PRO

UT2: Introducción al lenguaje Python

José Antonio
Benítez Chacón

Índice

1. Introducción

Python

Características:

- Es un lenguaje de alto nivel.
- Es un lenguaje interpretado.
- Es un lenguaje que usa tipado dinámico y está fuertemente tipado.
- Su sintaxis es simple y legible.
- Es un lenguaje de programación multiparadigma (orientación a objetos, programación estructurada, programación funcional, etc.).

La **programación estructurada** es un paradigma que se basa en utilizar solo funciones (o subrutinas) y tres estructuras de control:

- secuencias de instrucciones (una tras otra)
- selecciones, elecciones o condicionales
- iteraciones, ciclos o bucles

1. Introducción

Una instrucción es una “unidad lógica” que realiza una acción concreta.
Esta acción puede ser una **asignación** (tras realizar alguna operación)

```
a = 2 + len("Hola")
edad = input("Dime tu edad")
```

Puede ser un **condicional**:

```
if int(edad) < 18:
    print("Eres menor de edad")
else:
    print("Eres mayor de edad")
```

Puede ser un **bucle**:

```
for i in range(5):
    print(i)
    i += 1
```

O puede ser una llamada a una **función** (predefinida o definida por el usuario):

```
print("Hola, ¿cómo estás?")
saludar("Alberto")
```

2. Asignaciones

Una asignación consiste en asignar a una variable o una constante un valor.
El nombre de esta variable o constantes (y también el de otras cosas como funciones) se llama **identificador**.

```
a = 2 + len("Hola")
edad = input("Dime tu edad")
```

Los identificadores de variables han de comenzar por un guion bajo (“_”) o por una letra (minúscula por convención). El resto del nombre puede estar formado por caracteres alfanuméricos y guiones bajos.

Hay distinción de mayúsculas y minúsculas. Es decir, las variables:
`varAux = 12` y `varAUX = 13`
son variables diferentes.

Además, un identificador no podrá ser nunca una de las palabras reservadas por el lenguaje (*while, for, if, elif, break, pass*, etc.)

NOTA: En **Python** no hay distinción estricta entre **variables** y **constantes** (no deben cambiar su valor una vez establecido), pero, por convención, una constante tendrá su nombre entero en mayúsculas.

```
MAYORIA_EDAD = 18
ABREV_DOCTOR = "Dr."
```

3. Condicionales

Sirven para tomar una decisión en base a una condición. Si la condición se cumple, se ejecuta el bloque asociado. Este bloque estará indentado a la derecha, y cuando termine, se volverá a la indentación que tenía.

if: evalúa la condición y si se cumple ejecuta el bloque asociado
elif: (opcional): contracción de “else if”. Sirve para no tener que anidar bloques if de forma innecesaria. Siempre va precedido de un *if* (o *elif*) que no han cumplido sus condiciones. elif tiene una condición propia que, si se cumple, sirve para ejecutar el bloque asociado.
else: sirve para asociar un bloque de código cuando no se han cumplido las evaluaciones anteriores del bloque condicional.

```
letra = input("introduce una letra:").lower()

if letra in ["a", "e", "i", "o", "u"]:
   print("Has introducido una vocal no acentuada.")
elif letra.isalpha() and len(letra) == 1:
   print("Has introducido una letra.")
else:
   print("No has introducido una letra.")
```

3. Condicionales

Desde la versión 3.10 también está **match** (que es equivalente a switch de otros lenguajes).

Con match se evalúa una expresión y se comprueba sus resultados con case. Se utiliza el guion bajo (“_”) como comodín, por si no coincide con ningún patrón:

```
animal = input("introduce un animal:")
match(animal):
    case "perro":
        print("El animal elegido es un perro.")
    case "gato":
        print("El animal elegido es un gato.")
    case _:
        print("El animal elegido no es un perro ni un
gato.")
```

4. Bucles

Un **bucle** es una estructura que repite el bloque de instrucciones que contenga mientras se cumpla una condición.

