IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

PYTHON

0. Escenario

Día 1 de python.

1. Características de Python

Es sensible a las mayúsculas. VARIABLE, Variable, variable y VaRiAbLe son todas palabras distintas para el intérprete de Python.

Los textos deben encerrarse entre comillas (”). Los números no.

Para nombrar variables y archivos no se puede utilizar ninguna de las palabras reservadas del lenguaje.

Los comentarios en el código empiezan por # y continúan hasta el final de línea

El separador decimal es el punto (.)

Para delimitar bloques no se usan ni paréntesis ni llaves. Se usa la indentación. Cuidado porque el tabulador no es lo mismo que los espacios en blanco...

2. Variables

Las variables son contenedores de valores. Estos valores pueden cambiar a lo largo de la ejecución del programa.

El nombre de las variables en Python debe cumplir:

- Empieza por una letra o un guión bajo.
- No empieza por un número
- Solo caracteres alfanuméricos (A-z. 0-9,)
- Python es sensible a mayúsculas por lo que Nombre no es lo mismo que nombre
- El nombre no puede ser el mismo que ninguna de las palabras reservadas de python

Para poder acceder a estos contenedores, las variables deben tener un nombre que solo utilice letras y números, preferiblemente letras del alfabeto inglés (nada de ñ, Ç, ü, ó,...).
Se asigna valor a las variables de derecha a izquierda usando el signo igual.

```
valor = 23
```

También es posible asignar varios valores a varias variables simultáneamente:.

```
dia, mes, anho = 13, 3 ,2023
```

El valor a la derecha puede ser una expresión formada por el cálculo de otras variables.

```
valor = dato1 + dato2

valor = dato1 – dato2

valor = dato1 * dato2

valor = dato1 / dato2

valor = dato1 // dato2  # División entera, 9 // 4 daría 2
```

Las variables pueden ser globales si están creadas fuera de la definición de una función. En caso de crear una variable dentro de una función esta será local permaneciendo la global inalterada.

Para cambiar el valor de una variable global dentro de una función es necesario la palabra clave global y el nombre de la variable.

```
x = “Un texto”

def miFuncion():
     global x
     x = “Otro texto”

print(x)
miFuncion()
print(x)
```

En las expresiones con variables se utilizan diferentes operadores que podemos clasificar en:

- Aritméticos: + - * / % ** //
- Asignación: = += -= *= /= %= //= **= &= |= >>= <<=
- Comparador: == != > < >= <=
- Lógicas: and or not
- Identidad: is is not
- Membresía: in not in
- Operadores de bit: & | ^~ << >>

3. Comandos básicos

Un básico en todos los lenguajes de programación es la posibilidad de intercambiar información con el usuario a través de teclado y pantalla. Además estos comandos son útiles para depurar y testear código

IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

mediante la introducción a mano de valores "controlados" y mostrando textos informativos a modo de banderas.

Para mostrar información en pantalla usamos el comando print.

```
print("Esto es un texto")

print("Texto en una línea\nY en otra")

print("El total es ", valor)
```

Como puedes ver en los ejemplos, podemos mostrar una salida de texto. Recuerda que al ser texto debe ir encerrada entre comillas. También podemos mostrar texto formateado con caracteres de escape como el \n que indica salto de línea. Otra posibilidad es la de mostrar el valor de variables. En el último ejemplo aparece una concatenación de cadena de texto y variable usando la coma.

```
nombre = input("Indica tu nombre: ")
```

El comando input permite que el usuario introduzca una cadena de texto hasta que pulse ENTER. Opcionalmente, permite pasar como parámetro una cadena de texto que se mostrará por pantalla sin necesidar de tener que usar un print.

El valor que escribe el usuario queda asignado y guardado a la variable de la izquierda de la misma manera que dábamos valor a variables en el apartado anterior.

```
edad = int(input("Indica tu edad: "))
```

Si quieres forzar a que el dato introducido por el usuario se interprete como un tipo de dato concreto se pueden utilizar conversores como int(texto) que transforma una cadena de texto en un entero.

Analicemos esta última sentencia. Python determina que es una asignación de un valor a la variable edad así que primero ejecuta la parte derecha. En la parte derecha primero ejecuta la parte más interna que es el input. En cuanto el usuario presione ENTER, el texto introducido se pasará como parámetro a la función int() que tratará de convertir ese texto a un número entero. Una vez la expresión de la derecha acabe de generar el valor se ejecutará la asignación del valor a la variable,

4. IF

El comando if permite ejecutar unos comandos u otros en función del valor de una condición. Esta condición debe ser booleana y evaluarse como verdadero o falso.

```
if nota < 5:
     print("Suspenso")
elif nota == 10:
     print(Matrícula")
else:
     print("Aprobado")
```

La primera línea del comando if contiene una condición booleana que finaliza con dos puntos(:). Si su evaluación es verdadero ejecutará el primer bloque de sentencias. Recuerda que en python los bloques se indentan.
Si la condición se evalúa como falso se ejecutará la siguiente condición en la sentencia elif.
Así seguirá sucesivamente hasta que encuentre una condición que se evalúe a verdadero. Opcionalmente podemos añadir una sentencia else que será la última opción por defecto a la que se llegará si anteriormente ninguna otra condición se evaluó como verdadero.

