Resumen Python

1 – Introducción a Python

Python es un lenguaje de alto nivel, multiplataforma e interpretado. Además, tiene muchas características de los lenguajes compilados, por lo que se dice que es un lenguaje semi interpretado.
En Python, el código fuente se traduce a un pseudo código maquina intermedio llamado bytecode la primera vez que se ejecuta.

Para escribir un programa en Python solo hace falta abrir en un editor de texto un archivo con extensión .py. En Linux para abrir el intérprete adecuado: *#!/usr/bin/python*

Para ejecutar el código, hay que escribir en el intérprete: *python3 -i Nombre.py*
Y luego escribir la función.

Tipos básicos:

- Int (enteros).
- Long (enteros largos).
- Float (números racionales).
- Complex (números complejos).
- Cadenas de texto.
- Valores Booleanos.

Para verificar que valor tiene una variable se puede utilizar la función *type*.

Operadores aritméticos:

Cuando se mezclan tipos de números, Python convierte a todos los operandos al tipo mas complejo de entre los tipos de los operandos que forman la expresión.

Operador	Descripción	Ejemplo	
+	Suma	`r = 3 + 2  # r es 5`	
-	Resta	`r = 4 - 7  # r es -3`	
-	Negación	`r = -7  # r es -7`	
*	Multiplicación	`r = 2 * 6  # r es 12`	
**	Exponente	`r = 2 ** 6  # r es 64`	
/	División	`r = 3.5 / 2  # r es 1.75`	
//	División entera	`r = 3.5 // 2 # r es 1.0`	→Redondea hacia abajo
%	Módulo	`r = 7 % 2  # r es 1`	

Para llamar a comandos matemáticos usamos: *from math import* * y obtenemos

Function name	Description
abs(value)	Valor absoluto
ceil(value)	Redondea para arriba
cos(value)	Coseno, en radianes
floor(value)	Redondea, para abajo
log10(value)	Logaritmo, base 10
max(value1, value2)	Mayor de dos valores
min(value1, value2)	Mínimo de dos valores
round(value)	Entero más cercano
sin(value)	Seno, in radians
sqrt(value)	Raíz cuadrada

Constant	Description
e	2.7182818...
pi	3.1415926...

Operadores booleanos:

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	`r = True and False # r es False`
or	¿se cumple a o b?	`r = True or False  # r es True`
not	No a	`r = not True  # r es False`

Operadore relacionales:

Los valores booleanos son además el resultado de expresiones que utilizan operadores relacionales.

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	`r = 5 == 3 # r es False`
!=	¿son distintos a y b?	`r = 5 != 3 # r es True`
<	¿es a menor que b?	`r = 5 < 3  # r es False`
>	¿es a mayor que b?	`r = 5 > 3  # r es True`
<=	¿es a menor o igual que b?	`r = 5 <= 5 # r es True`
>=	¿es a mayor o igual que b?	`r = 5 >= 3 # r es True`

Strings:

Los String son texto encerrado en comillas simples o dobles. En ellas se puede añadir caracteres especiales precedentes de \ (\n *'nueva línea'* , \t *'tabulación')*.
Una cadena precedida por el carácter r, indica que es una cadena Raw (cruda).

Estas cadenas se distinguen de las normales en que los caracteres especiales no se sustituyen por sus contrapartidas.
También se puede formar String con triples comillas.

Los caracteres de un String son asociados a un índice ([ ]).

Ejemplo:

Name = "P. Diddy"

index	0	1	2	3	4	5	6	7
character	P	.		D	i	d	d	y

Por lo visto en el ejemplo se puede generalizar que para acceder a un carácter individual se usa *variable[índice].*

Usar "," como separador de Sting nos permite formar una cadena con diferentes datos.

Para concatenar String podemos usar "+". Solo se puede concatenar dos String.

Ejemplo:

"Hola" + "mundo" = "Hola mundo"

Se puede multiplicar un String con un entero.

Ejemplo:

*"Hola" * 2 = HolaHola*

Bloques:

Los bloques de código en Python están determinados por las identaciones.

Funciones:

Una función es un fragmento de código que tiene asociado un identificador que realiza una serie de operaciones y devuelve un valor.
Cuando un fragmento de código tiene un valor asociado y no devuelve valor se le llama procedimientos, pero en Python no existen ya que cuando no se especifica valor la función devuelve None.
También existen funciones anónimas las cuales no cuentan con identificador, lo que impide su reutilización.

Las funciones están declaradas de la siguiente manera:

```
def identificador (param1, param2):
        print param1
        print param2
```

Python permite asignar valores por defecto como parámetro, en caso que no sean pasados se toma el asignado.

Ejemplo:

```
def imprimir (texto, veces = 1)
        print (texto * veces)

imprimir("hola") == "hola"
```

También Python puede identificar el valor sin respetar el orden de la definición.

