzioaren emaitzaren hasiera (denbora apur bat behar du):

```

>>>
python3 p12-4.py house
etiketa eta lema ohikoenak:
[('NOUN', 260583), ('VERB', 123176), ('ADJ', 73116), ('ADV', 31549), ('ADP', 8576),
 ('NUM', 7195), ('DET', 1683), ('X', 1209), ('PRT', 888), ('PRON', 816)]
[('one', 2873), ('would', 2677), ('said', 1943), ('could', 1580), ('time', 1556),
 ('two', 1311), ('may', 1292), ('first', 1242), ('like', 1237), ('man', 1151)]
ADJ ohikoenak:
[('other', 1627), ('new', 1060), ('first', 947), ('more', 941), ('such', 903),
 ('many', 896), ('own', 750), ('good', 693), ('same', 679), ('little', 673)]
two-family house
empty house
own house
small house
...

```

Zoritxarrez NLTKn ez dago integratuta euskararako halako etiketatzailerik (badaude baina beste programazio-lengoaia batzuetan; adib. hemen: <<http://ixa.eus/Ixa/Produktuak/1273217967>>).

Ondoko metodoen bitartez jakin daiteke zein hizkuntzatarako dauden eskura-garri hainbat tresna:

```

print(" ".join(nltk.stem.snowball.languages))
print(" ".join(nltk.corpus.stopwords.fileids()))

```

12.2.4. Ariketa: Antzeko hitzak lortzen

Batzuetan interesgarria da ortografikoki antzekoak diren hitzak lortzea, zuzentzaile ortografikoetan esaterako. Horretarako NLTK-k *edit_distance* metodoa eskaintzen du, edizio-distantzia oinarritzen dena. Distantzia horrek karaktere-aldaketak, ezabatzeak edo txertatzeak kontatzen ditu. Posible da karaktere kontsekutiboen arteko trukea ere kontuan hartzea (3. parametroan 1 jarritz). Beraz, bateko distantziara dauden hitzak oso antzekoak izango dira.

Proba egiteko, 12.5. programan *edit_distance* metodo hori erabiltzen dugu sartutako lema oker batetik gertuen dauden lemak (bateko distantzia daudenak) lortzeko. *Brown Corpus*etik bildutako lema-biltegia erabiltzen da bilaketa horretan.

Definitutako *bilatu_antza* funtzioan maiztasun handieneko lemen biltegia eta lema bat jasotzen dira parametro gisa, eta lema horrekin antzekotasun handiena duten biltegiko lemak lortzen dira.

```
import nltk, sys

#nltk-ren corpora inportatu: brown
from nltk.corpus import brown

def bilatu_antza(maiz_lem, lem):
    dmin = 10
    zerr = []
    for l,m in maiz_lem:
        d = nltk.edit_distance(l, lem, 1)
        if d == 1 :
            zerr.append(l)
    return zerr

# corpus etiketatua jaso
corp_etik = brown.tagged_words(tagset='universal')

# lemen kontaketa
lemak = [h for (h,et) in corp_etik if h.isalpha()]
lema_kontaketa = nltk.FreqDist(lemak)

# maiztasun handieneko 10.000 lema
maiz_lem = lema_kontaketa.most_common(10000)

# hitza eskatu eta ez badago corpusean antzekoenak lortu
lem = input("sartu lema bat (zuzen ez badago ere): ")
if lem not in lemak:
    z = bilatu_antza(maiz_lem, lem)
else:
    z = lem
print(z)
```

12.5. programa. Antzeko lemak lortzen.

Ondoren, proba baten emaitza dago:

```
>>>
python3 p12-5.py
sartu lema bat (zuzen ez badago ere): tken
Antzekoenak: ['then', 'taken', 'ten', 'token']
```

12.3. NLTK GURE TESTUEKIN ERABILTZEA

Aurreko kasu praktikoetan NLTKtik ekarritako corpusarekin lan egiten ikasi dugu. Oraingo honetan euskaraz idatzitako testu handi bat lortuko dugu eta testu horiek prozesatu beharko ditugu.

Testuen jabegoarekin arazorik ez edukitzeko, XVII. mendeko testu klasiko bat hautatu dugu: Pedro Agerriren (Axular) *Gero*. Armiarma guneari esker eskuragarri dugu (<http://klasikoak.armiarma.eus/idazlanak/A/AxularGero.htm>) RTF formatuan. Gure makinan gordetzean TXT formatuan gordetzeko esango diogu (*AxularGero.txt* fitxategia gure adibideetan).