while

Evalúa una condición y, mientras se cumpla, ejecuta el bloque de instrucciones asociado

```
i = 0
while i < 10:
    print(f"Hola:+{i}")
    i += 1
else:
    print(f"i es {i}. Ya no es menor que 10")
```

Puede tener un **else** al final, que se ejecuta al terminar el bucle (si este no ha sido interrumpido por un **break**)

4. Bucles

for

Con un bucle for se recorren los elementos de una cadena de texto o una estructura de datos (lista, diccionario, etc.). Se ejecuta tantas veces como elementos haya (caracteres en la cadena de texto, elementos en las demás) y almacena el elemento sobre el que está iterando en ese momento en una variable.

```
for palabra in ["árbol", "silla", "marea"]:
   print(palabra.upper())


for caracter in "DAM1":
   print(caracter)
else:
   print("Terminé de recorrer la cadena" )
```

También puede tener un **else** al final, que se ejecuta al terminar el bucle (si este no ha sido interrumpido por un **break**)

4. Bucles

for (con range)
range() genera un iterable (una especie de lista en la que se generan los elementos al recorrerla, tiene evaluación perezosa).
la sintaxis de range puede ser:

range(5) -> Genera un iterable [0,5), es decir, desde el 0 hasta el 4 (el 5 es el primer elemento en quedar fuera).
range(2,5) -> Genera un iterable [2,5), es decir, desde el 2 hasta el 4 (el 5 es el primer elemento en quedar fuera).
range (0,10,2) -> Genera un iterable [0,10), es decir, desde el 0 hasta el 9 (el 10 es el primer elemento en quedar fuera) pero que se recorre de 2 en 2 elementos (empezando por el 0).

```
a = range(0,10,2)
for i in a:
   print(i)

lista = ["árbol", "silla", "marea"]
for i in range(len(lista)):
  print(f"El elemento {i} de lista es {lista[i]}")
```

4. Bucles

break

Instrucción que sirve para romper el bucle en el que nos encontramos (el programa continuaría por la línea siguiente al bucle).

continue

Instrucción que sirve para terminar la iteración en curso del bucle. Se pasa a la siguiente.

```
for numero in range(20):
    print("Recorro lista: analizando número " +str(numero))
    if(numero %2 == 0):
        print(f"{numero} es par")
    else:
        continue

    if(numero == 10):
        print("Número 10 encontrado" )
        break
```

5. Tipos de datos

Python es un lenguaje fuertemente tipado. Veamos primero algunos de los tipos básicos:

Cadenas de texto (str)
Puede guardar caracteres, palabras, frases y párrafos.
Se pueden declarar entre comillas simples, entre comillas dobles o entre triples comillas dobles para indicar texto multilínea.

Además, existen las **fstring**, que sirven para poder interpolar expresiones o variables en la propia cadena, sin tener que fragmentarla y concatenar con el operador +.

```
a = 'soy una cadena.'
b = "yo también soy una cadena de texto."
c = """Yo soy
una cadena de texto
multilínea."""
d = f"<<{a}>> y <<{b}>> son cadenas de texto."
print(d)
print(d+c)
```

5. Tipos de datos

Cadenas de texto (str)

Formato:

```
personas = {"Juan": {"edad":23, "altura":1.75, "peso":70},
   "Alicia": {"edad":22, "altura":1.60, "peso":59},
}


print("Juan tiene {} años. Mide {} m y pesa
{:.2f}kg".format(personas["Juan"]["edad"],personas["Juan"]["altura"],personas["Juan"]["peso"]))

print("Nombre\t\tEdad\tAltura\tPeso")
for elem in personas:

print(f"{elem:<8}\t{personas[elem]["edad"]:<4}\t{personas[elem]["altura"]:.2f}\t{personas[elem]["peso"]:.2
f} ")


print("El otro dia vi un {queVi} cerca de {donde} {haciendo}."format(donde="en la plaza",
haciendo="jugando con un ovillo", queVi="un gato"))

print("{1}, {0}.".format("Pablo","Buenos días"))
```