Los operadores para las condiciones son los típicos de comparación (==, !=, <, >, >=, y <=) y los típicos lógicos (and y or)

```
if num >= 10:
     if num <= 20:
          print(“Entre 10 y 20”)
     else:
          print(“Por encima de 20”)
else:
     print(“Por debajo de 10”)
```

En este código se puede apreciar la importancia de la indentación para que python tenga claro los bloques.

En algunos casos, condiciones muy complejas se pueden simplificar usando un enfoque distinto. Comprobar una contraseña es sencillo porque o coincide exactamente o no. En otras ocasiones nos interesa comparar textos sin tener en cuenta mayúsculas. Por ejemplo, un usuario que dice que su color favorito es el rojo, otro dice que es el ROJO y otro dice que es el Rojo… todos se refieren a lo mismo.

```
if preferido==”rojo” or preferido == “ROJO” or preferido == “Rojo”:

if str.lower(preferido) == “rojo”
```

La primera sentencia comprueba algunos (pero no todos) las posibles combinaciones de mayúsculas y minúsculas. Estaría bien pero incompleta. En cambio, la sengunda sentencia es 100% completa. Todos los casos están previstos ya que usa la versión en minúscula de los datos que se guardaron en la variable preferido.

5. Cadenas

Las cadenas son textos que deben ir entrecomillados. Puedes usar la comilla doble (”) o la sencilla (’) pero una vez tomes la decisión debes mantenerla. Dentro de las comillas hay algunos caracteres como las comillas y la barra que no se pueden poner directamente porque python los interpretaría no como texto y sí como símbolo reservado. Para poder usarlos es necesario “escaparlos” con una barra invertida como prefijo. Así, si queremos ver una comilla doble dentro de un texto tendremos que ponerla de la siguiente manera:

```
print(“… gira hasta escuchar \”click\” para desbloquear…”
```

En las cadenas de texto podemos introducir un salto de línea con el carácter de escape \n o también podemos usar la triple comilla doble.

```
texto = “””Esto es una cadena
```

IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es


```
con saltos de línea
y algún    tabulador"""
```

Algunos comandos interesantes para trabajar con cadenas de texto

```
len(texto) #Devuelve la longitud del texto
texto.capitalize() #Primer carácter en mayúscula, el resto minúsculas.
texto.lower()
texto.upper()
texto.title() #primeras letras de cada palabra en mayúscula, resto minúscula
texto3 = texto1+texto2 #Devuelve la concatenación sin espacios
```

En python no hay un tipo de datos para almacenar un carácter, se guardan en cadenas de longitud 1. El tipo string es un array de unicodes. Puedes acceder a una posición del array usando corchetes y la posición (empezando desde cero)

texto = "Hola mundo"
print(texto[0]) #Imprime la H
print(texto[2:6] #imprime "la m"

6. Matemáticas

Para algunas de las funciones matemáticas es necesario importar la librería math mediante el siguiente comando que colocaremos en las primeras líneas del archivo.

```
import math
```

Desde este momento tendremos disponibles en nuestro programa constantes definidas en el módulo math como pi, e, infinito,..

```
circunferencia = 2 * math.pi * r
```

Fíjate que para usar la constante es necesario inidicar el módulo, punto y el nombre de la constante.

También tenemos funciones disponibles a las que pasamos uno o varios parámetros y nos devuelven un valor

```
math.ceil(valor) #Devuelve el entero más pequeño pero mayor que valor
math.gcd(valores) #Devuelve el máximo común divisor.
math.factorial(valor) #Devuelve el factorial.
math.pow(mantisa,exponente)
math.trunc(valor) #Devuelve la parte entera
```

7. Otros módulos.

Ahora que ya hemos visto la utilidad de los módulos puedes consultar otros módulos de python en su documentación oficial: https://docs.python.org/es/dev/library/math.html o en otras web contrastadas como https://www.w3schools.com/python/default.asp

8. Tipos

Python no tiene comandos para definir el tipo de las variables, de hecho las variables no se crean hasta que se les asigna valor por primera vez. Posteriormente se puede cambiar el tipo de la variable haciendo una nueva asignación con otro tipo de valor.

```
x = 5 #tipo entero
x = “Hola mundo” #ahora tipo cadena
```