Baina testu horretan gaur egungo euskara estandarrean ez dauden hitz asko agertzen dira. Kasu batzuetan estandarrez idatzitako testuak tratatu nahi izango ditugu. Lizentziarekin arazorik ez edukitzeko, Wikipediako testuetara jo dugu. Erabiliko dugun testua IXA taldearen zerbitzarian dago, Wikipediaren lizentzia berarekin: CC BY-SA. Helbide honetatik eskura dezakezu: <<http://ixa.eus/Ixa/Produktuak/1464941552>> (bertsio laburrena eskuratu eta *wiki_eu_labur.txt* izenarekin erabili da ariketetan).

12.3.1. Ariketa: Euskarazko n-gramen maiztasunak

Hizkuntzak modelatzeko (eta hizkuntzen arteko desberdintasunak kalkulatzeko) erabiltzen da n-gramen eredua. Testuetan elkarren segidan azaltzen diren n elementuzko konbinazio posible guztien maiztasunak kalkulatu dira horrelakoetan. n-ren balioa 1 eta 5 artean egon ohi da (handiagoak ere erabiltzen dira) eta normalean "gramak" edo elementuak karaktereak izaten dira (hitzen n-gramak ere erabiltzen dira beste batzuetan, ariketa honetan bezala).

Ariketa honetan kalkulatu nahi dira zeintzuk diren maiztasun handieneko 10 n-gramak euskararako. Zeintzuk dira euskaraz gehien errepikatzen diren 3 hitzezko sekuentziak?

Gero liburua izango da gure corpusa (ahal izanez gero, handiago eta eguneratuago bat erabiltzea gomendagarriagoa izan daiteke).

```

import nltk, sys, operator

#nltk-ren n-gramak
from nltk.util import ngrams

fitx = sys.argv[1]

# irakurri eta tokenizatu
f=open(fitx,'rU')
gordin=f.read() # unicode
tokenak = nltk.word_tokenize(gordin)
testu = nltk.Text(tokenak)
hitzak = [h.lower() for h in testu] #hitzak letra xehez

nhizt = dict()

n=3
# hizkuntz ereduak (hitzak)
ngramak = ngrams(hitzak,n)
for g in ngramak:
    try:
        if g in nhizt:
            nhizt[g] += 1
        else:
            nhizt[g] = 1
    except:
        continue

# inprimaketa sailkatuta. Lehen m ngramak (10)
n = 0
m = 10
for ng in sorted(nhizt, key=nhizt.get, reverse=True):
    print(ng, nhizt[ng])
    n = n+1
    if n == m: # bakarrik lehen 10ak
        break

```

12.6. programa. n-gramen sorkuntza eta erabilera.

12.6. programan NLTK moduluko *ngrams* metodoa erabiltzeaz gain hiztegi bateko sarrerak nola ordena daitezkeen ikus dezakegu. Aurretik azaldutako *sorted* metodoa erabili da hemen ere, baina hiztegi batean erabili ahal izateko bigarren parametro bat zehaztu behar izan dugu (*key*). Hiztegiaren gainean *get* metodoa aplikatuta balioaren arabera ordenatzea lortzen da. Horrez gain, *reverse* ere zehaztu dugu maiztasun handieneko n-gramak eskuratzeko, ez maiztasun txikieneoak. Bukatzeko, hitzak letra xehetara pasa ditugu (*lower* metodoaren bitartez) trigramak sortu baino lehen, horrela hitz bera ez da desberdin moduan tratatuko larri/xehe moduan egoteagatik.

Ariketa probatzeko aipatutako testua erabili dugu:

```
python3 p12-6.py AxularGero.txt
```

Lortzen den emaitza hau da:

```
(' ', 'eta', 'ez') 320
(' ', 'eta', 'hala') 88
('erraiten', 'du', 'san') 84
(' ', 'eta', 'bai') 78
(' ', 'lib', ' ') 62
(' ', 'kap', ' ') 60
('da', ' ', 'eta') 60
(' ', 'hala', 'erraiten') 60
('dioen', 'bezala', ':') 56
('2', ' ', ' ') 56
```

