Curso Básico de Python

2. El Intérprete de Python

Contenido

Python es un lenguaje interpretado, no compilado. Esto permite escribir un comando y ver inmediatamente su ejecución, sin requerir un proceso intermedio entre la escritura y la ejecución (compilación). Cuando Python se usa escribiendo comandos para ver el resultado en forma inmediata, se dice que se lo está usando en forma interactiva. El usuario emplea el lenguaje a través de un *intérprete de comandos*, una aplicación que recibe los comandos que escribimos, los ejecuta, y nos muestra el resultado.

El intérprete de comandos también permite ejecutar conjuntos de comandos guardados en un archivo (un programa o script), así como ejecutar funciones específicas de un conjunto contenido en un archivo (un módulo Python). En esta unidad nos familiarizamos con el intérprete de comandos, vemos las distintas formas de usar Python, y describimos someramente la estructura interna de los scripts y módulos de Python.

1 Introducción informal

Lecturas recomendadas:

1. Tutorial de Python, sección 3 Introducción informal a Python. El tutorial ofrece una guía para experimentar con Python en forma interactiva. Se puede hacer en línea, en una terminal, o en IDLE. Conviene ir escribiendo cada ejemplo, para ir memorizando los comandos y ver sus resultados.

Las siguientes secciones contienen una referencia rápida al contenido de esta unidad. Los ejercicios ayudan a fijar el conocimiento, pero también agregan información complementaria.

2 Tipos de datos principales

Numéricos, cadenas, lógicos, nulo.

- números enteros, tipo `int`; `type(24)`, `type(-33)` responden `int`.
- números en punto flotante, tipo `float`; `type(2.33)`, `type(-0.23)` responden `float`.
- booleanos, tipo `bool`; valores booleanos verdadero `True`, falso `False`; `bool(33 == 33)` responde `True`, `bool("abc" == "def")` responde `False`.
- cadenas de caracteres (string), tipo `str`; `type("abc+123")` responde `str`.
- tipo nulo `NoneType`, valor nulo `None`; `type(None)` responde `NoneType`.

Listas y diccionarios.

- lista, tipo `list`; `type([1, "abc", True, 23.1])` responde `list`.
- diccionario, tipo `dict`;
  `type( {"clave_1": "valor_1", "clave_2": "valor_2"} )` responde `dict`.
  `type( {"uno":1, 2:"dos", "bb":True} )` respone `dict`.

3 Operadores

- Aritméticos: `+` suma, `-` resta, `*` producto, `/` división, `//` división entera, `%` módulo.
- Comparación: `==` igual, `!=` distinto, `>` mayor, `<` menor, `>=` mayor o igual, `<=` menor o igual.
- Lógicos: `and` operador Y-lógico o AND, `or` operador O-lógico u OR, `not` operador NO-lógico o NOT.
- Caracteres especiales: `#` comentario hasta el final de la lína; `\n` nueva línea; `\<caracter>` escapa (no realiza) la interpretación como caracter especial.
- El operador de asignación `=` permite asociar un nombre con un valor, definiendo e inicializando o cambiando el valor de una variable:
  `nombre = "Antonio"; cant = 10; precision = 0.2`.
  No confundir el operador de asignación `=` con el operador comparativo de igualdad `==`.
  Es posible hacer asignaciones múltiples: `x, y, z = "Hugo", "Paco", "Luis"`.

El carácter `;` equivale a un fine de línea para un comando; permite escribir varios comandos en una línea. Escribir

```
x = "Antonio"; y = "Molonio"; print (x, y)
```

equivale a

```
x = "Antonio"
y = "Molonio"
print (x, y)
```

Usamos el `;` para ahorrar espacio en el instructivo, pero resulta más claro y se recomienda escribir los comandos en líneas separadas.

4 Cadenas de caracteres

- Una cadena de caracteres o string se define encerrando los caracteres entre comillas simples ' o dobles ": `"esto es una cadena"`, `'esto es otra'`. En `"una cadena es de tipo 'str'"` se toma la comilla simple como caracter. El comando `type('cadena')` responde `str`.
- El comando `print` mejora el formato de salida: si escribimos `"cadena de\ndos líneas"` el intérprete muestra los caracteres, si escribimos `print("cadena de\ndos líneas")` el comando `print` resuelve la codificación y muestra el cambio de línea.
- Las cadenas se unen por yuxtaposición (colocar una junto a otra) o con el operador `+` : `"Antonio" + " " + "Molonio"`.
- Una cadena puede multiplicarse con el operador `*` :
  `ss = "abracadabra"; ss*3; print(ss*3)`.
- En `print`, la coma separa los argumentos: `print(ss, ss, ss)`.

