

# Introducción a programación en Python

Clase 2  
Ing. Alexander López

1

## Introducción











Lenguaje de programación interpretado, lanzado al público el 1991 y desarrollado por Guido van Rossum

Su sintaxis obliga al programador a codificar de forma ordenada o de lo contrario el código no funciona.

Cuenta con reglas y estilos estandarizados, haciendo que su código siempre sea legible, de tal forma que otros programadores puedan entender cualquier código con gran rapidez y facilidad.

2

## ¿Cuál es el lenguaje más usado?

Language Rank	Types	Spectrum Ranking
1. C		100.0
2. Java		98.1
3. Python		98.0
4. C++		95.9
5. R		87.9
6. C#		86.7
7. PHP		82.8
8. JavaScript		82.2
9. Ruby		74.5
10. Go		71.9

<http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages>

3

## Características

Lenguaje interpretado de alto nivel

Multiplataforma

Asignación dinámica

Permite que se elija más de una técnica o paradigma para cada trabajo

Multiparadigma

4

## Elementos

### Variables

Python asigna un espacio determinado en su memoria para la variable sin que el usuario tenga que asignar una dirección de memoria, además los nombre que se eligen para las variables no deben tener signos de puntuación, para asignarse debe igualarse a un valor en particular.

**Ejemplo:**

```
>> mi_variable = 12
```

5

## Elementos

### Constante

Una constante es un valor permanente que no puede ser modificado, al menos no durante el programa, en Python los constantes se declaran en mayúsculas a diferencias de las variables, pero con las mismas reglas.

**Ejemplo:**

```
>> MI_CONSTANTE = 19
```

6

## Elementos

### Tipos de datos

Existen diversos tipos de variable que acepta Python, en seguida se muestra las más importantes.

**Ejemplo:**

String:	<pre>&gt;&gt; palabra = "Hola NAO!"</pre>
Entero hexadecimal:	<pre>&gt;&gt; numero = 0x23</pre>
Número real:	<pre>&gt;&gt; numero = 32.42</pre>
Booleano:	<pre>&gt;&gt; clave = True</pre>

7

## Elementos

### Operadores Aritméticos

A continuación se muestra con un ejemplo, los operadores matemáticos más utilizados:

Operación:	Ejemplo:	Resultado:
Suma:	<pre>&gt;&gt; x = 7 + 3</pre>	10
Resta:	<pre>&gt;&gt; x = 7 - 3</pre>	4
Negativo:	<pre>&gt;&gt; x = -6</pre>	-6
Multiplicación:	<pre>&gt;&gt; x = 9 * 5</pre>	45

8

## Elementos

## Operadores Aritméticos

A continuación se muestra con un ejemplo, los operadores matemáticos más utilizados:

Operación:	Ejemplo:	Resultado:
Exponente:	<code>&gt;&gt; x = 3 ** 2</code>	x igual 9
División:	<code>&gt;&gt; x = 7.5 / 2</code>	3.25
División entera:	<code>&gt;&gt; x = 7.5 // 2</code>	3.0
Módulo:	<code>&gt;&gt; x = 32 % 5</code>	6

9

## Elementos

## Comentarios

Los comentarios son sentencias escritas en el código que no se ejecuta, los programadores lo usan para colocar anotaciones que ayuden a entender las estructuras del código, enumerar los pasos que se sigue en el procesamiento o desactivar algunas líneas de código.

**Ejemplo:**  
`#variables`  
`a = 23 # Numero de casas`

10

## Elementos

## Descripciones

Usado para explicar que hace, que variables necesita, que retorna las funciones que han creado o simplemente para comentar varias filas de código.

**Ejemplo:**  
`"""# Lineas de comentarios o código no se ejecutan`  
`x = 25`  
`x = 25 / 5 """`

11

## Elementos

## Tipos de datos complejos

Aparte de los tipos de datos ya revisados, 3 tipos de datos más que admiten en ellos una colección de datos.

12

## Elementos

### Tipos de datos complejos

Lista de datos inmutables: Una lista de datos inmutables, puede contener varios tipos de datos.