Ikusten denez, puntuazio-marka asko daude. Testuz osatutako hirukoteak interesgarriagoak izan daitezke. Ariketa gisa proposatzen da (ikus kapitulu bukaeran)

12.3.2. Ariketa: Zuzentzaile ortografiko sinple bat

Aurreko ariketetan ikasitakoarekin nahiko erraza gertatuko zaigu zuzentzaile ortografiko sinple bat programatzea. Wikipediako testutik abiatuko gara eta suposatuko dugu bertan idatzitako hitz guztiak ondo daudela. Hitz horiek zerrenda batean metatuko ditugu (letra xehez), eta zuzentzaile ortografikoak ondokoa egingo du: aztertzen duen hitza zerrendan badago, ontzat joko du, eta bestela, alfabetikoa bada, antzekoenak bilatzen saiatuko da. 12.7. programan dago kodea.

```
import nltk, sys, operator

#nltk-ren n-gramak
from nltk.util import ngrams

# 12.5 ariketatik eratorria
def bilatu_antza(hzerr, hitz):
    zerr = []
    for h in hzerr:
        d = nltk.edit_distance(h, hitz, 1)
        if d == 1 :
            zerr.append(h)
    return zerr

fitx1 = sys.argv[1]    # corpusa
fitx2 = sys.argv[2]    # zuzentzeko testua

# irakurri eta tokenizatu (12.6 ariketatik hartuta)
f1=open(fitx1,'rU')
gordin=f1.read()
tokenak = nltk.word_tokenize(gordin)
```

```

testu = nltk.Text(tokenak)
hitzguztia = [h.lower() for h in testu      #hitzak letra xehez
               if h.isalpha()]
hitzak=set(hitzguztia)

f2=open(fitx2,'rU')
gordin2=f2.read()
tokenak2 = nltk.word_tokenize(gordin2)
testu2 = nltk.Text(tokenak2)
hitzak2 = [h.lower() for h in testu2      #hitzak letra xehez
            if h.isalpha()]

for h in hitzak2:
    if h not in hitzak:
        print(h, bilatu_antza(hitzak,h))
    else:
        print(h)

```

12.7. programa. Euskarazko zuzentzaile simple bat.

Exekutatzeko komandoa eta emaitzak jarraian ikus daitezke. Bi fitxategi erabiltzen dira, corpusa eta zuzentzeko testua.

```

>>>
python3 p12-7.py wiki_eu_labur.txt proba_zuzen
hau
proba
bat
da
hauek
ondo
daude
baiina ['baina', 'baiona']
haek ['hark', 'haiek', 'hašek', 'hauek']
ezz ['erz', 'ez', 'ezb']

```

12.3.3. Ariketa: Zuzentzaile ortografiko oso bat integratzen

Aurreko zuzentzailea, hala ere, ahula da, maiztasun txikiko hitzak (*egondakoarekin, ibilgailua*) ez-zuzentzat hartzen ditu eta, gainera, modu ez koherentean lan egiten du (*kalean* onartzen du baina *kalea* ez). Alde hori hobe daiteke corpus handiago batekin, baina beti muga batekin. Alde hori, eta aurreko ariketan aipatu dena, hobetzeko morfologia landu behar da, bai morfologian oinarritutako programa bat eginda (nahiko konplexua dena) bai eskura dagoen bat erabilita. *hunspell* programa erabiltzea da horretan erosoena, euskararako eskura dago-eta. *Mozillan* eta *Libreofficen* integratuta dagoen bera da. Gainera, software librea da.

Hunspell erabiliz hainbat hizkuntzatarako zuzentzaileak (eta analizatzaile morfologikoak) garatu dira. Kodeaz gain hizkuntza bakoitzeko bi fitxategi behar dira: lehen fitxategia (*.dic* luzapena duena) eta atzizkien fitxategia (*.aff* luzapeneko). Fitxategi horiek eskuratzeko jo helbide honetara: <http://xuxen.eus/eu/bertsioak#tab_hunspell> (*eu_ES.dic* eta *eu_ES.aff* fitxategiak).