- Al definir una cadena pueden generarse líneas de continuación poniendo `\` al final de la línea. Esto NO agrega un cambio de línea, que debe indicarse con `\n`.
```
"esta es una cadena larga \
sin cambio de línea"
"esta es una cadena \n con cambio de linea"
```
- Pueden definirse cadenas usando comillas triples, ya sea con comillas simples o dobles.
- Las cadenas crudas (*raw strings*), no son interpretadas por `print`:
`ss2 = r"cadena\nde dos líneas"; print(ss2)`.
- La función `len(s)` devuelve la longitud de una cadena: `len(ss2)`.
- Las cadenas de caracteres son secuencias: cada caracter tiene una posición. Los caracteres se indizan desde 0 y hasta longitud - 1. Es posible extraer un carácter indicando su posición, `ss[i], 0 <= i < len(ss)`:
```
alf = "abcdefghijklmnopqrstuvwxyz"
alf[0]; alf[3]; alf[20]; alf[len(alf)-1]
```
- Recorte de cadenas: `ss[i,j], 0 <= i,j <= len(ss)` ; el último caracter incluido es `j-1`. Valores por defecto: `i=0, j=len(ss)`.
`alf[0:5]; alf[5:10]; alf[20:len(alf)-1]; alf[:5]; alf[20:]`
- Indices negativos: -1 es el último caracter, -2 el penúltimo, -3 el antepenúltimo, -len(ss) el primero, ya que len(ss) - len(ss) = 0 :
`alf[-1]; alf[-2]; alf[-len(alf)]`
- Las cadenas de caracteres son inmutables, no se pueden modificar: `ss[0] = 'x'` fracasa. Para cambiar una cadena es preciso crear otra : `"0123" + alf[5:]` .

5 Listas

- Una lista es una secuencia de elementos de cualquier tipo:
```
ls = [33, 24.4, 17+3j, "una cadena", True, False, None]  # crea la lista
print(ls)
```
- La función `len(ls)` da la longitud de la lista lst1. Los elementos de la lista están indizados entre 0 y len(lst1) – 1:
`len(ls); ls[0]; ls[3]; ls[1]; ls[len(ls)-1]   # largo, acceso a elementos`
- Los elementos de una lista se ubican mediante un índice, que indica su posición. Es posible cambiar un elemento de la lista, así como agregar y quitar elementos. La lista es una estructura de datos "mutable"; una cadena de caracteres es "inmutable".
```
ls = ls + ["a", 2.4]; print(ls)    # agrega elementos a la lista
ls[-1] = "Fin"; print(ls)          # cambia el último elemento
```
- La lista soporta recorte (*slicing*) con la notación `[p1:p2]` para incluir elementos de las posiciones `p1` a `p2-1` :
`ls[0:3]; ls[3:5]; ls[3:-1]; ls[len(ls)-2:]`
También es posible cambiar valores en un recorte:
`ls[3:5] = ["nuevo1", "nuevo2"]; ls`
- `help(list)` muestra las funciones disponibles para el manejo de listas. En IDLE, dar doble clic para expandir el texto.

6 Diccionarios

- Un diccionario es un conjunto de pares *clave:valor*, donde cada clave es única en ese diccionario. La clave puede ser cualquier tipo inmutable, como número o cadena de caracteres.
- Creación:

```
edades = {"Juan":23, "María":25, "Santiago":27}  # crea un diccionario
edades      # muestra el diccionario
list(edades)  # muestra las claves
dd_vacio = {}   # crea  un diccionario vacío
```

- Operaciones principales:

```
edades["María"]  # obtiene el valor (edad) para una clave (nombre)
edades["Sebastián"] = 33   # agrega una pareja clave:valor
edades["Juan"] = 30        # cambia el valor asociado a una clave
"Juan" in edades; "Arturo" in edades  # verifica presencia de una clave
```

7 Variables

Una *variable* es un nombre asociado a un valor. Esta asociación de establece en una sentencia de asignación, como las ya vistas:

```
nombre = "Antonio"
edad = 45
generos = ["teatro", "novela negra", "humor"]
```

Los nombres de variables pueden incluir letras mayúsculas y minúsculas, números y el caracter `_` , y deben comenzar con una letra. Son nombre válidos: `edad, var_1, var_2_str, RouterWiFi, NombreDeClase, nombre_de_variable`. Los nombres de variable que empiezan con `_` indican variables para uso interno de Python o los programas. Se recomienda usar letras minúsculas para los nombres de variables, reservando las mayúsculas para nombres de clases, en `NotacionDeCamello`, según se verá más adelante.

