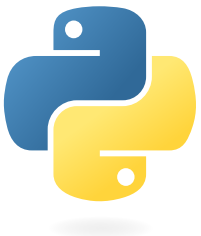
Llenguatges de Programació

Sessió 1: Python bàsic i funcional



python™

Gerard Escudero i Albert Rubio

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Contingut

Introducció

Paradigmes:

- imperatiu,
- orientat a objectes,
- funcional.

Característiques:

- interpretat,
- llegibilitat,
- *lazyness*.

Té una gran quantitat de llibreries disponibles.

Entorns

'Hello world!'

```
print('Hello world!')
```

```
nom = input('Com et dius? ')
print('Hola', nom + '!')
```

Línia de comandes:

`python3 script.py`

idle: entorn Python 3

`idle` o `idle3`

Codificació:

Utilitza 'utf-8'

Blocs i comentaris

Comentaris

```
# això és un comentari
```

Blocs

Els blocs (*suites*) es marquen per l'identació.

```
if condicio:
    print('ok!')
```

L'estil estàndard *PEP8* es marcar-ho amb 4 espais.

Línies llargues

Podem tallar línies amb `\`:

```
total = x*100 + \
        y*10
```

Variables i assignació

Declaració implícita (valor):

```
a = 2
```

Assignació:

```
x = y = z = 0
x, y, z = 0, 5, "Llenguatges"
x, y = y, x
```

Assignació augmentada:

```
a += 3
```

⚠ No té ni `--` ni `++`!

Tipus estàndard

Nombres

`int`, `float`, `complex`

Són tipus i funcions de conversió.

`int` no té rang. Pot tractar nombres arbitràriament llargs.

Operadors usuals excepte: `**` (potència) i `//` (divisió entera)

Booleans

`True`, `False`

Operadors: `and`, `or`, `not`

Funcions de tipus

```
type(3)  👉  int

isinstance(3, int)  👉  True

isinstance(3, (float, bool))  👉  False
```

Condicionals

Acció (*statement*):

```
if x < 0:
    signe = -1
elif x > 0:
    signe = 1
else:
    signe = 0
```

Els `else` i `elif` són opcionals.

Expressió:

```
x = 'parell' if 5 % 2 == 0 else 'senar'
```

Iteracions

Taula de multiplicar:

while

```
n, i = int(input('n? ')), 1
while i <= 10:
    print(n, 'x', i, '=', n * i)
    i += 1
```

for

```
n = int(input('n? '))
for i in range(1, 11):
    print(n, 'x', i, '=', n * i)
```

El `for` funciona sobre tipus *iterables*.

També podem usar el `break` i el `continue`, amb la semàntica usual sobre tots dos bucles.

Funcions I

Declaració:

```
def primer(n):
    for d in range(2, n // 2 + 1):
        if n % d == 0:
            return False
    return True
```

Crida:

```
primer(5)  👉  True
```

Retorn de múltiples valors:

```
def divisio(a, b):
    return a // b, a % b

x, y = divisio(7, 2)  # x 👉 3, y 👉 1
```

Quan tornem més d'un valor ho fa internament en forma de *tupla*.

Funcions II

Valors per defecte:

```
from string import punctuation

def remPunc(s, tl=True):
    rt = ''
    for c in s.lower() if tl else s:
        if c not in punctuation:
            rt = rt + c
    return rt

remPunc('Hola, sóc un exemple!')
👉  'hola sóc un exemple'

remPunc('Hola, sóc un exemple!', tl=False)
👉  'Hola sóc un exemple'
```

Contingut

- Elements bàsics
- Iterables
- Part funcional
- Exercicis

Strings I

Tipus `str`.

Els *strings* són iterables.

Operacions:

```
z = 'Llenguatges'

z[2]    👉  'e'          # posició

z[3:6]  👉  'ngu'        # subcadena

z[:4]   👉  'Llen'       # prefix

z[8:]   👉  'ges'        # sufix

z + ' Programació'   👉  'Llenguatges Programació'   # concatenar

z * 3   👉  'LlenguatgesLlenguatgesLlenguatges'      # repetir

len(z)  👉  11           # mida
```

Strings II

Altres operacions i mètodes

```
z = 'Llenguatges'

'ng' in z  👉  True

'ng' not in z  👉  False

z.find('ng')  👉  3              # cerca i torna posició

z.count('e')  👉  2              # comptar

'Hello world!\n'.strip()  👉  'Hello world!'   # treu el \n

'1,2,3'.split(',')  👉  ['1', '2', '3']    # parteix un string

','.join(['1', '2', '3'])  👉  '1,2,3'     # operació inversa
```

Els *strings* són *immutables*:

```
z[0] = 'l'  ❌   # TypeError: 'str' object does not support item assignment
```

Llistes

Les llistes (`list`) són heterogènies:

```
z = ["hola",5,"llenguatge",6.63,2]
```

Tenen les mateixes operacions que els *strings* i també són *iterables*.

Per recórrer dos o més iterables podem utilitzar el `zip`:

```
def prodEscalar(v, w):
    res = 0
    for x, y in zip(v, w):
        res += x * y
    return res
```

Altres operacions predefinides de la classe `list`:

- `append`, `count`, `pop`, etc.

