CURSO PYTHON

IVAN OSUNA AYUSTE

Índice ¿Que vamos a ver en el curso?

1
- ¿Qué es Python?
- Características básicas
- Ejemplos de uso

2
Introducción a lenguaje de programacion Python
Hello World, Definicion Variables, Operaciones Basicas

3
Estructuras de control de flujo parte 1
Sentencia if, else if, else

4
Operadores Binarios
Uso de import

Índice ¿Que vamos a ver en el curso?

1. ¿Qué es Python?

Caracteristicas Básicas

- **Es un lenguaje de programación interpretado**
- **Lenguaje de programación multiparadigma**
  - Soporta POO
  - Programación imperativa
  - Programación funcional
- **Lenguaje Interpretado**
- **Usa tipado dinámico y conteo de referencias para administracion de memoria.**
- **Es multiplataforma**
- **Usa licencia de código abierto**
  - Además es compatible con algunas versions de GNU (a partir de version 2.1.1)

Otros datos de interés

- **Filosofia muy similar a la de Unix, legibilidad y transparencia.**
- **Lenguaje de alto nivel**
- **Web official: https://www.python.org/**
- **Posee una gran biblioteca estandar para multitud de tareas (modulos de python)**
- **Existen 2 versiones:**
  - Python 2.x
  - Python 3.x
- **Extensiones mas comunes**
  - .py, .pyc, .pyd, .pyo, .pyw

Ejemplos de uso Python

Gestores de E-books

Servicios Streaming como Flumontion o Youtube

Software Center de Ubuntu
Software para Google

Desarrollo de Cliente official de DropBox

Gestores de listas de E-mail

Python permite escribir programas compactos y legibles. Los programas en Python son típicamente más cortos que sus programas equivalentes en C, C++ o Java

Por estos motivos:

- **Los tipos de datos de alto nivel permiten expresar operaciones complejas en una sola instrucción**
- **La agrupación de instrucciones se hace por sangría en vez de llaves de apertura y cierre**
- **No es necesario declarar variables ni argumentos.**

2. Introducción

“Distintos tipos de variables ”

python powered

```
print("Hello, world!")
```

- Imprimir consola
  - ✓ Uso de sentencia print
- Comentarios
  - ✓ Uso de # para una línea
  - ✓ Uso de “”” al principio y fin para multi-linea

Primer programa con Python

Ejercicio 1.1 : ¡Hola Mundo!

```
1    #Bienvenido a Python
2    #¡Hola Mundo!
3
4    print("Hello World")
```

2. Introducción

“Distintos tipos de variables ”

python powered

```
print("Hello, world!")
```

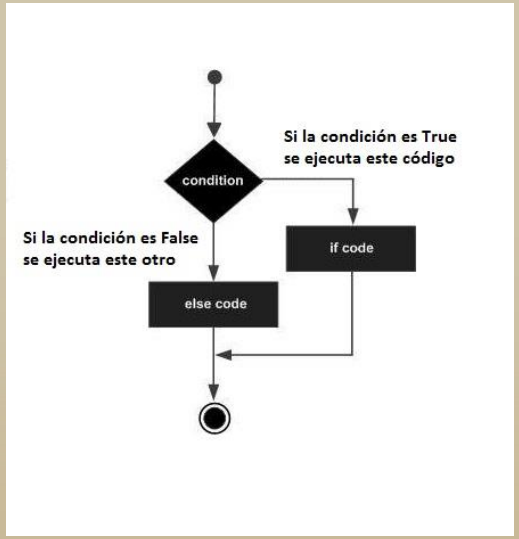
- **Ejercicio 1.2 - Comentario y print**
  Define diversas variables e imprímelas por pantalla
- **Ejercicio 1.3 – InputConsola**
  Introduce el nombre de tu padre y edad, guárdalo en una variable e i mprímelo por pantalla
- ***Ejercicio 1.4* – InputConsola uso de Eval**
  Hacer un programa que recoja las edades de tus padres por pantalla y que nos devuelva la suma de ella haciendo uso de eval.
- ***Ejercicio 1.5* – Operaciones Basicas1**
  Ejercicios básicos de: +, -, *, /, %, operaciones…
- ***Ejercicio 1.6* – Operaciones Basicas2**
  Ejercicios básicos abs, potencia, raíz, min, max…
- **Ejercicio 1.7 - OperacionesBasicas con texto**
  funciónes len, join, Split, substring…

3. Estructuras de control parte 1

“if, elif, else”

{ if else }

DIAGRAMA DE FLUJO BASICO:



ESTRUCTURA BASICA IF-ELSE:

```
1  if (condición1)
2     sentencia1
3  else if (condición2)
4     sentencia2
5  else if (condición3)
6     sentencia3
7  ...
8  else
9     sentenciaN
```

3. Estructuras de control parte 1

“if, elif, else”

- **Ejercicio 2.1 - Estructuras de control flujo1**
  Ejemplos sencillos if else, primer vistazo a funciones.

