Lenguaje de programación Python

python™

Python 3 - Guía de referencia

Documentación y ayuda

```
help(objeto)     objeto?     objeto??
```

Comentarios

```
#Esta línea es ignorada
```

```
"""
Estas también y son usadas para
documentar.
"""
```

Variables

```
> a = 100
> a = "100"
> a = True
```

```
> a = b
> a = 2, b = 3
```

Tipos de dato numéricos

```
> -100, 23
> 3.75, 0.1e20
> 7 + 0.5j, -j
```

int: Número entero
float: Número decimal
complex: Número complejo

Tipo de dato lógico (boolean)

```
>True
>False
```

Verdadero
Falso

Tipo de dato de texto (string)

```
>"123a"
>'a.py'
```

string
string

Listas

```
#Colección ordenada de datos.
```

```
> a = [], b = list()
> a = ['a', True, 100, b, [1,2,'a']]
```

Índices de listas

```
> l = ['a','b','c','d','e']
l[0]      #'a'
l[-1]     #'e'
l[2:]     #['c','d','e']
l[:2]     #['a','b']
l[0:5:2]  #['a', 'c', 'e']
l[::-1]   #['e', 'd', 'c', 'b', 'a']
```

Tuplas

```
#Son como las listas, pero inmutables
t = (0, 1.2, 'abc')
t = tuple('abc') #('a','b','c')
```

Conjuntos

```
#Colección no ordenada de elementos únicos

s = {2,1,3,1}     # {1, 2, 3}
s = set('aabcb') # {'a','b','c'}
```

Importar librerías

```
from numpy import *
import numpy as np
```

Control de flujo

```
if condición_1:
   expresión_1
elif condición_2:
   expresión_2
else:
   expresión_3
```

```
while condición:
  if condición_1:
      break    #Termina el ciclo
  else:
      expresión
      continue #Empieza una iteración
```

```
for elemento in colección:
   expresión
```

```
range(x) #Lista de enteros desde 0 hasta x-1
range(x, y)  #Lista de enteros desde x hasta y-1
range(x, y, step)  #Lista de enteros desde x hasta y-1,
```

Operadores lógicos

```
> a = True
> b = False
> a and b #False    Conjunción
> a or b  #True     Disyunción
> not a   #False    Negación
```

Operadores numéricos

```
> a = 3
> a + 2 # 5        Suma
> a - 2 # 1        Resta
> -a    #-3        Negación
> a * 2 # 6        Multiplicación
> a / 2 # 1.5      División
> a % 2 # 1        Residuo(mod)
> a ** 2 # 9       Exponenciación
> a // 2 # 1       División entera
```

Operadores relacionales

```
> a = 3, b = 2
> a == b #False    Igual
> a != b #True     Distinto
> a < b  #False    Menor que
> a > b  #True     Mayor que
> a <= b #False    Menor o igual
> a >= b #True     Mayor o igual
```

Diccionarios

```
#Colecciones que relacionan llaves con valores
```

```
> d = {
       "num": 500,
       "str": "Calabaza",
       "lst": [1,2,3]
     }
> d["num"] = 501
> d.get("num")   #501
> d = dict(num = 1, str = "abc")
```

Operaciones en listas

`l[i] = x`	Asigna el valor de x en la posición i de la lista
`l.copy()`	Copia la lista
`l.clear()`	Vacía la lista
`l.sort()`	Ordena los elementos
`l.append(x)`	Agrega el elemento x al final de la lista
`l.pop(i)`	Elimina el elemento en la posición i
`l.remove(x)`	Elimina el elemento con valor x
`l.insert(i, x)`	Agrega el elemento x en la posición i
`l.index(x)`	Posición del elemento x

Funciones

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

> foo(3,5) # 8

#Argumentos por defecto
def foo(a = 5, b = 0):
    """
    Esta es la documentación
    de la función.
    """
   return a + b
> foo?       # Esta es la d...
> foo(3)     # 3
> foo(b = 5) # 10
```

Entrada y salida

```
#Entrada            #Salida
x = input()         print(x)
```

Expresiones lambda

```
lambda argumentos: expresión

foo = lambda a,b,c : a + b + c
foo        # <function>
foo(1,2,3) # 6
```

Operadores a nivel de bits

```
#Asumiendo enterosde 4 bits

> a = 6   #0110
> b = 12  #1100

#Se opera en base 2 o binario

> a & b   #0100 (4)    and
> a | b   #1110 (14)   or
> a ^ b   #1010 (10)   xor
> ~ a     #1001 (9)    not
> a << 1  #1100 (12)   left-shift
> a >> 1  #0011 (3)    right-shift
```

Operaciones en cadenas de texto

```
> a = "A", b = ' bcd '
> a + b                    'A bcd '
> a * 3                    'AAA'
> a.lower()                'a'
> b.upper()                ' BCD '
> b.replace('b','A')       ' Acd '
> b.strip()                'bcd'
> a.count('b')             0
> b.find('c')              2
```

islower()	isupper()	isdigit()	isalpha()
isspace()	istitle()	isalnum()	isupper()

Operaciones en conjuntos

```
s.add(x)              Añade el elemento x
s.remove(x)           Elimina el elemento x
s.pop()               Elimina un elemento
                      aleatorio
a.difference(b)       Diferencia
a.intersection(b)     Intersección
a.union(b)            Unión
a.issubset(b)         Subconjunto
a.issuperset(b)       Superconjunto
a.isdisjoint(b)       Conjuntos disyuntos
```

Operaciones en diccionarios

```
s.get(key)        Retorna el elemento
                  asociado a key
s.pop(x)          Elimina el elemento
                  asociado a key
s.popitem()       Elimina el último elemento
s.keys()          Retorna la lista de llaves
s.values()        Retorna la lista de valores
s.items()         Retorna la lista de parejas
                  de llaves y valores.
```

Operadores in y is

```
¿x está en y?     ¿Son el mismo objeto?
x in y            x is y
```

Comprensión de listas

```
l = []
for elemento in colección:
   if condición:
     l.append(elemento)

#Con comprensión de listas
l = [elemento for elemento
in colección if condición]
```

Función *Map*

```
l = map(función, coleccion)

> m = map(lambda x: x.upper(), 'abcd')
 <map object at 0x7f43fed922b0>
> list(m)
 ['A', 'B', 'C', 'D']
```

Función Filter

```
l = filter(función, colección)
> f = filter(lambda x: x.isupper(), 'AbCdEf')
   <filter object at 0x7f43fed92c18>

> "".join(f)
   'ACE'
```