
python™

Contenido

- Introducción a Python
- Historia
- Interpretado Vs Compilado
- Instalar Python
- Tipos de datos
- Estructuras de datos
- Funciones
- Control de Flujo

Introducción a Python

Python es un lenguaje de programación poderoso y fácil de aprender.

Se trata de un lenguaje multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Introducción a Python

La filosofía hace hincapié en una sintaxis que favorezca un código legible.

Tiene influencias de ABC, ALGOL 68, C, Haskell, Icon, Lisp, Modula-3, Perl, Smalltalk, Java

Pagina Oficial: http://www.python.org

Comunidad Argentina: http://www.python.org.ar

Historia

Python fue creado a finales de los 80 por Guido van Rossum en el Centro para las Matemáticas y la Informática, en los Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.

En 1991, van Rossum publicó el código de la versión 0.9.0 en alt.sources. En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones, funciones y los tipos modulares, como: str, list, dict, entre otros.

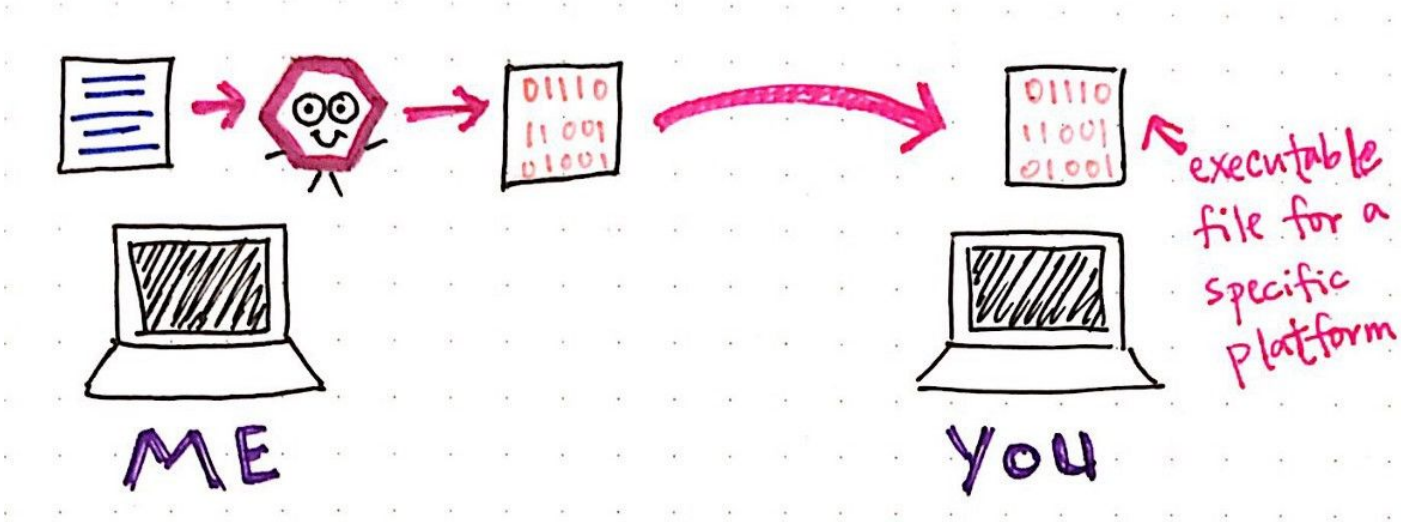
Historia - Guido Van Rossum

Van Rossum es el principal autor de Python, y su continuo rol central en decidir la dirección de Python es reconocido y por el le otorgaron el título de "Benevolente Dictador Vitalicio".

El nombre del lenguaje proviene de la afición de su creador por los humoristas británicos Monty Python.