En Python, así como en otros lenguajes, existe un conjunto de palabras reservadas que no pueden usarse como nombres de variables, porque tienen un significado especial para el intérprete de comandos. La lista de palabras reservadas puede verse con el comando `help("keyword")`.

8 Nomenclatura.

Definición de algunos términos usados en programación.

- *valor.* Una de las unidades básicas de datos, como un número o una cadena, que un programa manipula.
- *tipo.* Una categoría de valores. Ejemplos: enteros (tipo int), números en punto flotante (tipo float), y cadenas (tipo str).
- *variable.* Un nombre que hace referencia a un valor.
- *palabra clave.* Una palabra reservada que es usada por el compilador para analizar un programa; no se pueden usar palabres clave como if, def, while como nombres de variables.
- *operador.* Un símbolo especial que representa un cálculo simple, como suma, multiplicación o concatenación de cadenas.
- *expresión.* Una combinación de variables, operadores y valores que representan un único valor resultante.
- *sentencia.* Una sección del código que representa un comando o acción. Son sentencias las asignaciones, print, type, help.
- *asignación.* Una sentencia que asigna un valor a una variable.

Fuente: Severance, Charles. *Python para Todos*. 2020.

9 Ejercicios

Los ejercicios permiten experimentar las posibilidades del lenguaje. Realizar las tareas indicadas, verificar los resultados del código propuesto, experimentar.

Números

1. Tipos de datos: numérico, cadena de caracteres o *string*, lista. En el intérprete de comandos (en línea, terminal, o IDLE), probar los siguientes comandos:

```
type(2); type(2.0); type(3.3333); type("cadena de caracteres")
float(2); int(3.33); float("2.44"); int(42); str(2.23); str(42)
float(2); int(3.33); float("2.44"); int(42); str(2.23); str(42)
type("esto es una cadena de caracteres")
type([]); type(["lista", "no", "vacía"])
```

Puedes copiar y pegar cada línea en el intérprete. Asegúrate de entender qué hace cada comando y su resultado.

2. Tipos booleanos. En el intérprete de comandos, probar los siguientes comandos:

```
bool(True); bool(False); type(True); type(False)
bool(0); bool(1); bool(33); bool(3.333)
bool(""); bool("una cadena no vacía")
bool([]); bool(["lista", "no", "vacía"])
```

3. La sentencia `isinstance(<valor>, <tipo>)` permite verificar el tipo de un dato. Probar con los tipos ya estudiados. Lossiguientes ejemplos responden `True`:

```
isinstance(2.3, float)
isinstance(2.3, (int, float))
ls = [1, "abc", True, 23.1]
isinstance(ls, list)
```

4. Operadores aritméticos. Probar los siguientes comandos:

```
10 + 4; 10 - 4; 10 * 4; 10 ** 4
10 / 4; 10 // 4; 10 % 3
```

5. Operador de asignación.

```
x = 5; print(x)
s = "una cadena de caracteres"; print(s)
x, y = [5, 7]; print(x, y)            # asignación  múltiple
si, no = True, False; print(si, no)   # asignación  múltiple
w, x, y, z = 5, 7.0, "uno", True; print(w, x, y, z)
```

6. Palabras reservadas.
   Visualizar la lista de palabras reservadas, usando el comando `help("keywords")`. Esas palabras no pueden usarse como nombres de variables.

7. Números complejos.
   El carácter `j` o `J` indica la parte imaginaria.

```
3 + 2j; -1.12 +  0.35J; complex(3 + 1j)   # definen números complejos
3 + j; 3 + J   # fracasan, toma j como variable; escribir 3 + 1j, 3 + 1J
c = 3 + 2j; c.real; c.imag
```

8. Biblioteca `math`.
   La biblioteca math contiene funciones matemáticas:

```
import math; dir(math)   # importa la biblioteca, muestra sus funciones
math.sin(math.pi/4); math.tan(math.pi/4)    # seno y tangente de pi/4
from math import *       # importa todas las funciones de math
sin(pi/4); tan(pi/4)     # ya no requiere indicar la biblioteca
abs(c) == sqrt(c.real**2 + c.imag**2)       # módulo de un complejo
_; print(_)   # la variable _ retiene el último resultado.
```

Acceso interactivo a documentación: `dir(math); help(math); help(math.sqrt)`.