Programa instalatzeko ondoko komandoa exekutatu behar da:

```
sudo apt-get install python3-hunspell
```

Modulua instalatuta eta aipatutako bi fitxategiak eskuratuta, aurreko ariketa dezente sinplifikatzen da eta, gainera, askoz hobea da lortzen den zuzentzailea. Programa sinpleagoa da, *spell* metodoaz hitzaren zuzentasuna egiazta daitekeelako, eta *suggest* metodoaz proposamenak lortu.

```
import nltk, sys, operator

import hunspell

#klasea kargatu
zuzentz = hunspell.HunSpell('eu_ES.dic', 'eu_ES.aff')

fitx = sys.argv[1] # zuzentzeko testua

f=open(fitx,'rU')
gordin=f.read()
tokenak = nltk.word_tokenize(gordin)
testu = nltk.Text(tokenak)
hitzak = [h for h in testu if h.isalpha()]

for h in hitzak:
    #egiaztapena
    if not zuzentz.spell(h):
        #proposamena
        print("OKER:", h, "PROPO:", zuzentz.suggest(h))
    #lema
    else:
        print("ZUZEN:", h, "LEMA:", zuzentz.stem(h)[0])
```

12.8. programa. Euskarazko zuzenketa *hunspell* bitartez.

Exekutatzeko komandoa eta emaitzak jarraian ikus daitezke. Bi fitxategi erabiltzen dira, *corpusa* eta *zuzentzeko testua*.

```

>>>
python3 p12-8.py proba_zuzen 2>tmp1
ZUZEN: Hau LEMA: hau
ZUZEN: proba LEMA: proba
ZUZEN: bat LEMA: bat
ZUZEN: da LEMA: da
ZUZEN: hauek LEMA: hauek
ZUZEN: ondo LEMA: ondo
ZUZEN: daude LEMA: daude
OKER: baiina PROPO: ['baiona', 'baina', 'bazina', 'baiena', 'balina', 'bagina',
'Bariaina', 'Baiona', 'ainarba']
OKER: haek PROPO: ['hark', 'hasek', 'haxek', 'haiek', 'hauek', 'hagek', 'habek',
'Saek', 'HAeek', 'EHAek', 'EHAeek']
OKER: ezz PROPO: ['zez', 'ez', 'ezaz', 'ezez', 'eziz', 'ezoz', 'eza', 'eze', 'ezi']
ZUZEN: flexionatuak LEMA: flexionatu
ZUZEN: etxekoa LEMA: etxeko
ZUZEN: etxetik LEMA: etxe
ZUZEN: etxekoarekikoak LEMA: etxeko

```

Adibidean ikus daitekeenez³⁷, egiaztapenaz zein proposamenen eskaeraz gain beste eragiketa garrantzitsu bat egin daiteke: lematizazioa edo *stemming*. Horretarako *stem* metodoa erabili ohi da³⁸.

12.4. PROPOSATUTAKO ARIKETAK

1) 12.1. programaren bigarren bertsio bat egin behar duzu *ts*, *tx* eta *tz* fonemak bilduta kontatzeko, jokoaren fitxetan fonema horiek agertzen baitira.

2) Egin aurreko ariketaren hedapen bat, karaktere solteak kontatu beharrean karakterezko n-gramak kontatzeko. n-ren balioa parametro gisa jasoko da. Hitzen hasieran eta bukaeran zuriune bat gehituko da hitz-hasierako eta hitz-bukaerako n-gramak ere bereizi ahal izateko.

3) Osatu 12.2b. programa honako ezaugarri hauekin:

- Esteken bitartez Internetetik jasotako webgune bakoitzarekin testu-corpus bat osatu, eta 12.1. programan kalkulatu diren probabilitateak kalkulatu testu guztien bildumarekin.
- Lortutako emaitza horiek eta beste corpusetik lortutakoak konparatu eta kalkulatu desbiderapen tipikoa.

37. 2> *tmp* gehitu da fitxategiek arazo bat ematen dutelako ñ-ren UTF kodeketan.

38. Eragiketa gehiago egiteko beste metodo batzuk ere badaude: <<https://github.com/blatinier/pyhunsPELL>>.

4) Egin 12.4. programan aldaketak, kategoria eta hitz desberdinetako adibideak lortzeko. Adibidez, lortu zeintzuk diren maizen agertzen diren izen-izen bikoteak eta zeintzuk diren *small* eta *big* hitzekin gehien agertzen diren izenak. Euskararako, hitzekin lan eginda, lortu zeintzuk diren *polita* eta *itsusia* hitzen aurretik maizen agertzen diren hitzak.

