

Variables

Entero
a = 1
Punto flotante
b = 1.0
Número Complejo
c = 1 + 2a
String
d = 'a'
Boolean (Verdadero/Falso)
e = False
Definiendo variables, Python 3.5+ admite 'anotaciones de tipo':
{varname}: {type}
{varname}: {type} = {value}
fname: str = "lorenzo"

Data Structures

Listas

Creación de listas
a = [1, 2, 3, 4, 5]
Procesando en una sola línea
vals = [value for value in collection if condition]
Esto es equivalente a:
vals = []
for value in collection:
 if condition:
 vals.append(expression)

Tuplas

Las tuplas son inmutables y generalmente contienen una secuencia heterogénea.

Para crear tuplas coloca los valores entre paréntesis
a = (1, 2, 3, 4, 5)

Sets

Los Sets son secuencias mutables y desordenadas de elementos únicos.
Para crear Sets coloca valores entre llaves.
a = {1, 2, 3, 4, 5}

Diccionarios

Conjunto desordenado de clave: pares de valores.
Para crear diccionarios, coloque la clave: pares de valores entre llaves.
a = {'first_name': 'Lorenzo', 'age': 30}

Strings

Colección de caracteres
Comillas simples
'Hello'
Comillas dobles
"World"
Multi-línea
"""
Hello
world
"""
Concatenar
"Hello" + "world"
Multiplicar
>>> "hello" * 4
'hellohellohellohello'
Tamaño
>>> len("Hello world")
11

Formato: Accede a los argumentos por posición ordinal.
>>> '{0}, {2}, {1}'.format(1, 2, 3)
'1, 3, 2'
Formato: Argumentos posicionales implícitos
>>> '{}', {}, {}'.format(1, 2, 3)
'1, 2, 3'
Formato: Acceso a argumentos de palabras clave por nombre
>>> '{val1}, {val2}, {val3}'.format(val1=1, val2=2, val3=3)
'1, 2, 3'

Bucles

For

for i in range(0, 10):
 print(i)
Iterar sobre una matriz
my_array = [1, 1, 2, 3, 5, 8, 13]
for d in my_array:
 print(d)
Iterar sobre una matriz y obtener el índice
my_array = [1, 1, 2, 3, 5, 8, 13]
for index, d in enumerate(my_array)
 print(index, d)

While

counter = 0
while counter < 10:
 print(counter):
 counter += 1

Funciones

Declarando Funciones

Básicas:

```
def name(param, key_parm=default_value):  
    ...  
    return result
```

*args y **kwargs te permiten pasar un número variable de argumentos a una función

- *args parámetros sin nombre
- **kwargs parámetros con nombres

```
def name(param, key_parm=default_value,  
*args, **kwargs):  
    ...  
    return result
```

Python 3.5+ admite anotaciones de tipo

```
def add(a: int, b: int) -> int:  
    return a + b
```

Excepciones

```
try:  
    # bloque de procesamiento normal  
    ...  
except Exception as e:  
    # error processing block  
    ...  
finally  
    # block for final processing in all cases  
    ...
```

Clases

Declarando una clase

```
class Dog:  
    race = 'Dog' # Class Attribute  
    # Initializer  
    def __init__(self, name):  
        self.name = name  
  
    # instance method  
    def description(self):  
        return "{} is a {}".format(  
            self.name, self.race)
```

```
courage = Dog("courage")
```

Herencia

```
class Bulldog(Dog):  
    race = 'bulldog'  
  
monster = Bulldog("Monster")
```

Trabajando con Archivos

Abrir archivos

```
f = open(file, mode='r', encoding=None)
```

file: ruta al archivo

mode:

- 'r' Leer
- 'w' Escribir
- 'x' Creación exclusiva, falla si el archivo ya existe.
- 'a' Adjuntar

'b' Modo binario
'+' Abrir un archivo de disco para actualizar (leer y escribir)
Encoding: es el nombre de la codificación utilizada para decodificar o codificar el archivo.

Leer

Leer el archivo entero

```
f.read()
```

Leer línea por línea

```
f.readline()
```

Leer todas las líneas y devolver una lista de líneas

```
f.readlines()
```

Escribir

Necesita estar abierto en modo escritura:

```
f = open('test.txt', 'w')
```

Escribir contenido

```
f.write("hello world")
```

Escribir líneas

```
f.writelines(["hello", "world"])
```

Siempre cerrar el archivo

```
f.close()
```

Con el enunciado *with*

Para leer:

```
with open('test.txt', 'r') as f:  
    content = f.read()
```

Para escribir:

```
with open('test.txt', 'w') as f:  
    f.write("Hello world")
```