Tema 3. Introducción a Python

October 1, 2020

1 Resumen

- ¿Qué es python?
- Características de Python
- Componentes de un programa
- Tipos de datos en Python
- Almacenando datos en memoria: variables y constantes
- Imprimiendo datos
- Instrucciones: operadores
- Fuentes de datos: teclado y números aleatorios
- Comentarios

2 ¿Qué es Python?

- Lenguaje de alto nivel, orientado a objetos, de código abierto
- 1991 por Guido Van Rossum
- El nombre viene de Monty Python's Flying Circus
- Propósito general
- Fácil de usar y de aprender
- Interpretado
- Sensible a mayúsculas
- Sensible a sangrías (indentación)
- Tipado dinámico (no hace falta definir el tipo de los datos, lo averigüa directamente)
- Librería muy extensa
- Recolector de basura (no es necesario liberar la memoria que ya no se va a usar, el intérprete lo hace automáticamente)
- Extensible (podemos añadir instrucciones al lenguaje, pero eso implica modificar el intérprete)
- Soporta programación estructurada, orientada a objetos y funcional
- Muchos intérpretes: CPython, IPython
- Ideal para hacer programas rápidos
- Filosofía: Debería haber una forma sencilla y obvia de hacer las cosas y solo una

2.1 Versiones

- 0.9 (1991)
- 2.7 (2010)
- 3.0 (2008) No hay compatibilidad con las versiones 2.x
- 3.7 (2018)

- 3.8 Actual, hay una 3.9 prevista en breve
- Cultura general: versiones alfa y beta

3 Componentes de un programa

- programa = datos + algoritmo (instrucciones) + comentarios
- Tipos de datos que hay en Python (tema 3)
- Instrucciones: operadores (tema 3) + instrucciones de control de flujo (tema 4) + instrucciones de E/S (tema 3)
- Tipos de comentarios (tema 3)

4 Tipos de datos en Python

- Números: tres tipos distintos
  - enteros (tipo int)
  - decimales, reales o flotantes (tipo float)
  - complejos (tipo complex)
- Letras o caracteres (tipo str)
- Booleanos o lógicos: para representar cosas que son verdaderas o falsas (tipo bool)
- Python tiene tipado dinámico: no hace falta declarar el tipo de dato que voy a guardar en memoria. En otros lenguajes hace falta decir el tipo de dato antes de guardarlo (hay que decirle al lenguaje si voy a guardar un número, una letra, etc.)

4.1 Enteros

- Tipo int (Ejemplos: 4_324, 324, 2, 1, 4, -21)
- El valor máximo depende de la memoria del ordenador y es prácticamente infinito
- Se pueden usar guiones bajos para facilitar la lectura de números largos agrupando las cifras de 3 en 3: 2_243_342

4.2 Reales o decimales

- Tipo float, se usa el punto decimal (Ejemplos: 3.4 45_453.54 222.2)
- Se guardan en 64 bits. Lo más pequeño que podemos guardar es +-4.9E-324. Lo más grande +-1.79769313486231570E+308
- Problema de precisión: solamente se guardan las primeras 15 o 16 cifras, el resto se redondea: 12345678901234567.0 y 12345678901234568.0 para Python son el mismo número (1.2345678901234568E16 = 1.2345678901234568 * 10^16)

4.3 Complejos

- Tipo complex (3 + 3j, 2.5 + 4.2j)
- Python es uno de los pocos lenguajes que soportan números complejos

5 Booleanos o lógicos

- Tipo bool

- Guardan algo que es verdadero o falso
- True o False (con la primera letra en mayúscula)
- Hay cierta compatibilidad con 1 y 0 (pero no se recomienda poner 1 en lugar de True o 0 en lugar de False)

6 Caracteres

- Tipo str
- Representan una cadena de letras (`"hola" "qué tal estás" 'adiós' """esto es una cadena muy larga que podría ocupar varias lineas""" '''esto también sería una cadena muy larga que ocuparía varias líneas en mi programa'''`)
- Se pueden usar indistintamente comillas simples `''` o dobles `""`
- Con 3 comillas se crea una cadena de caracteres que puede ocupar varias líneas
- Se recomienda no poner más de 79 caracteres en una línea

7 Guardando datos en memoria: variables

- Una variable es una posición de memoria a la que damos un nombre y en la que vamos a guardar un dato
- Tipo inferido: el tipo de la variable se infiere directamente por el dato que voy a guardar, no hace falta declarar de qué tipo va a ser la variable como en otros lenguajes
- Tipo dinámico: puedo cambiar el tipo de una variable asignándole un valor de otro tipo. Muy mala práctica de programación. Se penalizará.
- Uso la función `type()` para saber el tipo
- No puedo declarar una variable si no la inicializo (si no tiene valor), aunque puedo declararla y ponerla a None (se debe evitar)