5) 12.6. programa hobetu behar duzu hizkuntza-eredutik lortzen diren trigramak (3 hitzeko sekuentziak) soilik hitzez osatuta egon daitezen (ez zenbakiak, ez puntuazioa). Horretarako, funtzio bat definitzea eta erabiltzea gomendatzen da. Kapitulu honetan erabilitako *isalpha* metodoa erabil daiteke horretarako, baina kontuan hartu behar da hiztegiako elementuak zerrendak direla eta metodoa zerrendako elementuei (karaktare-kateei) aplikatu behar zaiela.

6) 12.7. programa hobetu behar duzu hiru aldetatik:

- Batetik, proposamen ez-zuzenen kopurua murriz liteke, ontzat hartzeko corpusean bi aldiz agertzea eskatuko bagenu (edo aurretik corpus osoa ohiko zuzentzaile batetik pasako bagenu).
- Bestetik, erabili behar dugun bakoitzean corpora kargatzea ez da ideia ona eraginkortasunaren aldetik, beraz, programa bitan banatu behar da: batetik, hitz zuzenen zerrenda osatuko da eta biltegitratuko da (*pickled* paketea erabiltzea gomendatzen da). Bigarren programak corpora eskuratu beharrean biltegitratutako zerrenda erabiliko du.
- Proposamenak hitzaren maiztasunaren arabera ematea (maiztasun handienekoak aurretik).

7) *hunspell* erabilita lematiza ezazu Wikipediako corpora eta lor itzazu maiztasun handieneko lemak eta lema-bikoteak.

8) Seigarren ariketa hobetu behar duzu proposamenak testuinguruaren arabera emateko. Horretarako, lehenik eta behin euskararen hitz-hirukoteen hizkuntza-eredu bat sortu behar duzu. Gero, zuzentzeko proposamenak jaso eta gero, aurreko eta ondoko hitzak (edo aurreko biak) kontuan hartuta, aukeratu behar duzu zein den probabilitate handienekoa hizkuntza-ereduaren arabera.

Informatika sailean argitaratu diren beste liburu batzuk

Algoritmika

Rosa Arruabarrena
1997an argitaratua
ISBN: 84-86967-82-1

Ordenadore bidezko irudigintza

Joseba Makazaga, Asier Lasa
1998an argitaratua
ISBN: 84-86967-90-2

Oinarrizko programazioa. Ariketa-bilduma

Arantza Diaz de Illaraza, Kepa Sarasola
1999an argitaratua
ISBN: 84-8438-002-5

Zirkuitu elektriko eta elektronikoen oinarrizko analisia

Olatz Arbelaiz, Txelo Ruiz
2001ean argitaratua
ISBN: 84-8438-018-1

LINUX Sistemaren eta sarearen administrazioa

Iñaki Alegria
2003an argitaratua
ISBN: 84-8438-040-8

Sistema Digitalen Diseinu-hastapenak. Oinarrizko kontzeptuak eta adibideak

Olatz Arbelaiz eta beste
2005ean argitaratua
ISBN: 84-8438-069-6

Softwarearen ingeniariatza [I. atala: Softwarearen garapenaren zenbait arlo]

Jose Ramon Zubizarreta
2006an argitaratua
ISBN: 84-8438-085-8

Softwarearen ingeniariatza [II. ATALA: Garapen monolitikotik hiru mailako arkitekturara bezero/zerbitzariak bisitatuz]

Jose Ramon Zubizarreta

2009an argitaratua

ISBN: 978-84-8438-165-5

Linux: Sistemaren eta sarearen administrazioa 2. argitaraldia (Debian eta Ubuntu)

Iñaki Alegria eta Roberto Cortiñas

2008an argitaratua

ISBN: 978-84-8438-178-5

TAPE Testu-analisirako Perl erremintak

Aitzol Astigarraga eta beste

2009an argitaratua

ISBN: 978-84-8438-233-1

TCP/IP Sareak (3. argitaraldia)

Jose M^a Ribadeneyra Sicilia

2009an argitaratua

ISBN: 978-84-8438-235-5

Robot mugikorak. Oinarriak

Aitzol Astigarraga eta Elena Lazkano

2011n argitaratua

ISBN: 978-84-8438-377-2

Programen Espezifikazio, Egiartzapen eta Eratopen Formala

Javier Álvez, Xabier Arregi, Jose Gaintzarain, Paqui Lucio, Montse Maritxalar

2016an argitaratua

ISBN: 978-84-8438-590-5