- **Ejercicio 2.2 - Estructuras de control flujo2**
  funciones: esVocal, contar vocales de un texto, if y else.

4. Operadores Binarios

“Uso de import”

OPERADORES A NIVEL DE BIT / OPERADORES BINARIOS:

Operador	Descripción	Ejemplo
&	and	r = 3 & 2 # r es 2
|	or	r = 3 | 2 # r es 3
^	xor	r = 3 ^ 2 # r es 1
~	not	r = ~3 # r es -4
<<	Desplazamiento izq.	r = 3 << 1 # r es 6
>>	Desplazamiento der.	r = 3 >> 1 # r es 1

SOLO OPERAN CON ARGUMENTOS ENTEROS.

4. Operadores Binarios

“Uso de import”

- **La sentencia import se utiliza para importar el contenido de otros módulos en Python.**
- **Para importar un modulo se usa import, seguido del nombre del paquete (si hay) mas el nombre del modulo (sin el .py) que se quiera importar.**

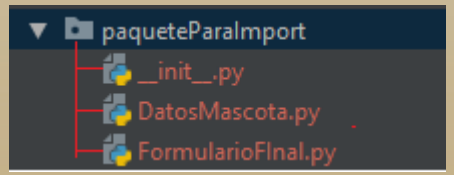
Python tiene sus propios módulos, los cuales forman parte de su **librería de módulos estándar**, que también pueden ser importados.

```
import modulo  # importar un módulo que no pertenece a un paquete
import paquete.modulo1  # importar un módulo que está dentro de un paquete
import paquete.subpaquete.modulo1
```

4. Operadores Binarios

“Uso de import”

- **Para ver un ejemplo mas claro, podemos crearnos una carpeta y dentro nuestros archivos.py (módulos) con nuestro código.**
- **Podemos importar el contenido de uno de ellos, en otro fichero, cuando nosotros queramos, haciendo uso de esta sentencia.**
- **Además, podemos acceder a sus variables como si de una clase se tratase.**
- **Debemos de crear primeramente un fichero llamado __init__.py que puede estar lleno o vacío.**
- **Y a continuación los demás ficheros con nuestro código.**



4. Operadores Binarios

“Uso de import”

- **Una vez creada esta estructura, solo tendremos que poner al principio del código, la sentencia import y el nombre del modulo/fichero de esa carpeta que queramos importar.**

- **<u>También debemos de saber:</u>**
  - **Los paquetes pueden contener otros sub-paquetes**
  - **Los módulos no necesariamente, deben pertenecer a un paquete.**
  - **Podemos importar todos los elementos de un modulo poniendo al final “*”**
    - **Paquete.modulo import ***

4. Operadores Binarios

“Uso de import”

- **Ejercicio 3.1 -OperadoresBinarios1**
  Evaluación básica con operadores binarios

- **PaqueteParaImport – Carpeta (ejercicio 3.2)**
  - **Init.py**
  - **DatosMascota.py**
  - **FormularioFinal.py**

5.Funciones

“Funciones básicas y recursividad”

Una función es un fragmento de código con un nombre asoc iado que realiza una serie de tareas y devuelve un valor.

- **La definición de funciones se realiza mediante la instrucción def + un nombre de función para el cuál, aplican las mismas reglas que para el nombre de las variables - seguido de paré ntesis de apertura y cierre. Finaliza con dos puntos (:)**

- **Si la función hace retorno de datos, se le puede asignar a un a variable.**

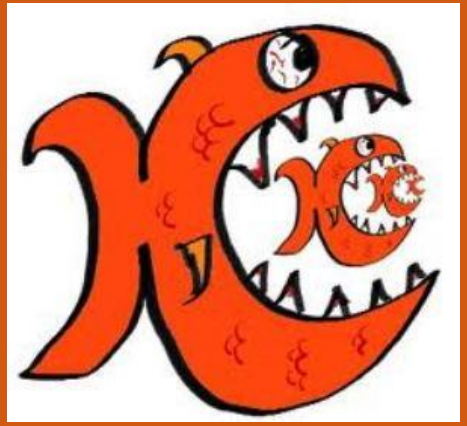
- **Un parámetro es un valor que la función espera recibir c uando sea invocada (puede recibir: ninguno o los que se ne cesiten) *a modo de variables, a fin de poder utilizarlos co mo tales, dentro de la misma función.***

```
def funcion():
    return "Hola Mundo"

frase = funcion()
print frase
```

5.Funciones

“Funciones básicas y recursividad”



Se denomina llamada recursiva (o recursividad), a aquellas funciones que en su algoritmo, hacen referencia sí misma.

