

Django ingurunea

Python



Python lengoaia

- **Zer da?**

- Doako script-lengoaia bat, objektuei orientatua.
- Tcl eta Perl-ekin konparatu ohi da, baina desberdina da.

- **Historia**

- *Guido van Rossum*-ek sortua, 1990eko hamarkadaren hasieran.
- Izena *Monty Python* taldetik datorkio.
- Eraginak: ABC, Modula-2, Lisp, C eta shell scripting-a.

- **Non eskuratu**

- <http://www.python.org>
- Linux banaketa gehienetan dator.
- Bertsioak Unix, Windows eta Macintosh-erako.
 - *IronPython*: .Net-erako bertsioa, C#-ez idatzia (*Python for .NET* ere badago)
 - *Jython*: Java-z idatzia (<http://www.jython.org>).
 - *PyPy* (Python-ez idatzia, esperimentatzeko-eta ezaugarri aurreratuak dituela...)



Aurkezpenaren eskema

- Ikuspegi orokor eta laburra
- Alderdi lexikala eta sintaktikoa, gainetik
- Motak eta objektuak
- Eragileak eta espresioak
- Kontrola
- Funtzioak
- Klaseak eta objektuei orientatutako programazioa
- Moduluak eta paketeak
- Sarrera/irteera
- Liburutegi-modulu erabilgarriak



Hasierakoak

- Unix-en: `$ python`
- Windows-en eta Mac-etan: aplikazioa abiarazi eta interpretatzailearen *prompt*-a agertuko zaigu

```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC  
v.1500 64 bit (AMD64)] on win32
```

```
Type "help", "copyright", "credits" or "license" for more  
information.
```

```
>>>
```

- Interpretatzailea abiarazita: programa interaktiboki sartzen has gaitezke.
- Azken bertsio egonkorak:
 - Python 3.6.0 (2016eko abendua)
 - Python 2.7.13 (2016eko azaroa)



Lehen programatxoak

- ***Kaixo, hi!***

```
>>> print "Kaixo, hi!"
Kaixo, hi!
>>>
```

- **Kalkulagailu gisa:**

```
>>> 3*4.5 + 5
18.5
>>>
```

- Modu interaktibo hau, funtsean, irakurri-eta-ebaluaratu begizta sinple bat da.

- **Indentazioa (ez giltzarik, ez *begin/end* blokerik):**

```
>>> for i in range(0,3):
...     print i
...
0
1
2
>>>
```



Programak eta fitxategiak

- Programak, normalean, *.py* fitxategietan idazten dira.
- Unix-en, programa-fitxategia argumentu gisa pasatuko diogu interpretatzaileari:

```
$ python helloworld.py
```

```
Hello World unix
```

```
$
```

- Edo, exekuzio-baimena emanaz gero:

```
$ ./helloworld.py
```

```
Hello World unix
```

```
$
```

- Windows-en, klik bikoitza.



Aldagaiak eta espresioak

◦ **Espresioak**

- Ohiko eragile aritmetikoak, beste lengoaietan bezalaxe:

```
3 + 5
```

```
3 + (5*4)
```

```
3 ** 2
```

```
'Kaixo,' + 'hi!'
```

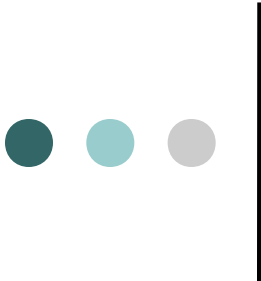
◦ **Asignazioa**

```
b = a * 4.5
```

```
c = (a+b) / 2.5
```

```
a = "Kaixo, hi"
```

- Aldagaiak mota dinamikoki hartzen dute (mota espliziturik ez; mota aldatzen ahal da exekuzioan).
- Aldagaiak objektu baten izena baino ez dira (ez daude memoria-gune bati lotuak).



Oinarrizko motak: zenbakiak eta string-ak

○ Zenbakiak

```
>>> a = 3 # Osoa
>>> b = 4.5 # Errealala (koma higikorra)
>>> c = 5172888333333L # Oso luzea
>>> d = 4 + 3j # Konplexua
```

○ String-ak

```
>>> a = 'Hello' # komatxo bakunak
>>> b = "World" # komatxo bikoitzak
>>> c = "Bob said 'hey there.'" # Bietakoak nahastuta
>>> d = '''Hiru komatxo jarritz gero, string-a lerro batetik
bestera pasatzeko arazorik ez dago'''
>>> e = """Berdin egin daiteke komatxo bikoitzekin, hurrengo
lerrora luzatuz"""
```




Oinarrizko motak: listak []