[2]:
```
# Declaro una variable llamada a y guardo un 8 en ella
# La variable es de tipo int
a = 8
# Le pido al intérprete que me diga el valor de a
a
# b es una variable de tipo str
b = "hola"
# c es de tipo bool
c = False
# d es de tipo float
d = 2.5
# Le pido al intérprete que me diga el valor de c
c
# Cambio el tipo de a para que sea str (no recomendado)
a = "adios"
# Le pido al intérprete que me diga el tipo de a
type(a)
# Le pido al intérprete que me diga el tipo de c
type(c)
# Declaro una variable sin tipo ni valor (no recomendado)
```

```
v = None
```

8 Reglas para nombres de variables

- Solo pueden empezar por letra o _ (guión bajo)
- Después pueden ir letras, _ o números
- Nombres válidos y5, *eeeee* _____w
- Nombres inválidos 3333333a, !!a a?, a%
- Python es sensible a mayúsculas
- Recomendación: Usar nombres que tengan significado
- Convención: usar minúsculas y si el nombre es más de una palabra guiones bajos, no empezar los nombres por guión bajo
- También se puede usar camello bajo (lower camel case)
- Hay una serie de palabras prohibidas: int, float, complex, bool, str, etc.

[9]:
```
# dinero y Dinero son variables distintas
dinero = 5000
Dinero = 1222
esto_es_un_nombre_valido_de_variable_que_cumple_las_recomendaciones = "hola"
# nombre que no cumple la convención porque empieza por mayúscula
Esto_no_la_cumple = False
# nombre en camello bajo, menos recomendable pero aceptado
estoTambienPodriaValer = 2.3
# Puedo declarar e inicializar varias variables a la vez
var1, var2, var3 = 33, False, "adios"
# Pero esto daría error porque si declaro n variables debo dar n valores a la␣
 ↪derecha
var4, var5 = 322
# Declaración de varias variables con el mismo valor
var6 = var7 = var8 = 34.6

# Se puede cambiar el contenido de una variable
mi_variable = 8
mi_variable = 12

# El signo = no es igualdad matemática sino asignación: toma lo de la derecha y␣
 ↪guárdalo en la izquierda
primera = 12
# Copiamos el contenido de primera en segunda (ahora segunda vale 12)
segunda = primera
# Cambiamos el contenido de primera
primera = 4
# Pero el valor de segunda seguirá siendo 12

# Cuando hay múltiples asignaciones, se realizan en paralelo
var1, var2 = 3, 5
```

```
# Ahora var2 vale 3 porque se hace antes de cambiar el valor de var1
var1, var2 = 12, var1

# Esto intercambia el valor de las variables
var1, var2 = var2, var1
```

9 Maneras de inicializar variables

- Dándole directamente un valor, a este valor se le llama literal (asignándole un literal)
- Dándole el valor de otra variable (asignándole otra variable)
- Dándole el valor con una expresión (un conjunto de variables, literales y operadores)
- Pidiéndole el valor al usuario
- Dándole un valor aleatorio

10 Constantes

- Variables cuyo valor no puede cambiar
- Python no tiene constantes
- Si ponemos el nombre de una variable en mayúsculas estamos indicando que no queremos que cambie (es solo una recomendación)

[12]:
```
# Al ponerlo en mayúsculas indico que es una constante
PI = 3.1416
# Pero nada me impide cambiar su valor o su tipo
PI = "no lo sé"
PI
```

[12]:
```
'no lo sé'
```

11 Instrucciones

- Control de flujo + operadores + E/S

11.1 Instrucciones para salida: impresión en pantalla

- Se utiliza la función print()
- Secuencias de escape (permiten añadir caracteres especiales al imprimir)

Secuencia de escape	Descripción
\b	borrar el último carácter
\t	tabulador
\r	retorno de línea
\n	nueva línea
\’	comilla simple ’
\”	comilla doble ”
\\	barra invertida \

[5]:
```
# imprimimos un 3
print(3)
# declaramos una variable e imprimimos su valor
a = 33
print(a)
# podemos imprimir varias cosas separándolas por comas
print(a, 22, 55)
print("El valor de la variable a es",a)
# Ejemplos de uso de secuencias de escape
cadena = "Esto es una cadena que se imprime en\ndos líneas"
print(cadena)
cad2 = "\t y aquí separo \t con tabulador"
print(cad2)
# Para poner comillas en una cadena
cad4 = "me ha dicho \"hola\""
print(cad4)
# Aunque es más fácil si usamos comillas simples fuera
cad5 = 'me ha dicho "hola"'
print(cad5)
```

```
3
33
33 22 55
El valor de la variable a es 33
Esto es una cadena que se imprime en
dos líneas
         y aquí separo   con tabulador
me ha dicho "hola"
me ha dicho "hola"
```