- **Las llamadas recursivas suelen ser muy útiles en casos muy puntuales, pero debido a que es muy fácil caer en iteraciones infinitas, deben extremarse las medidas**
- **Solo se deben de utilizarse cuando sea estrictamente necesa rio y no exista una forma alternativa viable, que resuelva el pr oblema evitando la recursividad.**
- **Python admite las llamadas recursivas, permitiendo a una fun ción, llamarse a sí misma, de igual forma que lo hace cuando llama a otra función.**

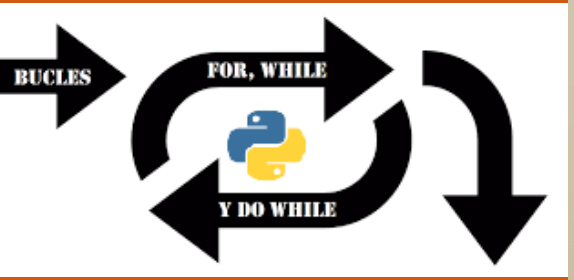
5.Funciones

“Funciones básicas y recursividad”

- **Ejercicio 4.1 – funciones1**
  función suma, perímetro, imprimirXlista…
- **Ejercicio 4.2 – funciones2recursividad**
  funciones con recursividad

6.Estructuras de control parte 2

“For”



Un bucle es una estructura de control que repite un bloque de instrucciones.

- **Un bucle for es un bucle que repite el bloque de instrucciones un número predeterminado de veces.**
- **El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.**

```
for variable in elemento iterable (lista, cadena, range, etc.):
    cuerpo del bucle
```

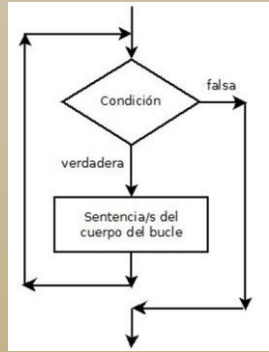
```
#bucle for
for i in  [0,1,2,3,4] :
    print ("Iteración")
print("fin")

#bucle for con range()
for i in range(0,5):
    print("Iteración")
print("fin")
```

Tenemos varias formas de definir un for:

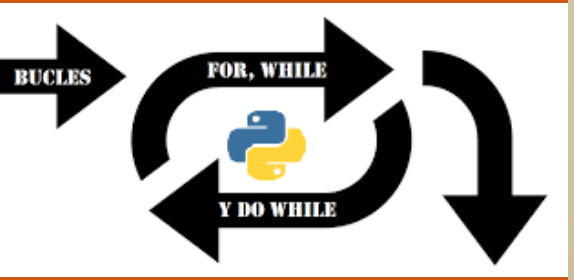
-A través de una tupla

-A través de función range() …



6.Estructuras de control parte 2

“While ”

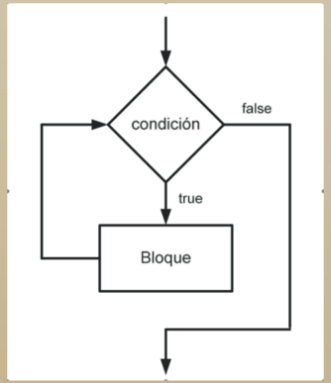


Un bucle while permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (mientras la condición tenga el valor True).

- **Un bucle for es un bucle que repite el bloque de nstrucciones un número predeterminado de veces.** **i**
- **El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.**

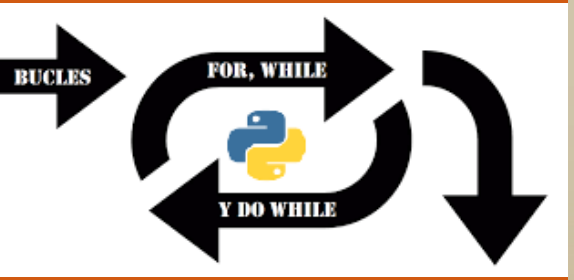
```
while condicion:
    cuerpo del bucle
```

```
i=1
while i<=4:
    print (i)
    i+=1
print("fin de la ejecución")
```



