Introducción a Python

Python es un lenguaje de programación de alto nivel (su sintaxis e instrucciones se aproximan al lenguaje -inglés- natural), cuya filosofía hace hincapié en la legibilidad de su código. Se utiliza para desarrollar aplicaciones de todo tipo; por ejemplo, Instagram, Netflix y Panda 3D, están desarrolladas en este lenguaje.

Se trata de un lenguaje de programación multiparadigma: admite instrucciones para la computadora (programación imperativa), instrucciones para datos (programación orientada a objetos) y, en menor medida, la operación con funciones (programación funcional basada en el uso de funciones matemáticas, o más específicamente, en el cálculo lambda).

Es un lenguaje interpretado (sus programas, con extensión “py”, se ejecutan traduciendo en el momento sus instrucciones al lenguaje de la computadora, en oposición a los lenguajes traducidos, cuyos programas se traducen completos al lenguaje de la computadora generando archivos ejecutables), dinámico (sus variables representan valores que pueden cambiar su representación según lo que se les asigne, por ejemplo, números enteros, números reales, números complejos, cadenas de caracteres, etc.) y multiplataforma (hay versiones o intérpretes de Python para computadoras de cualesquiera arquitectura y sistema operativo).

Los programas en Python se pueden editar en distintos *IDE (Integrated Development Environment),* ambientes integrados de desarrollo (aplicaciones con editores especializados de programas que permiten ejecutarlos y depurar sus errores), como por ejemplo PyCharm, KDevelop, SlickEdit, Thonny, Visual Studio, Atom, LiClipse, Spyder, Pyzo, Geany, Wing; sin embargo, Python tiene su propio ambiente denominado *IDLE (Integrated Development and Learning Environment*, ambiente integrado de desarrollo y aprendizaje, que incluye una ventana de interfaz o *shell* para ejecutar instrucciones directamente o ver los resultados de programas, y se pueden abrir una o más ventanas de edición de programas como archivos de texto con menús de administración de archivos *(File)*, edición *(Edit y Format)*, para ejecutarlos *(Run)*, configurar el editor *(Options)* o consultar documentación del lenguaje *(Help)*.

Programas en Python

Comentarios

Los comentarios se pueden poner de dos formas. La primera y más apropiada para comentarios largos es utilizando la notación ''' comentario ''', tres apóstrofos de apertura y tres de cierre. La segunda notación utiliza el símbolo #, y se extienden hasta el final de la línea.

El intérprete no tiene en cuenta los comentarios, lo cual es útil si deseamos poner información adicional en el código. Por ejemplo, una explicación sobre el comportamiento de una sección del programa.

```
'''
Comentario largo en una o más líneas en Python
'''
print("Hola") # También es posible añadir un comentario al final de una línea
```

Variables

Las *variables* se utilizan en los programas para representar datos cuyos valores deben utilizarse reiteradamente o son desconocidos (porque se le piden al usuario del programa o son resultado de alguna operación); sus valores se definen de forma dinámica, lo que significa que no se tiene que especificar cuáles son sus tipos de antemano y pueden tomar distintos valores en otro momento, incluso de un tipo diferente al que tenían previamente. Se usa el símbolo `=` para asignar valores.

```
x = 1
x = "texto" # Esto es posible porque los tipos son asignados dinámicamente
```

Los nombres de variables pueden contener números y letras (incluyendo a la ñ y a las vocales acentuadas), pero deben comenzar por una letra, además existen 28 palabras reservadas que no pueden utilizarse como nombres de variables porque representan otras cosas en el lenguaje:

- `and`
- `assert`
- `break`
- `class`
- `continue`
- `def`
- `del`
- `elif`
- `else`
- `except`
- `exec`
- `finally`
- `for`
- `from`
- `global`
- `if`
- `import`
- `in`
- `is`
- `lambda`
- `not`
- `or`
- `pass`
- `print`
- `raise`
- `return`
- `try`
- `while`

Python es sensible a las mayúsculas, por lo que el nombre de una variable, en un programa, debe escribirse siempre igual.

Para la comprensión de los programas es recomendable que cada variable se utilice para un único propósito y tenga un nombre que aluda o represente a dicho propósito. También es recomendable que la primera vez que se use una variable se comente para qué se utilizará o cómo se actualizará en el programa o función que se la use.

Tipos numéricos

Existen tres tipos numéricos distintos: números enteros *(integers)*, números de punto flotante (*floating point* -una representación para números reales basada en la notación científica en base 2). Además, los booleanos *(booleans)* son un subtipo de los enteros (`1` representa el valor verdadero - `True` - y `0` representa el valor falso - `False`). Los enteros tienen una precisión ilimitada. La precisión y la representación interna de los números de punto flotante dependen de la máquina en la que se ejecuta el programa. Los números complejos *(complex)* tienen una parte real y otra imaginaria, cada una de las cuales es un número de punto flotante. Para extraer estas partes de un número complejo `z`, se utilizan `z.real` y `z.imag` (órdenes a datos).