Tuples

Les tuples (`tuple`) són:

- com les llistes
- *immutables* (de només de lectura)

```
z = ("hola",5,"llenguatge",6.63,2)
z = (5,)      # tupla d'un sol element
              # (5) és l'enter 5
```

Conjunts

Els conjunts (`set`) admeten les operacions:

- `len`, `in`, `not in`, issubset (`<=`), issuperset (`>=`),
- union (`|`), intersection (`&`), difference (`-`),
- `add`, `remove` ...

Diccionaris

Els diccionaris (`dict`):

- contenen parells clau-valor
- permeten accés directe

```
dic = {}                    # diccionari buit
dic["prim"] = "el primer"   # afegir o actualitzar un element
del(dic['prim`])            # esborrar-lo
dic = {"nom": "albert","num":37899, "dept": "computer science"}
                            # inicialitzar-lo amb dades
```

Els diccionaris són iterables (en iterar amb el for recorrem les claus):

```
def suma(d):
    s = 0
    for k in d:
        s += d[k]
    return s

suma({'a': 1, 'b': 2}) 👉  3
```

Contingut

Funcions anònimes

Les funcions són un tipus intern en python (*function*). Poden ser tractades com a dades i, per tant, com a paràmetres d'una funció.

Disposem de funcions anònimes tipus *lambda*:

```
lambda parametres: expressió
```

on els `parametres` són zero o més paràmetres separats per comes.

```
doble = lambda x: 2 * x      # una altra forma de definir funcions

doble(3)  👉  6

type(doble)  👉  <class 'function'>
```

Una aplicació pràctica és el paràmetre `key` de les funcions `max`, `min` i `sort`:

```
d = {'a': 2, 'b': 1}

max(d, key = lambda x: d[x])  👉  'a'      # clau amb valor màxim d'un diccionari
```

Funcions d'ordre superior I

map

`map(funció, iterable)` 👉 `generador`: aplica la funció a cadascun dels elements de l'iterable.

```
list(map(lambda x: x * 2, [1, 2, 3]))  👉  [2, 4, 6]
```

filter

`filter(funció, iterable)` 👉 `generador`: és el subiterable amb els elements que fan certa la funció booleana.

```
mg3 = lambda x: x > 3

list(filter(mg3, [3, 6, 8, 1]))  👉  [6, 8]
```

Funcions d'ordre superior II

reduce (fold)

`reduce(funció, iterable[, valor_inicial])` 👉 `valor`: desplega una funció per l'esquerra.

```
from functools import reduce

reduce(lambda acc,y: acc+y, [3,6,8,1])  👉  18

reduce(lambda acc,y: acc+y, [3,6,8,1], 0)  👉  18
```

Llistes per comprensió I

amb llistes

`[expressió for variable in iterable if expressió]`

```
[x ** 2 for x in range(4)]  👉  [0, 1, 4, 9]

[x for x in [0, 1, 4, 9] if x % 2 == 0]  👉  [0, 4]

[(x, y) for x in [1, 2] for y in 'ab']
👉  [(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')]
```

amb conjunts

```
{x for x in range(4) if x % 2 == 0}
👉  {0, 2}
```

Llistes per comprensió II

amb diccionaris

```
{x: x % 2 == 0 for x in range(4)}
👉  {0: True, 1: False, 2: True, 3: False}
```

amb generadors

```
from itertools import count      # count és un generador infinit

g = (x**2 for x in count(1))     # g és un generador
                                 # dels quadrats dels naturals

next(g)  👉  1

[next(g) for _ in range(4)]  👉  [4, 9, 16, 25]
```

Mòdul *operator*

La llibreria *operator* conté tots els operadors estàndard en forma de funcions, per a ser usades en funcions d'ordre superior.

```
from operator import mul
from functools import reduce

factorial = lambda n: reduce(mul, range(1, n+1))

factorial(5)  👉  120
```

Alguns exemples de funcions que conté són:

- `iadd(a, b)`: equivalent a `a += b`
- `attrgetter(attr)`:

  ```
  f = attrgetter('name')
  f(b)  👉  b.name
  ```

Mòdul *itertools*

Aquesta llibreria conté moltes funcions relacionades amb les iteracions.

Té moltes funcions equivalents a Haskell:

```
from operator import mul
from itertools import accumulate

factorials = lambda n: accumulate(range(1, n + 1), mul)

[x for x in factorials(5)]  👉  [1, 2, 6, 24, 120]
```

Altres funcions amb equivalents Haskell són: `dropwhile`, `islice` (`take`),`repeat` o `takewhile`.

Té algunes funcions que fan d'iteradors combinatoris: `product`, `permutations`, `combinations` o `combinations_with_replacement`.

És interessant fer un repàs a la documentació d'aquesta llibreria:
https://docs.python.org/3.7/library/itertools.html

Contingut

Exercicis

- Feu aquests problemes de Jutge.org:
  - P84591 Funcions amb nombres
  - P51956 Funcions amb llistes
  - P80049 Ús d'iterables
  - P66679 Llistes per comprensió
  - P73993 Funcions d'ordre superior