5. Tipos de datos

Cadenas de texto (str): algunos métodos

Método	Funcionalidad	Método	Funcionalidad
len(cadena)	Devuelve el número de caracteres de cadena	cadena.split(a)	Devuelve una lista con los elementos de dividir cadena con a. Si no se indica a, es el espacio " ".
cadena.count(a)	Cuenta las apariciones de a en cadena	cadena.capitalize()	Devuelve cadena con el primer carácter en mayúsculas
cadena.find(a)	Devuelve la posición de la primera aparición de a en cadena. -1 si no está.	cadena.lower()	Devuelve cadena en minúsculas
cadena.index(a)	Devuelve la posición de la primera aparición de a en cadena. Eleva excepción no está.	cadena.upper()	Devuelve cadena en mayúsculas
cadena.startswith(a)	Devuelve True si cadena empieza por a	cadena.swapcase()	Devuelve cadena con mayúsculas y minúsculas cambiadas.
cadena.endswith(a)	Devuelve True si cadena termina en a	cadena.title()	Devuelve cadena con las palabras en minúsculas salvo la primera letra de cada una
cadena.isalnum()	Devuelve True si cadena es un alfanumérico	cadena.center(num, a)	Devuelve la cadena centrada en un bloque de num caracteres, completando con a. Si no se especifica a, es espacio " "
cadena.isalpha()	Devuelve True si cadena es texto	cadena.ljust(num, a)	Como center, pero justifica a la izquierda
cadena.isdigit()	Devuelve True si cadena es solo dígitos	cadena.rjust(num, a)	Como center, pero justifica a la derecha
cadena.isnumeric()	Devuelve True si cadena es solo números (incluye caracteres para fracciones y números romanos)	cadena.zfill(num)	Rellena cadena con ceros a la izquierda hasta alcanzar los num caracteres si hiciese falta
cadena.isspace()	Devuelve True si cadena es solo espacios	nexo.join(lista_cadenas)	Devuelve una cadena en base a unir las cadenas de lista_cadenas mediante nexo
cadena.islower()	Devuelve True si cadena tiene al menos una letra y todas las letras que contiene están en minúsculas.	cadena.splitlines()	Devuelve una lista cuyos elementos son cada una de las líneas de cadena
cadena.isupper()	Devuelve True si cadena tiene al menos una letra y todas las letras que contiene están en mayúsculas.	cadena.partition(a)	Divide una cadena en una lista de 3 cadenas: lo que hay a la izquierda de la primera ocurrencia de a, a y lo que hay a la derecha de la primera ocurrencia de a. Si no encuentra a, devuelve una lista con cadena y dos cadenas vacías
cadena.istitle()	Devuelve True si todas las palabras de cadena comienzan por mayúsculas y las demás son minúsculas		
cadena.format(...)	Da formato al texto de cadena (visto antes)		
cadena.replace(a,b)	Devuelve una cadena donde cada ocurrencia de a ha sido sustituida por b		
cadena.strip()	Quita los espacios al inicio y al final de cadena		
cadena.lstrip()	Quita los espacios al inicio (izquierda) de cadena		
cadena.rstrip()	Quita los espacios al final (derecha) de cadena		

5. Tipos de datos

números

enteros (int): No hay un máximo como en otros lenguajes
decimales o coma flotante (float)
complejos (complex)

```
entero = 231
decimal = 23.12
complejo = 2 +3j
```

booleanos (bool)

```
a = True
b = 10 < 9
c = a or b
d = not c
print(d)
```

None

Valor asignado a variables que se quiere instanciar pero si asignarles aún valor.

```
a = None
print(a)
```

5. Tipos de datos

Es necesario hacer conversión de tipos (o casting) si necesitamos convertir datos a otros tipos (si es posible)

```
a = input("Introduce un número:")
print(f"El doble de a es: {a*2}")
print(f"El doble de a es: {int(a)*2}")

a = 34
cadena = "Jorge tiene "+str(a)+" años."
print(cadena)
```

Cada tipo de datos, además, admite unos operadores. Hay ciertos operadores que pueden adoptar una forma abreviada.

```
a = 23
b = "Hola"
a += 4 #equivale a a = a + 4
b += ", ¿cómo estás?" #equivale a b = b +", ¿cómo estás?"
```

6. Estructuras de datos

Una variable permite almacenar, temporalmente, un dato para utilizarlo en nuestro código.