Para que una función retorne un valor se debe utilizar la palabra *return*.
print y *return* no son lo mismo. El *print* lo único que hace es mostrar en pantalla lo que se le asigna, por ej.: print("Hola") muestra en pantalla la cadena Hola.
En cambio, el *return* almacena en la memoria la variable que se le esta asignando, asi si es llamado por otra función se puede utilizar, en cambio el *print* no almacena nada.

Entradas:

Para obtener información por parte del usuario se puede usar la función *input* que toma como parámetro una cadena usada como prompt y devuelve una cadena con los caracteres o números introducidos por el usuario.

Condicionales:

En Python existe una estructura de control donde, dependiendo del valor de una condición se ejecuta una sentencia.

```
if condición1:
        bloque si se cumple condición1
elif condición2:
        bloque si se cumple condición2
else:
        bloque en caso contrario
```

La opción *elif* y la *else*, es opcional y equivalente a hacer otro *if*, en el caso que la condición anterior sea falsa. Se pueden poner un *elif* o más, pero existe un único *else*.

2, 3 – El modelo iterativo – Mas iteraciones

Listas:

Las listas son una colección ordenada de elementos. Pueden contener cualquier tipo de datos. Se indica entre corchetes y se separa por comas los valores incluidos en la lista. Se puede acceder a cada uno de los elementos de la lista indicando el nombre de la lista e indicando el índice del elemento. Las listan son indexadas. Se puede utilizar como índice un numero negativo, devolviendo el final de la lista y así sucesivamente.

.append(): agrega un elemento en la última posición de la lista.

.insert(): se agrega un elemento en la posición indicada de la lista. Si pongo una posición mayor al len (tamaño) de la lista se agrega al final.

El "." Significa la aplicación de una función sobre un dato, modificándolo.

Slicing:

Si en lugar de indexar con un solo número, escribimos dos números separados por dos puntos (inicio:fin) entonces Python devuelve una lista que va desde la posición inicio hasta fin (sin incluir).
Si escribimos tres números (inicio:fin:salto), el tercero se utiliza como un "salto", quiere decir, cada cuantas posiciones se añade un elemento a la lista.
Si no se aclara el principio o el final del slicing se usan por defecto el inicio y el final.
El slicing se puede utilizar para modificar la lista (crea una nueva lista, la sobreescribe).

```
>>> l = [99, True, "una lista", [1, 2, 3, 4]]
>>> l[0:2] = ["nuevos", "valores"] #modifico valores
>>> l
['nuevos', 'valores', 'una lista', [1, 2, 3, 4]]
>>> l[0:2]= [] #elimino elementos
>>> l
['una lista', [1, 2, 3, 4]]
>>> l[1:1] = ["una cadena"] #inserto elementos
>>> l
['una lista', 'una cadena', [1, 2, 3, 4]]
```

For:

Representa una forma de recorrer una lista, usando una variable. La variable va tomando diferentes valores que forman la lista, en orden, según la posición.
Esta forma de recorrer la lista se llama iterativa y cuando la variable pasa a un nuevo valor de la lista se dice iteración.
Dentro del for va un iterable.

Las listas permiten la definición por compresión, creando una nueva lista y aplicando una expresión de otra lista que cumpla una determinada condición.

[expresion for variables in iterable if condición]

range(): genera una lista de números enteros, sin incluir el elemento ingresado. Si se pasan dos argumentos, el primero indica el inicio y el segundo el fin (sin incluirlo). Y si pasamos tres argumentos, el tercero indica el salto.

While:

Esta forma de iteración se realiza mediante una condición, esto quiere decir que mientras se cumpla la condición, se realiza las sentencias que queremos.

```
while condición:
        sentencia
```

4 – Tuplas

Una tupla es una colección ordenada de elementos. Para definir una tupla se usan los paréntesis y los elementos separados por comas.

t = (“hola”, “mundo”)

Las tuplas son indexadas por lo que podemos usar el operador “[ ]” para tomar un elemento. También, las tuplas son iterables y, también se puede aplicar el slicing sobre ellas. Consumen menos recursos que las listas.

La característica que poseen las tuplas es la no modificación, inserción y eliminación. Es decir, son inmutables, una vez creadas no se pueden modificar y tienen tamaño fijo. Los String también son inmutables.

Dato:

Se puede hacer una tupla del estilo:

l = [0, “Hola”, True]
t = (1, l)

Y también se puede: *t[1].append(“Chau”) o l.append(“Chau”)*

Esto es posible ya que la tupla no esta en el mismo lugar de la memoria que la lista. Lo que pasa en este ejemplo es que la tupla guarda el lugar de la memoria de la lista, mejor dicho, el puntero de memoria de la lista. Al modificar la lista, no pasa nada pues ya esta guardado el puntero que ocupa la lista en la memoria. No se puede modificar en la tupla la dirección del puntero donde se encuentra la lista.