- **Edozein motatako osagaiz osatuak**

```
>>> a = [2, 3, 4] # Zenbaki osozko lista
>>> b = [2, 7, 3.5, "Hitza"] # Lista konbinatua
>>> c = [] # Lista hutsa
>>> d = [2, [a,b]] # Osagaitzat lista bat duen lista
>>> e = a + b # Bi listaren kateadura
```

- **Listetako osagaien atzipena**

```
>>> x = a[1] # Bigarren osagaia eskuratu (0.a da lehena)
>>> y = b[1:3] # Azpilista lortu: y = [7, 3.5], kontuz!
>>> z = d[1][0][2] # Lista habiatuen barruko atzipena: z = 4
>>> b[0] = 42 # Osagai bat aldatu
```

- **Listen metodoak**

```
>>> a.append("zzz") # Erantsi osagai bat
>>> a.insert(1,"xxx") # Txertatu osagai bat 1. posizioan
>>> len(a) # Listaren luzera
>>> del a[2] # Ezabatu osagai bat
```



Oinarrizko motak: tuplak ()

○ **Tuplak**

```
>>> f = (2,3,4,5) # zenbaki osozkoa  
>>> g = (1,) # osagai bakarrekoa  
>>> h = (2, [3,4], (10,11,12)) # era askotako objektuak osagai
```

○ **Tuplen erabilera**

```
>>> x = f[1] # Osagaien atzipena: x = 3  
>>> y = f[1:3] # Atal edo "xerrak": y = (3,4)  
>>> z = h[1][1] # Egitura habiatuetako atzipena: z = 4
```

○ **Oharrak:**

- Tuplak listen antzekoak dira, baina tamaina finkoa dute sorreratik.
- Ezin dira osagaiak ordezkatu.
- Eztabaidagai da tuplen beharrik ba ote dagoen.



Oinarrizko motak: hiztegiak { }

- **Hiztegiak** (array asoziatiboak: gako-balio bikotez osatuak)

```
>>> a = { } # hiztegi hutsa
>>> b = { 'x': 3, 'y': 4 }
>>> c = { 'uid': 105, 'login': 'e08', 'name' : 'Ane' }
```

- **Osagaien atzipena**

```
>>> u = c['uid'] # osagai bat atzitu: u = 105
>>> c['passw'] = "b120321" # osagai bat gehitu
>>> if c.has_key("directory"): # osagai bat dagoenez egiaztatu
...     d = c['directory']
...     else:
...         d = None
>>> d = c.get("directory",None) # Gauza bera, trinkoago
>>> k = c.keys()
...     # gako guztiak lista batean: ['login', 'passw', 'uid', 'name']
>>> l = c.values()
...     # balio guztiak lista batean: ['e08', 'b120321', 105, 'Ane']
```



Baldintzak

- **if-else**

```
# a eta b-ren arteko handiena gorde z-n:  
>>> if a < b: z = b  
... else: z = a  
...
```

- **pass sententzia**

```
>>> if a < b: pass # ez egin ezer  
... else:  
...     z = a  
...
```

- **Oharrak:**

- Indentazioa da *then/else* gorputzak adierazteko modua.
- *pass*-ek gorputz hutsa adierazten du.



Baldintzak (II)

- **elif sententzia**

```
>>> if a == '+':  
...     erag = 'GEHI'  
... elif a == '-':  
...     erag = 'KEN'  
... elif a == '*':  
...     erag = 'BIDER'  
... else:  
...     erag = 'EZEZAGUNA'
```

- Ez dago *switch* edo *case*-ren parekorik.

- **Espresio boolearrak: and, or eta not**

```
>>> if b >= a and b <= c:  
...     print "b a eta c-ren artean dago"  
>>> if not (b < a or b > c):  
...     print "b a eta c-ren artean dago"
```



Begiztak

- **while sententzia**

```
>>> while a < b:  
...     a = a + 1
```

- **for sententzia (sekuentzia baten osagaien gainean)**

```
>>> for i in [3, 4, 10, 25]:  
...     print i  
>>> for c in "Hello World":  
...     print c  
>>> for i in range(0,100):  
...     print i
```



Funtzioak

- **def sententzia**

```
# a/b-ren hondarra
>>> def hondarra(a,b):
...     q = a/b
...     r = a - q*b
...     return r
...
>>> a = hondarra(42,5) # a = 2 (funtzio-deia)
```

- **Balio anitz itzuli nahi izanez gero:**

```
>>> def zatitu(a,b):
...     q = a/b
...     r = a - q*b
...     return q,r
...
>>> x,y = zatitu(42,5) # x = 8, y = 2
```