Las estructuras de datos permiten almacenar colecciones de datos para acceder a ellos de manera sencilla y dinámica:

Ejemplos de estas colecciones son:

- Listas
- Tuplas
- Diccionarios
- Conjuntos (Sets)

6. Estructuras de datos

Listas (list)

Es una colección de datos mutable. Puede contener elementos repetidos. Tiene un orden. **[0,1,2]**

```
a = ["manzana", "pera", "plátano"]
print(a)
a.append("uva")
a.remove("pera")
print(a)
print(a.index("plátano"))

b = list(range(27))
b.extend(["a","e","i","o","u"])
print(b)
for index,elem in enumerate(b):
    if str(elem).isalpha() and index%2 == 0:
        b[index]=elem.upper()
    print(index,elem)
print(b)
```

6. Estructuras de datos

Tuplas (tuple)

Es una colección de datos inmutable. Puede contener elementos repetidos. Tiene un orden. **(0,1,2)**

```
a = ("rojo", "verde", "azul")
print(a)
print("violeta" in a)

lista = []
for i in range(26):
   lista.append(chr(ord("a")+i))
b = tuple(lista)
print(b)
print(b[3])
for i in range(25):print(b[i],end="->")
else: print(b[25])
```

6. Estructuras de datos

Diccionarios (dict)

Es una colección de datos mutable. Es una colección de pares clave:valor (separadas por 2 puntos). Las claves no pueden estar repetidas. **{"a":"xxx", "b":"yyy"}**

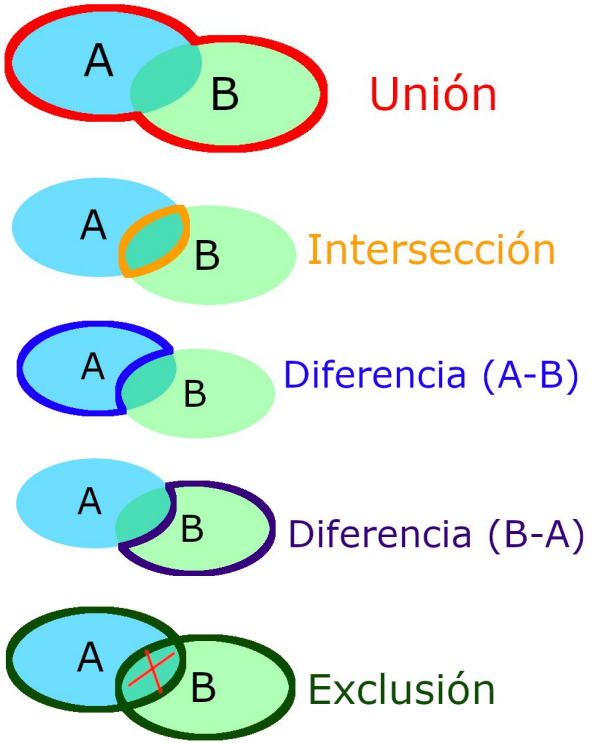
```
diccio = dict()
diccio[0]="Mañana"
diccio[1]="Tarde"
diccio[2]="Noche"

for elem in diccio:
   print(elem, diccio[elem])
print(diccio.get(1))
print(diccio.get(10,"No existe"))

diccio2 = {"rojo":{"rgb":"FF0000","calor":"cálido","opuesto":"cian"},
       "verde":{"rgb":"00FF00","calor":"frío","opuesto":"magenta"},
       "azul":{"rgb":"0000FF","calor":"frío","opuesto":"amarillo"},
       "cian":{"rgb":"00FFFF","calor":"frío","opuesto":"rojo"},
       "amarillo":{"rgb":"FFFF00","calor":"cálido","opuesto":"azul"},
       "magenta":{"rgb":"FF00FF","calor":"cálido","opuesto":"verde"}}

for elem in diccio2:
   print(f"{elem.upper()}:\n\tEs un color {diccio2[elem]["calor"]}.")
   print(f"\tSu código RGB es #{diccio2[elem]["rgb"]}.")
   print(f"\tSu color opuesto es el {diccio2[elem]["opuesto"]} (cuyo RGB es
#{diccio2[diccio2[elem]["opuesto"]]["rgb"]}).")
```