11.2 Operadores

- Varios tipos: aritméticos, sobre cadenas, afiliación/membresía, relacionales, lógicos, asignación, bit a bit e identidad

11.2.1 Asignación

- Asignan a una variable un valor
- El principal es = (asigna a la variable de la izquierda el valor que hay en la derecha)
- A la izquierda siempre debe haber una variable

11.2.2 Aritméticos

- `+` suma
- `-` resta
- `*` multiplicación
- `/` división
- `//` división entera (el se truncan los decimales del resultado)
- `%` módulo o resto de la división

- ** potencia

[2]:
```
var1, var2 = 8, 14
# Asigno a var3 la suma de las otras 2
var3 = var1 + var2
print(var1, "+", var2, "=", var3)
# División
var4 = var1 / var2
print(var1, "/", var2, "=", var4)
# División entera
var5 = var1 // var2
print(var1, "//", var2, "=", var5)
var6 = var1 % var2
print(var1, "%", var2, "=", var6)
var7 = var1 ** 2
print(var1, "** 2 =", var7)
# Si me salgo de rango en flotantes al multiplicar da infinito
print('1e308*1e308 = ',1e308*1e308)
# Pero al elevar a una potencia da error
# print(1e308 ** 2)
# Puedo mezclar datos de distintos tipos números
print('5.6 + 2 = ', 5.6 + 2)
# E incluso bool (no recomendado)
print(var1, "+ True =", var1 + True)
# División por cero
# print(6/0)
```

```
8 + 14 = 22
8 / 14 = 0.5714285714285714
8 // 14 = 0
8 % 14 = 8
8 ** 2 = 64
1e308*1e308 =  inf
5.6 + 2 =  7.6
8 + True = 9
```

12 Operadores relacionales

Operator	Description
==	Igual
!=	Distinto
>	Mayor
<	Menor
>=	Mayor o igual
<=	Menor o igual

- Al comparar string se compara el código Unicode/ASCII de cada letra. El Código de 'a' es 97, 'b' es 98, etc. mientras que el de la 'A' es 65, 'B' 66 y así sucesivamente.

[1]:
```
var1 = 4
var2 = 8
print(var1 == var2)
print(var1 >= var2)
# Se pueden mezclar tipos de números en las expresiones y el resultado es el␣
 ↪esperado
print (3 == 3.0)
# Incluso podemos comparar booleanos y números (no recomendado)
print(True == 1)
# "a" es menor que "b" porque el código de "a" es 97 y el de "b" 98
print ("a" > "b")
# "a" es mayor que "A" 97 > 65
print("a" > "A")
# Podemos encadenar comparaciones
print(3 < 5 > 7)
```

```
False
False
True
True
False
True
False
```

13 Operadores booleanos o lógicos

- Trabajan solamente con booleanos y el resultado es también un booleano
- `and`: verdadero sí y solo sí ambos operandos son verdaderos
- `or`: verdadero sí alguno de los dos es verdadero
- `not`: vuelve falso lo verdadero y viceversa
- Trabajan en cortocircuito: si saben la respuesta solo evaluando el primer operando no evalúan el segundo (poner siempre el más sencillo de evaluar primero)

[2]:
```
boo1, boo2 = True, False
boo3 = boo1 and boo2
print(boo3)
boo3 = boo1 or boo2
print(boo3)
print(not boo1)
# Como el primero es verdadero, tiene que comprobar también el segundo
print(3 > 2 and 5.2 > 8)
```

```
False
True
False
```

```
False
```

14 Operadores con String

- `+` concatena string
- `len()`: dice el tamaño de la cadena
- operadores de rebanado (slicing): permiten seleccionar partes de la cadena
- `string[numero]`: me da el carácter que hay en esa posición. El primero es el 0. Si me paso de índice tengo un error. Si el número es negativo empiezo contando por el final (el último es el -1)
- `string[numero1 : numero2]`: devuelve la parte de la cadena comprendida entre la posición numero1 y la numero2 (la última no incluida)
- El operador `*` vale para repetir cadenas