Los números se crean mediante literales numéricos o como resultado de funciones y operadores incorporados. Los literales numéricos sin adornos (incluidos los números hexadecimales, octales y binarios) producen números enteros. Los literales numéricos que contienen un punto decimal o un signo de exponente producen números de punto flotante. Si se añade 'j' o 'J' a un literal numérico se obtiene un número imaginario (un número complejo con una parte real nula) que se puede añadir a un entero o a un flotante para obtener un número complejo con partes reales e imaginarias.

Python soporta completamente la aritmética mixta: cuando un operador aritmético binario tiene operandos de diferentes tipos numéricos, el operando con el tipo "más estrecho" se amplía al del otro, donde el entero es más estrecho que el punto flotante, que es más estrecho que el complejo. Una comparación entre números de diferentes tipos se comporta como si se compararan los valores exactos de esos números.

Las funciones `int()`, `float()` y `complex()` pueden utilizarse para producir números de un tipo específico cuando el argumento es una variable o expresión que representan un valor o resultado de otro tipo.

Todos los tipos numéricos (excepto los complejos) admiten las siguientes operaciones:

Operación	Resultado	Notas
x + y	suma de *x* e *y*	
x - y	diferencia de *x* e *y*	
x * y	producto de *x* e *y*	

Operación	Resultado	Notas
x / y	cociente de *x* e *y* en punto flotante	
x // y	cociente entero de *x* e *y*	(1)
x % y	resto entero de x / y	(2)
-x	*x* opuesto (cambio de signo)	
+x	*x* sin cambio (de signo)	
abs(x)	Valor absoluto o magnitud de *x*	
int(x)	*x* convertido a entero	(3)(6)
round(x[, n])	*x* redondeado a *n* dígitos, redondeando mitad a par. Si *n* se omite, su valor por defecto es 0.	(7)
float(x)	*x* convertido a punto flotante	(4)(6)
complex(re, im)	un complejo con parte real *re* y parte imaginaria *im* (con valor por defecto 0)	(6)
c.conjugate()	conjugado del número complejo *c*	
divmod(x, y)	par (x // y, x % y)	(2)
pow(x, y)	*x* a la potencia *y*	(5)
x ** y	*x* a la potencia *y*	(5)

El orden de precedencia de los operadores es

**

+x, -x

*, /, //, %

+, -

Notas

1. También se denomina división de enteros. El valor resultante es un entero, aunque el tipo del resultado no es necesariamente int. El resultado siempre se redondea hacia el infinito negativo: `1//2` es `0`, `(-1)//2` es `-1`, `1//(-2)` es `-1`, y `(-1)//(-2)` es `0`.

2. No para números complejos. En su lugar, convierta a flotantes usando abs() si es apropiado.

3. La conversión de punto flotante a entero puede redondear o truncar; vea las funciones math.floor() y math.ceil() para conversiones bien definidas.

4. *float* también acepta las cadenas "nan" e "inf" con un prefijo opcional "+" o "-" para No es un número (NaN) e infinito positivo o negativo.

5. Python define `pow(0, 0)` y `0 ** 0` como `1`, como es habitual en los lenguajes de programación.

6. Los literales numéricos aceptados incluyen los dígitos `0` a `9` y los caracteres `-` para signo, `.` para separar parte entera de fraccionaria, y `e` o `E` para separar significante de exponente en notación científica.

```
>>> 8.431e3
8431.0
>>> 9.5E-2
0.095
```

7. x se redondea al múltiplo de 10 más cercano a la potencia -n; si dos múltiplos están igual de cerca, el redondeo se hace hacia la opción par (así, por ejemplo, tanto `round(0.5)` como `round(-0.5)` son `0`, y `round(1.5)` es `2`). Cualquier valor entero es válido para n (positivo, cero o negativo). El valor de retorno es un entero si n se omite o es `None`. En caso contrario, el valor de retorno tiene el mismo tipo que x.

Módulos

Existen muchas propiedades que se pueden agregar al lenguaje importando módulos, que son "minicódigos" (la mayoría escritos también en Python) que proveen de ciertas funciones para realizar determinadas tareas. Un ejemplo es el módulo *os*, que provee acceso a muchas funciones del sistema operativo. Los módulos se agregan a los códigos escribiendo `import` seguida del nombre del módulo que queramos usar.

El módulo `math` permite acceder a las funciones de matemática de punto flotante:

```
>>> import math
>>> math.trunc(16.371)
16
>>> math.floor(16.371)
16
>>> math.ceil(16.371)
17
>>> math.cos(math.pi / 3)
0,494888338963
>>> math.log(1024, 2)
10.0
```

El módulo `datetime` permite manejar fechas y tiempos:

```
>>> from datetime import date
>>> hoy = date.today()
>>> hoy
datetime.date(2022, 3, 29)
```