6.Estructuras de control parte 2

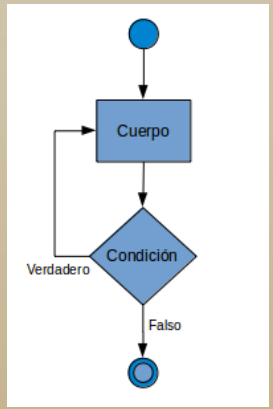
“Do-While ”



También llamado hacer-mientras, es una estructura de control de la mayoría de los lenguajes de programación estructurados cuyo propósito es ejecutar un bloque de código y repetir la ejecución mientras se cumpla cierta condición expresada en la cláusula while

```
// Ciclo do-while
do {
    cuerpo
} while (condición);
```

En Python NO existe esta sentencia

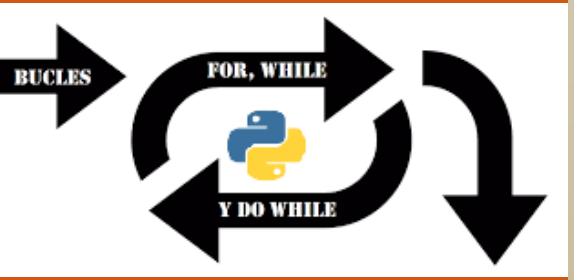


Pero podemos emularlo evaluando la condición al principio en un while, de la siguiente forma, en base a los ejemplos del: for y while:

```
i = 1
while True:
    print(i)
    i = i + 1
    if(i > 5):
        break
```

6.Estructuras de control parte 2

“for, while, do-while”



- **Ejercicio 5.1 – EjemplosBuclesDiapositivas**
  Los ejemplos mostrados en ellas.
- **Ejercicio 5.2 – BuclesFor**
  Ejercicios con bucles for
- **Ejercicio 5.3 – BuclesWhile**
  Ejercicios con bucles while
- **Ejercicio 5.4 – Repaso de for/while**

7.Menú

“ Validar Email”

Formato E.mail

XXXX@bbbb.com.net

NOMBRE DOMINIO EXTENSIÓN

Vamos a crear un programa con una estructura de menú. Dependiendo del estado o del caso en que nos encontremos hara una opción del menú u otra.

- **Vamos a trabajar con el fichero RellenaValidarEmail, el cual tiene un pseudocódigo para ir siguiendo la estructura del programa.**

- **Si el Email es validado la función devuelve true.**
- **Crearemos dos funciones:**
  - **mostrarError: la usaremos cada vez que quera mos mostrar un error por pantalla.**
  - **ValidarEmail: contiene todos los estados del menú.**
    - **Estado_Nombre**
    - **Estado_Dominio**
    - **Estado_Extension**

7.Menú

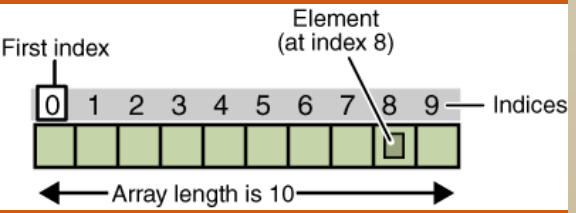
“Validar Nºteléfono”

En este segundo ejemplo, vamos a validar un numero telefonico

- **Vamos a trabajar con el fichero ValidarTelefonoRellenar, el cual tiene un pseudocódigo para ir siguiendo la estructura del programa.**
- **Iremos pasando de un estado a otro en base a lo que vayamos leyendo en el numero telefónico.**
- **Crearemos una función llamada normalizarTelefono y fuera de la función devolveremos el numero normalizado.**
- **Validaremos números telefónicos en base a los siguientes tipos:**
  - **+999 999 999 999**
  - **+0000-999-999-999-999**
  - **0999 999 999 999**
  - **999 999 999**

8.Listas y Arrays

“Tuplas/Arrays”



Python, posee 3 tipos de estructura más complejos que los básicos , que admiten una colección de datos. Estos tipos son: **Listas, Arrays y diccionarios. Las listas y Arrays en Python se les llama tuplas**

- **Una tupla es una variable que permite almacenar varios dat os inmutables (no pueden ser modificados una vez creados) de tipos diferentes. Ejemplo:**

```
mi_tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25)
```

Se puede acceder a cada uno de los datos mediante su índice correspondie nte, siendo 0 el índice del primer elemento:

```
print mi_tupla[1]  # Salida: 15
```