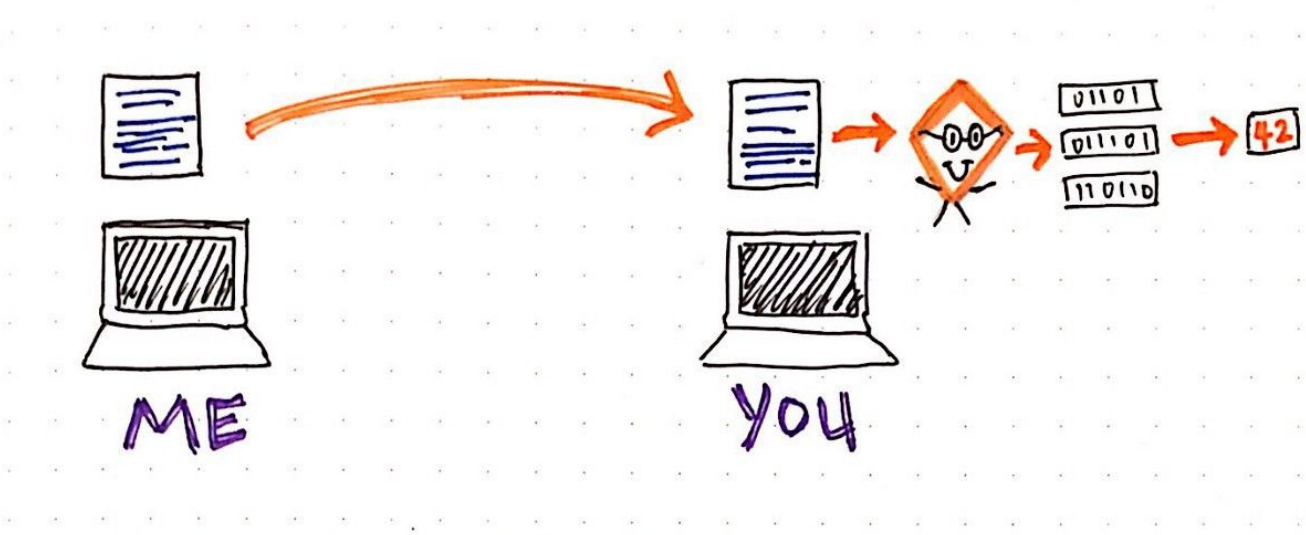
Interpretado Vs Compilado

Compilado



Interpretado Vs Compilado

Interpretado



Instalar Python

Linux

Ubuntu

```
$ sudo apt install python3.X
$ sudo apt install python-pip
$ sudo apt install python-virutalenv
```

Arch / Manjaro

```
$ sudo pacman -S python3.X
$ sudo pacman -S python-pip
$ sudo pacman -S python-virutalenv
```

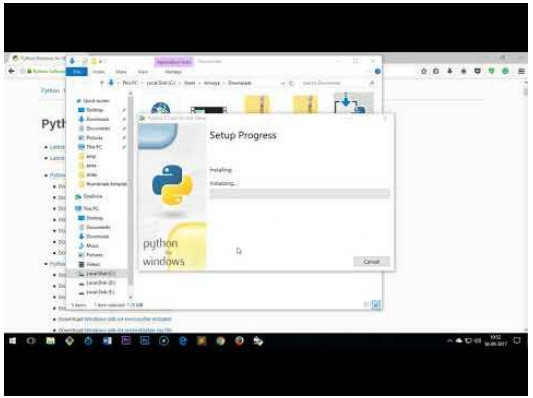
RedHat / OpenSuse / Fedora / Centos

```
$ sudo yum -y install rh-python36
$ sudo yum -y install rh-python-pip
$ sudo yum -y install rh-python-virtualenv
```

Instalar Python

Windows

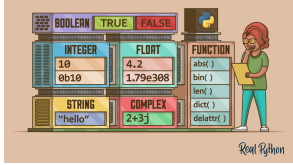
https://www.youtube.com/watch?v=Wx8XU2L2k6Q



Instalar Python

Windows

https://www.youtube.com/watch?v=uA8SA81nivg



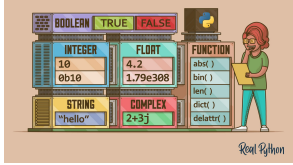
Tipos de datos - [BOOLEAN - AND, OR, NOT]

El tipo de dato lógico o booleano es en computación aquel que puede representar valores de lógica binaria, esto es 2 valores, que normalmente representan falso o verdadero.

Valores:

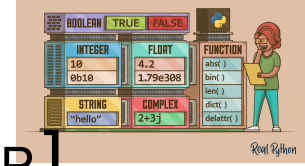
- True
- False

Operaciones	**Resultado**
x or y	si x es verdadera o y es verdadera es el resultado es verdadero
x and y	si x e y son verdaderos es verdadero
not x	es el opuesto del valor de x



Tipos de datos - [BOOLEAN - Comparación]

Operación	Descripción
a < b	cuando a es menor a b
a <= b	cuando a es menor o igual a b
a > b	cuando a es mayor a b
a >= b	cuando a es mayor o igual a b
a == b	cuando a es igual a b
a != b	cuando a no es igual a b
a is b	cuando a es identico a b
a is not b	cuando a no es igual a b

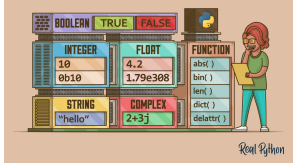


Tipos de datos - [TIPOS DE NUMEROS - INT, FLOAT, COMP.]