6. Estructuras de datos

Conjuntos (set)

Es una colección de datos mutable. No hay elementos repetidos. **{1,2,3}**

```
pares=set(range(0,30,2))
mult3=set(range(0,34,3))
print(f"pares: {pares}")
print(f"múltiplos de 3: {mult3}")
pares.add(30)
mult3.remove(33)
print(f"pares: {pares}")
print(f"múltiplos de 3: {mult3}")
print(f"unión: {pares.union(mult3)}")
print(f"intersección: {pares.intersection(mult3)}")
print(f"diferencia (pares menos mult3): {pares.difference(mult3)}")
print(f"exclusión: {pares.symmetric_difference(mult3)}")
```



6. Estructuras de datos

Resumen

	Listas	Tuplas	Diccionarios	Conjuntos
Declaración	a1= [1,2,3] a2 = **list**()	a1 = (1,2,3); a3 = (1,) a2 = **tuple**(lista)	a1 = {"a":"b"} a2 = **dict**()	a1 = {1,2,3} a2 = **set**()
Tamaño	**len**(a1)	**len**(a1)	**len**(a1)	**len**(a1)
Acceso a elemento	a1[0] a1[-1]	a1[0] a1[-1]	a1["a"] a1.**get**("a)	a1[0]
Modificación	a1[0] = "Hola"	-	a1["a"] = "Hola"	-
Añadir elemento	a1.**append**(elem) a1.**insert**(position, elem)	-	a1[elem] = valor	a1.**add**(elem) a1.**update**(lista) a1.**update**(set)
Eliminar elemento	a1.**remove**(elem) (error si no en lista) a1.**pop()** y a1.**pop**(pos) (y devuelve el elemento) **del** a1[0]	-	a1.**pop**(clave) (elimina por clave y recupera valor asociado) a1.**popitem**() (elimina último y devuelve tupla clave, valor) **del** a1[clave]	a1.**remove**(elem) (error si no en set) a1.**discard**(elem) a1.**pop**() (borra elemento cualquiera)
Eliminar la estructura	**del**(a1) o **del** a1	**del**(a1) o **del** a1	**del**(a1) o **del** a1	**del**(a1) o **del** a1
Vaciar la estructura	a1.**clear**()	-	a1.**clear**()	a1.**clear**()
Copia superficial	a1.**copy**() **copy.copy**(a1)	**copy.copy**(a1)	a1.**copy**() **copy.copy**(a1)	a1.**copy**() **copy.copy**(a1)
Copia profunda	**copy.deepcopy**(a1)	**copy.deepcopy**(a1)	**copy.deepcopy**(a1)	**copy.deepcopy**(a1)
Otras	a1 **+** a2 (unir listas) a1.**extend**(a2) (extender lista) a1.**reverse**() (invierte el orden) **reversed**(a1) (iterable con orden invertido) a1.**sort**() (orden ascendente) a1.**count**(elem) (apariciones de elem en la lista) a1.**index**(elem) (posición del elemento. Excepción si no está)	a1 **+** a2 (unir tuplas) a1.**count**(elem) (apariciones de elem en la tupla) a1.**index**(elem) (posición del elemento. Excepción si no está)	dict.**fromkeys**(a1) (crea nuevo diccionario con claves de a1 y valores None) dict.**fromkeys**(a1, val) (crea nuevo diccionario con claves de a1 y valores val) a1.**items**() (vista de tuplas clave, valor) a1.**keys**() (vista de claves) a1.**values**() (vista de valores)	a1.**union**(a2) (nuevo set) a1.**difference**(a2) (nuevo set) a1.**difference_update**(a2) (actualiza set) a1.**intersection**(a2) (nuevo set) a1.**intersection_update**(a2) (actualiza set) a1.**symmetric_difference**(a2) (nuevo set) a1.**symmetric_difference_update**(a2) (actualiza set) a1.**isdisjoint**(a2) (True si no hay elementos comunes) a1.**issubset**(a2) (True si todos los elementos de a1 están en a2) a1.**issuperset**(a2) (True si todos los elementos de a2 están en a1)