#### Ejemplo:

```
>> lista_di = ('string',21,17,-2,2.5,'dato')
>> print lista_di[1]
>> print lista_di[0:3]
>> print lista_di[-2]
```

#### Resultado:

```
21
('string',21,17)
2.5
```

13

## Elementos

### Tipos de datos complejos

Lista de datos: A diferencia de la lista de datos inmutables, en esta lista se pueden modificar los datos y hasta se pueden agregar datos a la lista.

#### Ejemplo:

```
>> lista = ['string',21,'valor',17]
>> print lista[1:3]
>> lista[1] = 12 #cambia de 21 a 12
>> lista.append('nuevo valor')
```

#### Resultado:

```
>> [21,'valor',17]
```

14

## Elementos

### Tipos de datos complejos

Diccionario: Contiene una lista de datos, estos datos son variables y cada uno tiene un valor el cual se le asigna al declarar el diccionario:

#### Ejemplo:

```
>> diccionaiio = {'valor1': Hola, 'valor5': ros}
>> print diccionario['valor_1']
```

#### Resultado:

```
Hola
```

15

## Elementos

### Tipos de datos complejos

Eliminar un elemento del diccionario:

```
>> del(diccionario['valor_2'])
```

Cambiar el valor de alguna de las variables que pertenecen al diccionario:

```
>> diccionario['valor_1'] = 'Hello'
```

16

## Control de Flujo

If

```
>> if numero <= 50:  
...     print "Menor o igual que 50"  
>> elif numero > 50 and numero < 200:  
...     print "Numero entre 51 a 199"  
>> else:  
...     print "Cualquier numero mayor a 200"
```

17

## Control de Flujo

While

Ejecuta las instrucciones dentro de su bucle mientras se cumpla una condición:

**Ejemplo:**

```
>> repeticion = 0  
>> while repeticion <= 3:  
...     print "Numero de repeticiones = ", str(repeticion)  
...     repeticion += 1
```

18

## Control de Flujo

While

**Resultado:**

```
Numero de repeticiones = 0  
Numero de repeticiones = 1  
Numero de repeticiones = 2  
Numero de repeticiones = 3  
fg:reanudar
```

19

## Control de Flujo

For

**Ejemplo:**

```
>> lista = ['Alejandro', 'Gladys', 'Camila']  
>> for nombre in lista:  
...     print nombre
```

**Resultado:**

```
Alejandro  
Gladys  
Camila
```

20

## Control de Flujo

For

Ejemplo:

```
>> for numero in range(0, 2):  
...     print "Lista de numeros: ", str(numero)
```

Resultado:

```
Lista de numeros: 0  
Lista de numeros: 1
```

21

## Módulos y paquetes

Módulos  
empaquetados

Se pueden usar métodos y funciones de otros scripts, cuando se importa dicho paquete o módulo, no debe escribirse con su extensión .py, solo su nombre. Importar módulos enteros:

Ejemplo:

```
>> import serial
```

Se hace un llamado al módulo de comunicación serial

22

## Módulos y paquetes

Módulos  
empaquetados

Importar usando un alias, el contenido de cada módulo se puede importar por completo con otro nombre usando la función:

Ejemplo:

```
>> import serial as ser  
>> import numpy as np
```

23

## Módulos y paquetes

Módulos  
empaquetados

Importar un módulo en particular, el contenido de un módulo en particular se puede importar por usando el comando "from", en el cual se especifica el nombre del paquete de dónde viene el módulo.

Ejemplo:

```
>> from serial.tools import list_ports
```

24

## Funciones

### Definiendo funciones

Ejemplo:

```
>> def funcion():  
...     print "Hola!"
```

Resultado:

```
funcion()
```

Hola!

25

## Funciones

### Parámetros de entrada

Ejemplo:

```
>> def funcion(x,y):  
...     return x+y*x
```

Resultado:

```
>> print funcion(3,2)
```

9

```
>> print funcion(2,3)
```

8

26

## Funciones

### Parámetros por omisión

Es posible asignar valores por defecto, quiere decir que tienen valores ya definidos y si estos no se cambian al ejecutar la función, se ejecuta usando sus valores por defecto.

27

## Funciones

### Parámetros por omisión

Ejemplo:

```
>> def funcion(nom, frase = 'que tal?):  
...     x = 'Hola' + ' ' + nom + ', ' + frase  
...     return x  
>> print funcion('Alex')
```

Resultado:

Hola Alex, que tal?

