

# curso python (código facilito)

python

[https://codigofacilito.com/videos/tutorial\\_python\\_28\\_-\\_clases\\_decoradoras](https://codigofacilito.com/videos/tutorial_python_28_-_clases_decoradoras)

## que es python

- reminiscencias perl, pero más natural (pseudocódigo)
- interpretado
- tipado dinámico
- fuertemente tipado (no conbinadar diferentes tipos de variables)
- multiplataforma
- OOP

## hello world

- ```
print "Hello World"
```

## Enteros, reales y operadores aritméticos

- enteros: int, long
- reales: float
- operadores aritméticos:

```
e = 5
e = 5L
r = 0.567
r = 0.56e-3

print r

a = 25
b = 11.3
c = 5
d = 3.5

#suma
print a + b

#resta
print c - a

#multiplicación
print d * c

#exponente
print c ** 2
```

```
#división
print a / c

#división float
print float(a) / c

#modulo
print 7%3
```

## booleanos, operadores lógicos y cadenas

```
#comillas simples
cads = 'Texto entre \ncomillas simples'

#comillas dobles
cadd = "Texto entre comillas dobles"

#comillas triples
cadt = """ Texto l1
l2
l3
"""

print type(cads)
print type(cadd)

#repetición y concatenación
cad3 = "Cadena " * 3
cad2 = "Cadena"

print cad3 + cad2
```

```
bT = True
bF = False

bAnd = True and False
bOr = True or False
bNot = not True
```

## listas, tuplas, diccionarios

### listas

- ordenada
- ≡ array, vectores

```
l = [ 2, "tres", True, ["uno", 10]]
print l

print l[1]
print l[3][0]
```

```
l[1] = False
print l

l2 = l[0:3] # desde elemento 0, 3 elementos
print l2
l2 = l[1::2] # desde elemento 1, todos los elementos, uno si, uno no
print l2
l[0:2] = [4,3] # sustituye
print l
l[0:2] = [5] # sustituye y elimina un elemento
print l

print l[-1] # ultimo elemento
```

## tuplas

- sin corchetes (como las listas)
- elementos separados por coma (y opcionalmente paréntesis)
- acceso a los elementos como en las listas
- las tuplas no se pueden modificar o añadir nuevos elementos

```
t = 1, True, "Hola"
print t

t2 = (2, False, "Adios")
print type(t2)

print t2[2]

t2[0] = 4 # error
```

## diccionarios

- clave, valor'
- matrices asociativas
- no permite tener como valores listas o diccionarios
- se pueden cambiar los valores pero no las claves
- no permite slice (nomenclatura selección elementos [::])

```
d = {'c1': [1, 2, 3], 'c2': "dos", 3: True}
print d

print d['c2']
print d[3]

d[3] = False
print d

d['c3'] = "tres"
print d

del d['c3']
```

```
print d
```

## operadores relacionales

- números, cadenas, listas, tuplas
  - en cadenas los operadores del estilo > no son exactos

```
a = 1
b = 1
c = 2

r = a == c # False
r = a == b # True
r = a != c # False
r = a < c # True
r = a > c # False
r = a >= c # False
r = a <= c # True
```

## sentencias condicionales

```
#encoding: utf-8
edad = 8
m_edad = 18

if edad >= m_edad:
    print "si"
    if True:
        print "mayor de edad"
    else:
        print "no se ejecuta nunca"
else:
    print "no"

print "fuera del if"

if edad >= 0 and edad < 18:
    print "menor de edad ñ"
elif edad >= 18 and edad < 28:
    print "jovenzuelo"
elif edad >= 27 and edad < 65:
    print "adulto"
else:
    print "tercera edad"
```

## bucles

- while

```
contador = 0
while contador < 10:
```

```
contador = contador + 1
if contador == 4:
    continue # se salta el print en el 4
elif contador == 8:
    break # rompe el while en el 8
print "hola " + str(contador)
```

- for

```
lista = ["elem1","elem2","elem3"]

for mi_item in lista:
    print mi_item

for letra in "Cadena":
    print letra
```

## funciones

```
def miFuncion(num1,num2=0):
    print num1+num2

miFuncion(1,2) # 3
miFuncion(1) # 1

def miFuncion2(cad,*algomas):
    # *algomas es una tupla
    print cad + algomas[0] + algomas[1]
    for palabra in algomas:
        print palabra

miFuncion2('cadena ', 'otra ', 'mas ')

def miFuncion3(cad,**algomas):
    # **algomas es un diccionario
    print cad,algomas['cadenaextra']

miFuncion3('cadena ', cadenaextra='cadena2', otracadena='que no voy a usar')

def miFuncion4(num1,num2):
    return num1+num2

resultado = miFuncion4(3,4)
print resultado
```

## cadena y métodos

- len()
- .count(valor,inicio,final)
- .lower()
- .upper()

- `.replace(valor, reemplazo, repeticiones)`
- `.split(separador, maxsplit)`
- `.find(valor, inicio, fin)`
- `.rfind(valor, inicio, fin)`
- `.join(secuencia)`

```

• #encoding: utf-8

s = "Hola Mundo"

print len(s)

print s.count('o')
print s.count('o', 0, 4)
print s.count('o', 5)

print s.upper()
print s.lower()

print s.replace('o', '0')
print s.replace('o', '0', 1)

print s.split(' ')
print s.split()
print s.split('o')
print s.split('o', 1)

print s.find('o') # retorna índice posición
print s.rfind('o')

t = ("H", "o", "l", "a")
t2 = ";"
t3 = ""
print t2.join(t)
print t3.join(t)