[15]:
```
s1, s2 = "hola", "adios"
# Concateno dos cadenas
s3 = s1 + s2
print("El valor de la cadena s3:", s3)
# Imprimo la longitud de la nueva cadena
print(len(s3))
print("El sexto elemento de s3 contando por el final:", s3[-6])
print("Los elementos entre la posición 2 y la 5 (no incluida):", s3[2:5])
# Esto me da desde la 4 hasta el final
print("Los elementos desde la cuarta posición hasta el final:", s3[4:])
# Y esto desde el principio hasta la 4, sin incluirla
print("Los elementos desde el inicio hasta la cuarta posición (no incluida):",␣
 ↪s3[:4])
# Esto me da toda la cadena
print("Toda la cadena (no muy útil):", s3[:])
print("Partes de s1 y de s3", s1[1:3], s3[5:8])
# Repitiendo cadenas
animando = "oe" * 5 + " " + "oe" * 2
print (animando)
```

```
El valor de la cadena s3: holaadios
9
El sexto elemento de s3 contando por el final: a
Los elementos entre la posición 2 y la 5 (no incluida): laa
Los elementos desde la cuarta posición hasta el final: adios
Los elementos desde el inicio hasta la cuarta posición (no incluida): hola
Toda la cadena (no muy útil): holaadios
Partes de s1 y de s3 ol dio
oeoeoeoeoe oeoe
```

15 Operadores de pertenencia

- De momento solo se pueden usar con variables de tipo str: `in` y `not in`

- Valen para saber si una subcadena está dentro de un str. Devuelven un bool.
- También se utilizan mucho con listas y tuplas (tema 5)

[8]:
```
print("¿Está 'la' en s3?", "la" in s3)
print("¿'la' no está en s3?", "la" not in s3)
```

```
¿Está 'la' en s3? True
¿'la' no está en s3? False
```

16 Entrada de datos por teclado

- Se utiliza la función `input("texto")`
- Lo que lee es siempre string. Para leer otro tipo de datos necesito hacer un casting (conversión de datos)

16.1 Casting

- Convertir un dato de un tipo a otro
- Casting implícito: se hace automáticamente, por ejemplo al sumar un int y un float todo se convierte a float
- Casting explícito: debo forzarlo, se utilizan las funciones `str()`, `int()`, `float()`, `bool()`, `complex()`
- `type(variable)`: devuelve el tipo de una variable
- `isinstance(variable, tipo)`: devuelve True si la variable es de ese tipo

[10]:
```
nombre = input("Introduce tu nombre: ")
print("¡Hola", nombre + "!")
# Hacemos un casting para que edad sea un número (si no es str)
# Si el usuario no introduce un número tendremos un error
edad = int(input("Dime tu edad "))
edad = edad + 1
print("El año que viene tendrás", edad)
```

```
Introduce tu nombre: Pepa
¡Hola Pepa!
Dime tu edad 18
El año que viene tendrás 19
```

[14]:
```
# Puedo hacer casting de números a bool, todo lo que no sea un 0 será True
var = bool(11)
print(var)
# También de str, todo lo que no sea "" (cadena vacía) es True
var = bool("False")
print(var)
# Con type puedo saber el tipo de una variable
print("El tipo de var es:", type(var))
print("¿Es var un booleano?:", isinstance(var, bool))
print("¿Es 8 un entero?:", isinstance(8, int))
```

```
# Cambio un str a entero
var1 = "122"
var1 = int(var1)
# Al cambiar de flotante a entero trunco los decimales
print(int(3.99))
```

```
True
True
El tipo de var es: <class 'bool'>
¿Es var un booleano?: True
¿Es 8 un entero?: True
3
```

17 Precedencia de operadores

- ()
- **
- +,- (cuando indican el signo de un número)
- * / % //
- + -
- <= < > >=
- == !=
- = %= /= //= -= += *=
- is, is not
- in, not in
- not, or, and
- En caso de duda se usan paréntesis

18 Números aleatorios

- Hay que importar la libreria random
- `random.random()`: genera un número flotante entre 0 y 1 (no incluido)
- `random.randrange(inicio, final)`: genera un entero entre inicio y final (no incluido)
- `random.randint(inicio, final)`: como randrange pero el último está incluido
- `random.uniform(inicio, final)`: como randrange pero genera flotantes (el final no está incluido)

[44]:
```
import random
a = random.random()
print(a)
b = random.randrange(1,10)
```

```
print(b)
c = random.uniform(1,10)
print(c)
```

```
0.10564120204598437
7
5.62622227941759
```

19 Comentarios

- Los comentarios valen para entender el código
- Son muy importantes en programación, y más este año
- La forma estándar es `#`: Python toma como comentario todo lo que hay desde la almohadilla hasta el final
- La convención es poner el comentario antes de la línea que se quiere comentar
- Para comentar varias líneas uso `""" """`, se les denomina docstring (texto de documentación). Mis programas deberían empezar siempre con un comentario de este tipo que incluya: el nombre del programador, la fecha, la versión del programa y una pequeña descripción de lo que hace
- NUNCA se borra código cuando se está programando

[ ]:
```
# Esto comenta la siguiente línea
# Declaro una variable entera
var6 = 88 # Este comentario es válido pero no sigue la convención
""" Esto es un comentario que
ocupa varias líneas"""
```