28

## Funciones

### Parámetros arbitrarios

Es posible que en una función pueda recibir un número indefinido de argumentos, dependiendo de las necesidades del usuario, todas ellas deben ser recibidas y utilizadas por la función. Para definir argumentos arbitrarios en una función, se antecede al parámetro un asterisco.

29

## Funciones

### Parámetros arbitrarios

#### Ejemplo:

```
>> def funcion(dato_fijo, *datos):  
...     print dato_fijo  
...     for argumento in datos:  
...         print argumento  
>> funcion('Fijo', 'dato 1', 'dato 2')
```

#### Resultado:

```
Fijo  
dato 1  
dato 2
```

30

## Funciones

### Llamadas recursivas

Son aquellas que dentro de su algoritmo, se llama a sí misma, lo cual genera un bucle, debe verificarse de que el bucle no se repita infinitamente ya que el programa podría colapsar, todo programa debe terminar en alguno momento.

31

## Funciones

### Llamadas recursivas

#### Ejemplo:

```
>> def juego(intento=1):  
...     respuesta = raw_input("¿De que color el cielo? ")  
...     if respuesta != "azul":  
...         if intento < 3:  
...             print "\nFallaste! Inténtalo de nuevo"  
...             intento += 1  
...             juego(intento) # Llamada recursiva  
...         else:  
...             print "\nPerdiste!"  
...     else:  
...         print "\nGanaste!"  
>> juego()
```

32



## POO

### Elementos y características

**Clases:** son la base del lenguaje de programación orientado a objetos, son los modelos sobre los cuales se construirán los objetos. A la creación de objetos se le llama instanciar una clase.

**Ejemplo:**

```
>> class Objeto:
```

33

## POO

### Elementos y características

**Propiedades:** Son las características intrínsecas del objeto, estas se representan como variables; al definir el objeto en el ejemplo anterior, no se le asignó ninguna cualidad, simplemente se creó sin cualidad alguna. En este caso veremos cómo se asignan algunas características del objeto.

34

## POO

### Elementos y características

**Ejemplo:**

```
>> class Objeto:
```

```
...     color = "verde"
```

```
...     tamano = "mediano"
```

```
...     peso = "55"
```

```
>> mark = Objeto()
```

```
>> print mark.color
```

```
>> print mark.tamano
```

**Resultado:**

verde

mediano

35

## POO

### Elementos y características

**Métodos:** son más que funciones y se denominan así para indicar que son diferentes de las funciones normales, están dirigidas a trabajar con los objetos.

36

## POO

### Elementos y características

#### Ejemplo:

```
>> class Objeto():
...     color = "verde"
...     tamaño = "mediana"
...     peso = "55"
...     def edad(self,x):
...         y = 21 + x
...         print y
>> mark = Objeto()
>> mark.edad(2)
```

#### Resultado:

23

37

## POO

### Elementos y características

La herencia es otro elemento de la programación dirigido a objetos, cuando una clase hereda algo de otra significa que la segunda clase hereda cada uno de los atributos y funciones de la primera, además de la declaración de herencia se pueden instanciar otros atributos o funciones.

38

#### Ejemplo:

```
class Objeto:
    color = "verde"; tamaño = "mediano"; peso = "55"
    def edad(self,x):
        y = 21 + x
        print y
class NuevoObjeto(Objeto):
    forma = "circular"
    def talla(self,x):
        x = x + 1.55
        print x

oak = NuevoObjeto()
print oak.color
print oak.forma
oak.talla(0.05)
```

#### Resultado:

verde  
circular  
1.60

39

## POO

### Elementos y características

Acceder a métodos y propiedades de un objeto. Algunos métodos se usan para obtener valores, estos pueden guardarse en otras variables para ser usados por el usuario sin modificar dichos valores.

40

Ejemplo:

Ejemplo:

class Objeto:

color = "verde"

tamano = "mediano"

peso = "55"

edad = 21

oak = Objeto()

edad = 1 + oak.edad

print edad

frase = "Es" + oak.color + ", " + oak.tamano + " y pesa " + oak.peso

print frase

Resultado:

22

Es verde, mediano y pesa 55

41

# ¡Gracias!

La única pregunta tonta es la que no se  
hace

42