- **Aldagai lokalak eta globalak:**

```
# global gako-hitza erabiltzen da funtzioaren barneko aldagaia globala dela adierazteko
>>> def f():
...     global s
...     print s
...     s = "barruan."
...     print s

s = "globala."
f()
print s
```



Funtzioak (II)

- Python-en, parametro guztiak erreferentziazkoak dira

```
>>> def f(l):  
...     l.append(3)  
...  
>>> a = [1,2]  
>>> f(a)  
>>> print a # a → [1,2,3]
```

- Baina parametroaren izen berdineko aldagai lokal bat definitzean, parametroa ez da balioz aldatzen:

```
>>> def f(l):  
...     l = [1,2,3]  
>>> a = [1,2]  
>>> f(a)  
>>> print a # a → [1,2]
```




Funtzioak (III): parametro-pasea

- **Parametro-pasea:** funtzioari deia egiterakoan, argumentuak adierazterako modu desberdinak daude:

- Beharrezko argumentuak: funtzioaren burukoan agertzen diren bezala

```
>>> def hondarra(a,b):  
...     q = a/b  
...     r = a - q*b  
...     return r  
...  
>>> a = hondarra(42,5) # a = 2
```

- Gako-hitz argumentuak: parametroaren izenak erabiliz identifikatzen dira

```
>>> def hondarra(a,b):  
...     q = a/b  
...     r = a - q*b  
...     return r  
...  
>>> a = hondarra(b=5, a=42) # a = 2
```

- Argumentu lehentsiak: funtzioa deia egitean argumentua adierazten ez bada, lehentsitako balio bat erabiltzen da

```
>>> def hondarra(a,b = 5):  
...     q = a/b  
...     r = a - q*b  
...     return r  
...  
>>> a = hondarra(42) # a = 2
```



Klaseak

- **class sententzia, klaseak definitzeko**

```
>>> class Kontua:
...     def __init__(self, hasierakoa):
...         self.saldoa = hasierakoa
...     def sartu(self, zenbatekoa):
...         self.saldoa = self.saldoa + zenbatekoa
...     def atera(self, zenbatekoa):
...         self.saldoa = self.saldoa - zenbatekoa
...     def saldoalortu(self):
...         return self.saldoa
```

- **Klase baten erabilera**

```
>>> a = Kontua(1000.00)  #eraikuntza: klasearen instantzia
>>> a.sartu(550.23)
>>> a.sartu(100)
>>> a.atera(50)
>>> print a.saldoalortu()
```



Klaseak (II)

- **Kontua klase hori kontuklasea moduluan definitua balego:**

```
>>> import kontuklasea
>>> a=kontuklasea.Kontua(345.80) #Kontua horrela idatzi behar da, maiuskula eta guzti!
>>> print a.saldoalortu() #hemen, berriz, ez da moduluaren aipamenik egin behar!
345.8
>>>
```

- **`__init__`: metodo pribatua eta sistemak definitua, klasearen eraikitzailea-edo.**

- Metodo pribatuek aurreko bi *underscore*-ak eramaten dituzte
- Sistemak definitutakoek aurretik bi *underscore*, eta atzetik beste bi (*magic* deitzen zaie)

- **saldoa atributu edo eremua ez da pribatua (!!!), eta kanpotik atzi daiteke (metodora jo beharrik gabe).**

- Non erazagutzen da? Inon ez!, aldagaiak ez dira erazagutzen.

- **Herentzia:**

- `class K (classA)` # K klaseak "classA" klase gurasoa du.
- `class K (classA, classB, classC)` # K klaseak "classA", "classB" eta "classC" gurasoak ditu.



Salbuespenak

- **try sententzia**

```
>>> try:
...     f = open("fitx.txt")
... except IOError:
...     print "Ezin da ireki fitx.txt fitxategia"
```

- **raise sententzia**

```
>>> def faktoriala(n):
...     if n < 0:
...         raise ValueError, "Zenbaki positiboa edo 0 behar du"
...     if (n <= 1):
...         return 1
...     else:
...         return n*faktoriala(n-1)
```

- **Harrapatu gabeko salbuespenak**

```
>>> faktoriala(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 3, in faktoriala
ValueError: Zenbaki positiboa edo 0 behar du
```



Fitxategiak

- **open() funtzioa**

```
>>> g = open("C:/boot.ini","r") # Ireki fitxategia, bertatik irakurtzeko
>>> f = open("C:/bidertaula.txt","w") # Ireki fitxategia, bertan idazteko
```