Es posible forzar el tipo de una variable mediante casting en el momento de la asignación.

```
w = str(5)
x = int(5)
y = 5
z = float(5)
```

Con el comando type obtenemos el tipo de datos de una variable.

```
print(type(y))
```

Los siguientes tipos son nativos para python:

- Tipos de texto: str
- Tipos de número: int, float, complex
- Tipos de secuencias: list, tuple, range
- Tipos de mapa: dict
- Tipos de conjuntos: set, frozenset
- Tipo booleano: bool
- Tipos binarios: bytes, bytearray, memoryview
- Tipo nulo: NoneType

9. Variables de valores múltiples

En python existen tipos que pueden almacenar múltiples valores en una sola variable:

- Dictionary
- List
- Set
- Tuple

```
diccionario = {clave1:valor1, clave2:valor2, clave3:valor3}
lista = [valor1, valor2, valor3]
conjunto = {valor1, valor2, valor3}
tupla = (valor1, valor2, valor3)
```

Nombre	Tipo	Tamaño variable	Ordenado	Permite cambio	Duplicados	Símbolo
Diccionario	dict	✓	✓	✓	✗	{clave:valor }
Lista	list	✓	✓	✓	✓	[valor]
Conjunto	set	✓	✗	✗	✗	{valor}
Tupla	tuple	✗	✓	✗	✓	(valor)

10. Diccionario

Nombre	Tipo	Tamaño variable	Ordenado	Permite cambio	Duplicados	Símbolo
Diccionario	dict	✓	✓	✓	✗	{clave:valor }

Los diccionarios se definen como parejas de clave valor separadas por dos punto (:). Cada una de estas parejas se separan con coma (,) y se enmarcan con llaves ({ })

```
pilotos = {14:”Fernando Alonso”, 55:”Carlos Sainz”, 44:”Lewis Hamilton”,
1:”Max Verstappen”}
```

En este ejemplo se ha usado como clave el dorsal de cada piloto y como valor el nombre del mismo. Durante la temporada podremos añadir y eliminar pilotos sin problema. En este ejemplo se entiende bien que en un diccionario no puede haber elementos duplicados, no tiene sentido en este ejemplo.

Es posible ordenar el diccionario para recorrerlo y mostrarlo en algún orden determinado.

Los diccionarios incluyen algunos métodos para su gestión, entre ellos:

- clear()
- copy/(
- fromkeys()
- get()
- items()
- keys()
- pop()
- popitem()
- setdefault()
- update()
- values()

11. Lista

Nombre	Tipo	Tamaño variable	Ordenado	Permite cambio	Duplicados	Símbolo

Lista	list	✓	✓	✓	✓	[valor]

Usamos listas cuando queremos una lista ordenada como podría ser una receta de cocina o un log del servidor. Es importante mantener el orden y también es posible repetir elementos.

```
receta = [“Mezclar”, “Reservar”, “Amasar”, “Reservar”, “Amasar”, “Horno”]
log = [“192.168.0.1”, “192.168.0.200”, “192.168.0.1”]
```

Los elementos de una lista están ordenados y se accede a ellos mediante su posición por ello se permiten valores duplicados. Los elementos se insertan al final de la lista.
Una lista puede contener valores de tipos distintos.

Podemos crear variables de tipo lista dando valores o mediante un constructor.

lista = [23, ”Hola”, 56.7]
otra_lista = list((”hola”, ”adios”, ”chao”)) #Ojo a los dobles paréntesis

Se acceder a los elementos de una lista mediante su posición entre corchetes empezando por cero. Si usamos valores negativos en la posición estamos indicando desde el final.
Se puede conseguir una sublista de un rango usando [inicio:fin]

Se puede usar la palabra reservada in para comprobar si un determinado elemento está en la lista.

```
lista = [“uno”, “dos”, “tres”]
if “dos” in lista:
     print(“Aparece en la lista”)
```

Las listas incluyen algunos métodos para su gestión, entre ellos:

- append()
- clear()
- copy()
- count()
- extend()
- index()
- insert()
- pop()
- remove()
- reverse()
- sort()

12. Conjunto

Nombre	Tipo	Tamaño variable	Ordenado	Permite cambio	Duplicados	Símbolo
Conjunto	set	✓	✗	✗	✗	{valor}

En ocasiones nos puede interesar una lista no ordenada ni con elementos repetidos, un conjunto. Un ejemplo del mundo real es una lista de la compra donde no hay repeticiones y el orden no es importante.

```
compra = {“huevos”, “aceite”, “azucar”}
```

Al no estar ordenado no se puede acceder a los elementos por suposición, es necesario recorrerlos y usar la palabra reservar **in**.

Toda la lógica matemática sobre conjuntos se aplica en este tipo de datos permitiendo que se puedan hacer uniones, intersecciones o diferencias entre conjuntos.