Valores:

- int: 125
- float: 123123.2123
- complex: 1+2j

Operacion	Resultado
x + y	suma
x - y	resta
x * y	multiplicacion
x / y	divicion
x // y	cociente entero
x % y	resto
-x	negativo
+x	no cambia
abs(x)	valor absoluto
int(x)	convierte en entero
loat(x)	convierte en flotante
complex(re, im)	convierte en numero imaginario
c.conjugate()	conjuga el valor c
divmod(x, y)	resultado divicion y modulo (x // y, x % y)
pow(x, y)	potencia de y en x
x ** y	potencia de y en x



Tipos de datos - [SECUENCIA DE TEXTO — STRX]

Valores:

- cita simple: 'allows embedded "double" quotes'
- cita doble: "allows embedded 'single' quotes".
- cita triple: '''Three single quotes''', """Three double quotes"""

Estructuras de datos - Listas

Las listas son estructuras de datos mutables que nos permite agregar distintos tipos de datos, por ejemplo enteros, string, otras listas.

Ejemplo

```
>>lista = [1, 2, 4, 6, 7, 8, 20, 30, 13, 'nombre', [2, 6]]
```

Estructuras de datos - Listas

Recuperar datos de las listas

Por Iteración

```
>>lista = [1, 2, 3]
>>for item in lista:
>>    print(lista)
```

Por Índice

```
>>lista = [1, 2, 3]
>>print(lista[2])
 3
```

Estructuras de datos - Listas

Agregar elemento a la lista

```
# Lista original
>>lista = [2, 5, 6]

# Agregar
>>lista.append(10)

# Resultado
>>print(lista)
 [2, 5, 6, 10]
```

Estructuras de datos - Listas

Agregar subconjunto a una lista existente

```
>>lista = [1, 2, 3]
>>lista.extend([7, 8])
>>print(lista)
 [1, 2, 3, 7, 8]
```

Remover elementos de una lista

```
>>lista = [1, 2, 3, 4, 5]
>>lista.remove(3)
>>print(lista)
 [1, 2, 4, 5]
```

Estructuras de datos - Listas

Obtener indice de un elemento

```
>>lista = [1, 2, 3, 4]
>>print(lista.Index(2))
1
```

Extracción de elementos

```
>>lista = [1, 2, 3, 4, 5]
>>lista.pop()
5
>>lista.pop()
4
>>print(lista)
[1, 2, 3]
```

Estructuras de datos - Listas

Ordenar elementos: Ascendente

```
>>lista = [3, 6, 7, 3, 1, 4, 5]
>>lista.sort()
>>print(lista)
 [1, 3, 3, 4, 5, 6, 7]
```

Ordenar elementos: Descendente

```
>>lista = [3, 6, 7, 3, 1, 4, 5]
>>lista.Reverse()
>>print(lista)
 [7, 6, 5, 4, 3, 3, 1]
```

Estructuras de datos - Tuplas

Las tuplas son como las listas estructuras de datos ordenadas, que puden contener distintos tipos de elementos.

Pero las tuplas son inmutables

Ejemplo:

```
>>tupla = (3, 6, 7, 3, 1, 4, 5)
```

Estructuras de datos - Diccionarios

Los diccionarios son estructuras de datos que permiten contener distintos tipos de datos, todos identificados por una llave -> key

Ejemplo:

```
diccionario = {
    'nombre': 'Pablo',
    'apellido': 'Dalmasso',
    'edad': 35
 }
```

Estructuras de datos - Diccionarios

Obtener elementos

```
>>diccionario = {
>>     'nombre': 'Pablo',
>>     'apellido': 'Dalmasso',
>>     'edad': 33
>>}
>>print(diccionario['nombre'])
 'Pablo'
```

dict()

```
>> dic = dict(nombre='nestor', apellido='Plasencia', edad=22)
{'nombre': 'nestor', 'apellido': 'Plasencia', 'edad': 22}
```

Estructuras de datos - Diccionarios

zip()

```
>>dic = dict(zip('abcd', [1, 2, 3, 4]))
 {'a': 1, 'b': 2, 'c': 3 , 'd': 4}
```

items()

```
>>dic = {'a': 1, 'b': 2, 'c': 3 , 'd': 4}
>>items = dic.items()
 [('a', 1), ('b', 2), ('c', 3), ('d', 4)]
```

keys()

```
>>dic = {'a': 1, 'b': 2, 'c': 3 , 'd': 4}
>>keys = dic.keys()
 ['a', 'b', 'c', 'd']
```

values()

```
>>dic = {'a': 1, 'b': 2, 'c': 3 , 'd': 4}
>>values= dic.values()
 [1,2,3,4]
```

Estructuras de datos - Diccionarios

fromkeys()

```
>>dic = dict.fromkeys(['a','b','c','d'],1)
 {‘a’ : 1, ’b’ : 1, ‘c’ : 1 , ‘d’ : 1}
```

get()

```
>>dic = {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
>>dic.get(‘b’)
2
>>dic.get(‘z’, 'n/a')
 'n/a'
```

pop()

```
>>dic = {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
>>dic.pop(‘b’)
2
>>print(dic)
 {‘a’ : 1, ‘c’ : 3 , ‘d’ : 4}
```