```

## listas y métodos

- `<search>` in list
- `.index(element)`: da el índice de `<element>`
- `.append(element)`: añade `<element>`
- `.count(element)`: cuenta el número de veces que aparece `<element>`
- `.insert(index, element)`: inserta en la posición `<index>` un `<element>`
- `.extend(list o tupla)`: añade a la lista existente los elementos pasados.
- `.pop(index)`: extrae el elemento con `<index>` de la lista (por defecto, el último)
- `.remove(element)`: eliminar la primera coincidencia de `<element>`. Más valores con bucle
- `.reverse()`: le da la vuelta a la lista.

```

• lista = [1, "dos", 3]

buscar = 1
print buscar in lista # True

print lista.index(buscar) # 0

```

```
if buscar in lista:
    print lista.index(buscar)
else:
    print "No existe el elemento"

lista.append('Cuatro')

print lista.count("dos")

lista.insert(3,"dos")
print lista
print lista.count("dos")

tupla = (1,2,3,4)
lista.extend(tupla)
print lista

lista.pop(1)
print lista

lista.remove("dos")
print lista

lista.reverse()
print lista
```

## diccionarios y métodos

- `.has_key(<key>)`
- `.items()`: retorna lista con tupla con el índice y el valor
- `.keys()`: retorna lista con las claves
- `.values()`: retorna lista con los valores
- `.pop(valor_clave[,default])`: retorna el valor de la clave y la extrae del diccionario. default se devuelve en caso de no encontrar el valor.
- `.clear()`: deja el diccionario vacío
- `.copy()`: crea un duplicado del diccionario

## funciones de orden superior

- enviar funciones como variables y ejecutarla al usar `()`

```
def prueba(f):
    return f()

def porEnviar():
    return (2+2)

print prueba(porEnviar) # sin parentesis en porEnviar

def seleccion(operacion):
    def suma(n,m):
        return n+m
```

```
def multi(n,m):  
    return n*m  
  
if operacion == "suma":  
    return suma  
elif operacion == "multi":  
    return multi  
  
fGuardada = seleccion("suma")  
print fGuardada(3,2)  
fGuardada = seleccion("multi")  
print fGuardada(3,2)
```

## función MAP

- iteraciones de orden superior

```
def operador(n,m):  
    if n==None or m==None:  
        return None  
  
    return n+m  
  
l1 = [1,2,3,4]  
t1 = (9,8,7,6)  
t2 = (5,4,3)  
s1 = "Hola"  
s2 = "Mundo"  
  
lr = map(operador,l1,t1)  
  
print lr  
  
print map(operador,l1,t2)  
  
print map(operador,s1,s2)
```

## función FILTER

- evalúa a través de una función todos los elementos de una lista y retorna aquellos que son True

```
def filtro(elem):  
    return (elem > 0)  
  
l = [1,-2,3,-4,5,-6]  
  
print filter(filtro,l)
```

## función REDUCE

- reducir una secuencia a un elemento
- coge pares de elementos, el resultado de la anterior iteración y el siguiente elemento.
  - Para ('H','o','l','a'), empezaria por H y o, continuaria con HO y l y acabaria con Hol y a

```
• s = ('H','o','l','a',' ','M','u','n','d','o')
```

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

print reduce(concatenar,s)
```

## funciones Lambda

- función anónima
- siempre retorna algo
- sólo 1 línea
- uso en map, filter, reduce
- reduce el número de ciclos de computación usados.

```
• li = [1,-2,1,-4]
  lo = [5,3,6,7]
  s = "Hola Mundo"
  lf = lambda n,m: n+m

print map(lambda n,m: n+m, li,lo)
print filter(lambda n: n=='o', s)
print reduce (lambda n,m: n+m, lo)

print reduce (lf,lo)
print lf(3,4)
```

## comprensión de listas

- reemplaza en python 3 a map, filter
- la operación a realizar se pone primero, las condiciones o recorridos detrás ¿?¿?¿?

```
• l = [1,2,-3,4]
  l2 = ["H","O","L","A"]

print [num for num in l if num>0]
print [c * num for c in s
      for num in l2]
print [c * num for c in s
      for num in l2
      if num > 0]
```

## Generadores

- equivalente a comprensión de listas
- recorre la lista, no devuelve una lista, si no los elementos

```
l = [1,2,-3,4]
l2 = ["H","O","L","A"]

print [num for num in l if num>0]
print [c * num for c in s
        for num in l2]
r1 = (c * num for c in s
        for num in l2
        if num > 0)
print r1.next()
print r1.next()

for letra in r1:
    print letra

def factorial(n):
    i = 1
    while n > 1:
        i = n*i
        yield i # ? objeto generador
        n -= 1

for e in factorial(5):
    print e
```

## Decoradores

- función que recibe una función y devuelve una función decorada
- es decir, podemos añadir cosas antes (y supongo que después) de la ejecución de la función pasada por parámetro

```
def decorador(funcion):
    def funcionDecorada(*args, **kwargs):
        print "función ejecutada ", funcion.__name__
        funcion(*args,**kwargs)

    return funcionDecorada

def resta(n,m):
    print n-m

#decorando
print decorador(resta)(5,3) # equivalente a resta(5,3)
decorada = decorador(resta)
decorada(6,3)

@decorador
def multi(n,m):
```

```
print n*m

multi(2,4)
```

- se podrían anidar varios decoradores, se ejecutan por orden

```
loggeado = False
usuario = "codigoFacilito"

def admin(f):
    def comprobar(*args,**kwargs):
        if loggeado:
            f(*args,**kwargs)
        else:
            print "no tiene permisos para ejecutar",f.__name__
    return comprobar

@admin
def resta(n,m):
    print n-m
```

From:  
<https://matebcn.eu/> - miguel angel torres egea

Permanent link:  
<https://matebcn.eu/info:cursos:python:codigofacilito>

Last update: **10/06/2022 11:28**