Los conjuntos incluyen algunos métodos para su gestión, entre ellos:

- add()
- clear()
- copy()
- difference()
- difference_update()
- discard()
- intersection()
- intersection_update()
- isdisjoint()
- issubset()
- pop()
- remove()
- symmetric_difference()
- symmetric_difference_update()
- union()
- update()

13. Tupla

Nombre	Tipo	Tamaño variable	Ordenado	Permite cambio	Duplicados	Símbolo
Tupla	tuple	✗	✓	✗	✓	(valor)

Usaremos una tupla cuando necesitemos una secuencia fija de valores. Los elementos de esta secuencia se pueden repetir y, al estar ordenados, se puede acceder a ellos mediante un índice.

Un ejemplo de la vida real en el que usaríamos tuplas puede ser el resultado de la quiniela. El tamaño de la tupla no va a cambiar, los elementos tienen que estar ordenados y se pueden repetir pero no modificar.

```
jornada15 = (“1”, “X”, “X”, ”1”, ”2”)
```

Las tuplas incluyen algunos métodos para su gestión, entre ellos:

- count()
- index()

14. For

Usado para iterar por secuencias. Recuerda que las cadenas son secuencias también.

```
for x in “Esto es un texto”:
     print(x)

lista = [“lunes”,”miércoles”,”viernes”]
for y in lista:
     print(y)
```

Fíjate en los dos puntos (:) que separan la cabecera y el bloque de código que se ejecutará. Ojo con la indentación del bloque.

En los bucles se puede usar el comando **break** para salir del bucle sin acabar todas las iteraciones. También es posible usar el comando **continue** para escapar de esa iteración en ese punto y continuar con la siguiente.

Los bucles for pueden tener opcionalmente un bloque else: que se ejecutará en último lugar siempre que no se haya usado un break para salir del bucle sin acabar todas las iteraciones.

En ocasiones nos puede interesar iterar sobre números y para ello se usar la función range()

```
for x in range(3):  #0, 1, 2
     print(x)

for x in range(2,6): #2, 3, 4, 5
     print(x)

for x in range(2,10,3): #2, 5, 8
     print(x)
```

15. While

El comando while evalúa una condición y repite el bloque del bucle mientras la condición se siga cumpliendo la condición.

```
i = 1
while i < 5:
     print(i)
     i = i + 1
```

En el bucle while también se pueden usar los comandos **break** y **continue** para salir del bucle o de la iteración actual respectivamente.

16. Funciones

Para definir funciones se utiliza la palabra reservada **def** junto con el nombre de la función y los argumentos separados por comas (,) entre paréntesis.
Si el argumento pasado es una tupla, la variable del argumento debe empezar por asterisco (*)

Si el argumento pasado es un diccionario, la variable del argumento debe empezar por doble asterisco (**)

```
def funcion1():
     print("Sin argumentos")

def funcion2(nombre, apellidos)
     print(nombre, apellidos)
```

Las funciones pueden devolver un valor como resultado de su ejecución y esto se hace con la palabra reservada **return** junto con el valor.

17. Clases

Una clase define un nuevo tipo de datos generalmente relacionado con el mundo real. Este nuevo tipo de datos tiene una serie de propiedades y métodos que modifican su estado y se comunica con el exterior.

```
class Coche:
     tipo_combustible = ("Diesel", "Gasolina", "Eléctrico", "Híbrido", "GLP")
```

A crear una variable de este nuevo tipo de datos se le llama instanciar, crear una instancia de esa clase o directamente crear un objeto.

```
c1 = Coche()
print(c1.tipo_combustible[1])
```

Las clases tienen una función constructora que se ejecuta en el momento de crear la instancia, la función __init__() con doble guión bajo antes y después.

```
class Coche:
     def __init__(self, comb, id):
          self.combustible = comb
          self.id = id

c2 = Coche("GLP", 1)
print(c2.id," -> ",c2.combustible
```

Las clases pueden definir métodos para modificar y estado. Se hace mediante funciones a las que obligatoriamente hay que pasar un mínimo de un parámetro llamado **self** que hace referencia a su propia instancia.

```
class Coche:
     def __init__(self,comb,id):
          self.combustible = comb
          self.id = id
          self.encendido = False

     def encender(self):
          self.encendido = True
```

IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es


```
    def apagar(self):
          self.encendido = False

    def esta_encendido(self):
          return self.encendido
```

Las propiedades de un objeto y el propio objeto se pueden eliminar con el comando **del**.

```
del c2
```

18. Módulo

Los módulos son agrupaciones de variables y funciones con una utilidad común. Se guardan en un archivo con extensión .py y se importan en otros archivos mediante el comando **import**.

La sintaxis para llamar a una variable o función de un módulos consiste en escribir el nombre del módulos y el nombre de la variable/función separadas por punto.