Estructuras de datos - Diccionarios

update()

```
>>dic_1 = {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}
>>dic_2 = {'c' : 6, 'b' : 5, 'e' : 9 , 'f' : 10}
>>dic_1.update(dic_2)
 {'a' : 1, 'b' : 5, 'c' : 6 , 'd' : 4 , 'e' : 9 ,
 'f' : 10}
```

Funciones

En informática, una función, se presenta como un subalgoritmo que forma parte del algoritmo principal, el cual permite resolver una tarea específica

Funciones

Definición de Funciones en python Ejemplo

```python
def make_fibonacci(limit, verbose=False) -> list:
    """
    Genera la serie de Fibonacci hasta n.
    Parametro:
        limit: Hasta el numero que se debe generar.
        verbose: Si debe imprimir la serie
    """
    a = 0
    b = 1
    result = []
    while a < limit:
        if verbose:
            print(a, end=', ')
        result.append(a)
        a, b = b, a+b
    if verbose:
        print()
     return result
```

Uso de la función

```python
make_fibonacci(200)
```

Funciones

ARGUMENTOS CON VALORES DEFAULT, PALABRAS CLAVES

Definición:

```
def fu(arg01='Primero', arg02='Segundo', arg03='Tercero'):
    print('{0}, {1}, {2}'.format(arg01, arg02, arg03))
```

Uso:

```
fu(arg02='Otro Valor')
```

Funciones

LISTA DE ARGUMENTOS

Definición:

```
def muchos_items(separador, *args):
    print(separador.join(args))
```

Uso:

```
muchos_items('.', 'Primero', 'Segundo', 'Tercero')
```

Funciones

LISTA DE ARGUMENTOS POR PALABRAS CLAVES

Definición:

```
def muchos items palabras(**kwargs):
    for key in kwargs.keys():
        pritn('{0}: {1}'.format(key,
 kwargs[key]))
```

Uso:

```
muchos_items_palabras(nombre='pablo', apellido='dalmasso')
```

Funciones

DEVOLVIENDO RESULTADOS

Definición:

```
def es_mayor_a_100(numero) -> bool:
    return numero > 100
```

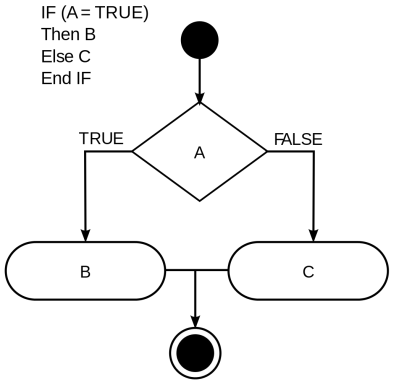
Uso:

```
>> es_mayor_a_100(10)
False
```

Control de Flujo

IF

```
if <condicional>:
    <bloque de
ejecucion>
elif <condicion2>:
    <bloque de
ejecucion>
else:
     <bloque de
 ejecucion >
```



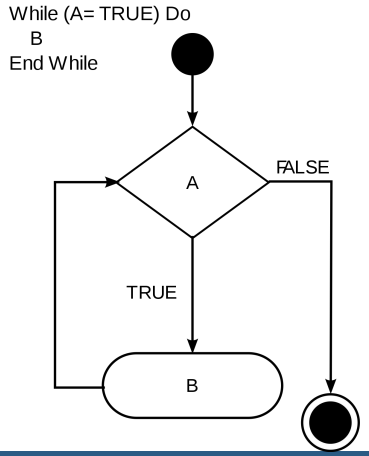
Control de Flujo

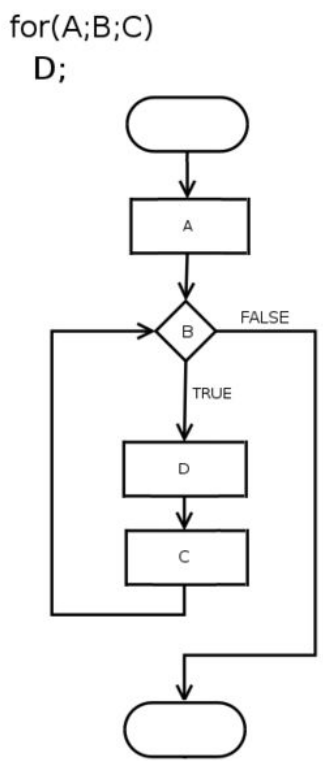
For

```
for <item> in <arreglo>:
    <bloque de codigo>
```

While

```
while <condicional>:
    <bloque de codigo>
```





<COMPLETAR>

<COMPLETAR>