También se puede acceder a una porción de la tupla, indicando (opcionalme nte) desde el índice de inicio hasta el índice de fin:

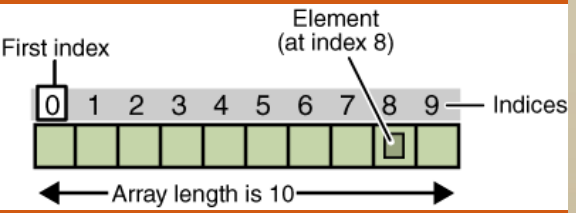
```
print mi_tupla[1:4]  # Devuelve: (15, 2.8, 'otro dato')
print mi_tupla[3:]   # Devuelve: ('otro dato', 25)
print mi_tupla[:2]   # Devuelve: ('cadena de texto', 15)
```

Otra forma de acceder a la tupla de forma inversa (de atrás hacia adelante), es colocando un índice negativo:

```
print mi_tupla[-1]  # Salida: 25
print mi_tupla[-2]  # Salida: otro dato
```

8.Listas y Arrays

“Listas”



Una lista es similar a una tupla con la diferencia fundamental de que permite modificar los datos una vez creados

```
mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]
```

A las listas se accede igual que a las tuplas, por su número de índice:

```
print mi_lista[1]    # Salida: 15
print mi_lista[1:4]  # Devuelve: [15, 2.8, 'otro dato']
print mi_lista[-2]   # Salida: otro dato
```

Las lista NO son inmutables: permiten modificar los datos una vez creados:

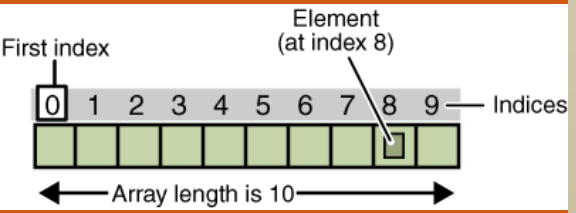
```
mi_lista[2] = 3.8  # el tercer elemento ahora es 3.8
```

Las listas, a diferencia de las tuplas, permiten agregar nuevos valores:

```
mi_lista.append('Nuevo Dato')
```

8.Listas y Arrays

“Diccionarios”



Mientras que a las listas y tuplas se accede solo y únicamente por un número de índice, los diccionarios permiten utilizar una clave para declarar y acceder a un valor:

```
mi_diccionario = {'clave_1': valor_1, 'clave_2': valor_2, \
                  'clave_7': valor_7}
print mi_diccionario['clave_2']  # Salida: valor_2
```

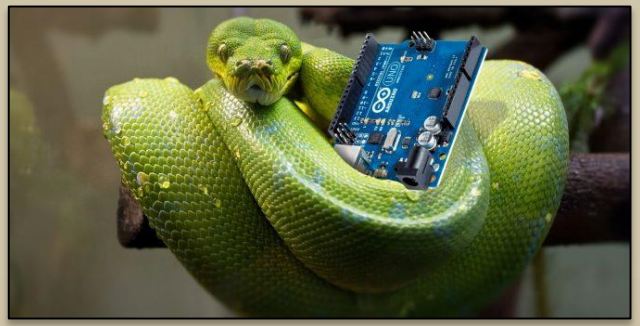
Un diccionario permite eliminar cualquier entrada:

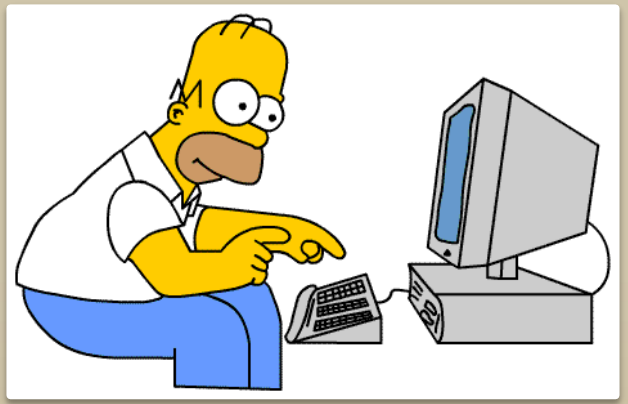
```
del(mi_diccionario['clave_2'])
```

Al igual que las listas, el diccionario permite modificar los valores

```
mi_diccionario['clave_1'] = 'Nuevo Valor'
```

- **Ejercicio 7.1 – TuplasDiccionario**





¡Muchas Gracias programadores!