7. Funciones

Las funciones son bloques de código o secuencias de instrucciones que se agrupan con un identificador, para poder reutilizar ese código en diferentes partes del código sin tener que duplicarlo.

Los valores que necesita una función como entrada para ejecutarse se llaman parámetros. Además, en Python, una función puede o no devolver un resultado.

Para definirla se utiliza la palabra reservada **def**, que precede al indicador de la función y los parámetros de la misma. Esta definición indica el comportamiento que tendrá la función cuando sea llamada, pero de por sí, sin ser llamada, no se ejecuta ninguna vez.

```
def genera_lista(a,b,n):
    lista = random.sample( range(a,b),n)
    lista.sort()
    return lista

print(genera_lista( 100,120,5))
```

7. Funciones

Ejemplo de funciones por defecto de Python: https://docs.python.org/3/library/functions.html

Función	Descripción	Función	Descripción
abs(a)	Valor absoluto de a: abs(1) = 1 abs(-31) = 31	open(f, m, …)	Sirve para abrir un fichero f en modo m (lectura, escritura, etc). Se verá más adelante.
all(a)	Devuelve True si todos los elementos de a evalúa a True. También si a no tiene elementos.	ord(caracter)	Devuelve el código UNICODE del carácter car
any(a)	Devuelve True si alguno de los elementos de a se evalúa a True. Si a está vacío devuelve False.	chr(num)	Devuelve el carácter asociado al número num en UNICODE
divmod(a,b)	Devuelve una tupla con el cociente y el resto de la división entera de a // b	pow(a, b)	Devuelve la potencia de elevar a a b
enumerate(a)	Devuelve un iterable, donde cada elemento es una tupla de la posición del elemento en a y su valor.	print(*a, sep=" ", end="\n")	Muestra por pantalla tantos elementos como se pase (*a), separados por sep (por defecto espacio) y terminando en end (por defecto salto de línea)
exec(a)	Permite la ejecución del texto a que se le pase como si fuese código: `exec("print(f\"Hola, {len(\"Pep\")}.\")")`	reversed(a)	Devuelve un iterable en orden invertido de a.

filter(funcion, iter)	Devuelve un iterable con los elementos de iter que evalúan funcion como True:
	`la = [1,2,3,4]`
	`print(list(filter(lambda i: i % 2 == 0, la)))`
		round(num)
		round(num, deci)
			Redondea un número num a deci decimales (si no se especifica, son 0 decimales).

format(a, formato)	Devuelve a con el formato especificado:
	`a = 2`
	`print(format(a,".2f"))`
		slice(stop)
		slice(start, stop)
		slice(start, stop, step)
			Sirve para definir los límites de un slice (fragmento) de cadena o estructura:
			`print(list("abcdefghijklmn"[slice(0,10,2)]))`

input(mensaje)	Muestra un mensaje por consola y lee una línea de la consola y la convierte a str.	sorted(iterable, key=None, reversed=False)	Devuelve una lista ordenada en base a los elementos de un iterable, siguiendo el criterio key (por defecto si no se especifica) y sin invertir (por defecto)
len(a)	Devuelve el tamaño de a (cadena o estructura)	sum(iterable, start=0)	Realiza el sumatorio de los elementos de iterable, comenzando por defecto en 0.
max(a,b,c,...n)	Devuelve el máximo entre los parámetros que se pasen de entrada	zip(*iterables)	Itera sobre varios iterables en paralelo, devolviendo tuplas de los elementos que ocupan la misma posición (traspone una matriz)
min(a,b,c,...n)	Devuelve el mínimo entre los parámetros que se pasen de entrada