- **Nola irakurri eta idatzi**

```
>>> datuak = g.read() # Dena irakurri
>>> lerroa = g.readline() # Irakurri lerro bakarra
>>> lerroak = g.readlines() # Irakurri datuak, lerro-lista bat eratuz
>>> f.write("Biren taula\n\n")
```

- **Formatua eman irteerari: % eragilea (C-ko *printf*-ren antzera)**

```
>>> for i in range(1,11):
...     f.write("%d bider bi = %d\n" % (i, 2*i))
>>> f.close()
```



Moduluak

- **Programa handiak modulutan bana daitezke**

```
# zenbakiak.py modulua
def zatitu(a,b):
    q = a/b
    r = a - q*b
    return q,r
def zkh(x,y):
    z = y
    while x > 0:
        z = x
        x = y % x
        y = z
    return z
```

- **import sententzia**

```
>>> import zenbakiak
>>> x,y = zenbakiak.zatitu(42,5)
>>> n = zenbakiak.zkh(7291823, 5683)
```

- import-ek izen-espazio (*namespace*) bat sortzen du



Python liburutegia

- **Python-ekin batera modulu estandarrez osatutako liburutegi bat dator**
 - String-en tratamendua
 - Sistema eragileen interfazeak
 - Sareak
 - Exekuzio-hariak
 - Erabiltzaile-interfaze grafikoak (GUIak)
 - Datu-baseak
 - Hizkuntza-zerbitzuak
 - Segurtasuna
- **Eta bestek egindako hainbat modulu ere badira:**
 - XML
 - Zenbakizko prozesamendua
 - Plotting-a eta grafikoak
 - etab.
- **import besterik ez da behar horietako edozein eskura izateko:**

```
>>> import string
>>> a = string.split(x)
```



Zertan erabiltzen da?

- Beste lengoaia batzuk erabiliz garatutako aplikazioen automatizazioan, scripting lengoaia gisa
- Multimedia-aplikazioetan
- Jokoen programazioan
- Wiki-motorrak garatzeko
- Web-edukien kudeaketarako plataformetan
- Banaketa-zerrenden kudeaketan
- Scripting bidez garatzeko egokiak diren aplikazioetan, oro har



Bukatzeko...

- Python nahiko lengoaia sinplea da.
- Ezer asko jakin gabe has daiteke "gauzatxoak" egiten.
- Kodea irakurgarria da.
- Ingurunea aproposa da probak eginez ikasteko.

- <http://www.python.org>
- <http://www.tutorialspoint.com/python>
- <https://www.python.org/dev/peps/pep-0008/> (estilo-gida)



1 ariketa

- `datetime` paketea erabiliz:
 - gaur zein astegun den idazten du.
 - Data bat eskatu (uuuu/hh/ee formatuan), eta zein astegun den adierazten du.
 - Aurrekoa bezala, baina asteguna euskaraz ematen du.
 - Jaiotze-data bat eskatu, eta idazten du: zenbat urte dituen egun horretan jaiotakoak eta zenbat egun, ordu, minutu eta segundo falta zaizkion hurrengo urtebetetzerako.



2. ariketa

- `Motxila` izeneko klasea sortu eta erabili.
 - Klase horrek, metodo eraikitzaileaz gain, `__str__` (edukiak erakusteko) eta `sartu` metodoak izango ditu.
 - `sartu` metodoak objektu bat sartzen du motxilan.
 - `Motxila` klasea honela erabili:

```
m2 = Motxila()
m3 = Motxila()
m2.sartu('bokadilloa')
m2.sartu('giltza sorta')
```

```
print m2
```

```
print m3
```



3. ariketa

- Hiztegia izeneko klasea sortu eta erabili.
 - Klase hori eu_en hiztegi elebidunak errepresentatzeko erabiltzen da:
 - Hiztegia errepresentatzeko *dictionary* objektu bat erabiltzen da, non gakoak jatorrizko hitzak diren eta elementuak, berriz, gakoen itzulpenak.
 - `__init__` eta `__str__` metodoez gain, beste hauek ere izango ditu:
 - `def kargatu(self, fitxategia, jatorria):`
 - `fitxategia`: string motako parametro bat da, eta testu-fitxategi baten izena adierazten du.
 - Testu-fitxategiak honako egitura du: lerro bakoitzean bi hitz, zuriunez bereizita, non lehenengoa euskarazkoa den eta bigarrena bere ingelesezko ordaina
 - `jatorria`: “eu” edo “en”
 - `def kontsultatu (self, hitza):`
 - `hitza` argumentuaren ordaina itzultzen du hiztegi horretan
 - string hutsa itzuliko du hitza hiztegian ez badago