Cadenas de caracteres

1. Comparar la salida de `"estas son \ndos líneas"` y `print("estas son \ndos líneas")`.

2. Probar las capacidades de interpretación de caracteres del comando `print` definiendo variables de cadena que contengan comillas, caracteres especiales como @ o *, y cambios de línea \n. Mostrar las variables 1) esribiendo su nombre en el intérprete, 2) usando el comando `print` ; observar las diferencias.

3. Probar la unión de cadenas por yuxtaposición (una junto a otra) y con el operador +. Multiplicar una cadena usando el operador *. Generar cadenas usando simultáneamente el operador + y el operador *.

4. Definir una variable de cadena usando continuación de líneas, el caracer \; verificar que no hay cambio de línea. Probar cambio de línea usando \n.

5. Definir una variable de varias líneas usando comillas simples o dobles. Digitar saltos de línea (sin escribir \n). Verificar que el comando `print` respeta el salto de línea introducido por digitación. El intérprete interactivo, por sí solo, no hace el cambio de línea digitado, pero muestra un \n incluido en la cadena.

6. Verificar comportamiento de cadenas crudas con `print`. Definir cadenas crudas con líneas de continuación \, cambio de línea \n, vocales acentuadas, caracteres especiales # @ *.

7. Definir una cadena. Determinar su longitud con `len()`. Extraer el primer caracter, el último caracter, el caracter del medio (¡sin contar los caracteres!).

8. Mostrar los primeros 3 caracteres y los últimos 3 caracteres de una cadena. Mostrar del caracter 3 hasta el final. Mostrar del caracter 0 hasta el antepenúltimo (dejar los 3 caracteres finales fuera).

9. Para una cadena `st` determinar que salida producen `st[3:], st[:5], st[:]`.

10. En `st5 = "01234567"` verificar las salidas de `st5[-1], st5[-len(st5)]`. Verificar las salidas de `st5[100], st5[-100]`. Verificar las salidas `st5[:100], st5[-100:]`.

11. Verificar que es posible incluir comillas simples o dobles en comentarios de varias líneas, en tanto las comillas que delimiten el comentario sean de otro tipo que las incluidas en el comentario. Ejemplo: `"""un 'string' es una cadena de caracteres"""`.

12. Para la variable `ss2 = r"cadena\nde dos líneas"` u otra similar, probar los siguientes comandos e interpretar los resultados:
```
type(ss2); help(ss2)
help(str)  # ayuda para string; en IDLE, dar doble clic para expandir
dir(ss2)   # muestra métodos disponibles para strings.
```

13. La yuxtaposición de cadenas sólo funciona con cadenas, no con variables o expresiones de cadena, por ejemplo si se incluyen funciones. Comparar:
```
"abra   ".strip() "cadabra"
"abra   ".strip() + "cadabra"
```
¿Qué hace la función `strip()`? Probar escribir `help(str.strip)`.

Listas

1. Crear una lista `lst` de 5 o 6 elementos, incluyendo elementos de diferentes tipos: int, float, str, boolean, None. Mostrar los elementos primero, segundo, último, penúltimo.

2. Cambiar el tercer elemento de `lst` por otro. Agregar un elemento 'Fin' al final de la lista; agregar un elemento 'Inicio' al principio de la lista.

3. Construir una lista `lst2` con los tres primeros elementos y los dos último de `lst`.

4. Analizar la salida de los comandos `lst2[0], lst2[-1], lst2[:100], lst2[-100:]`.

5. ¿Qué salida da el comando `lst2 + lst2[0:2]` ?

6. ¿Qué hace el comando `lst2*3` ?

7. Cambiar, en una sola operación, los elementos segundo y tercero de `lst2`.

8. Verificar que es posible anidar listas, i.e. poner como elemento de una lista otra lista.

9. Escribir o copiar un texto de 30 o 40 palabras, guardado en una variable `txt`. Generar una lista `pals` con las palabras de `txt`. Generar una lista `pals_ord` con las palabras de `pals` ordenadas. Mirar las ayudas en `help(str)`, `help(list)`, en particular las funciones `split` y `sort`.

10. Al aplicar la función `sort` a una lista, ¿es posible recuperar la ordenación anterior? ¿Por qué?

11. Constuir una lista `frutas` con 5 nombres de frutas. Aplicar a la lista `frutas` las funciones `append, count, extend, index, insert, pop, remove, reverse, sort`.

Diccionarios

1. Crear un diccionario de animales y sus géneros: el perro y el gato son mamíferos, la calandria es un ave, la yarará es un reptil. Realizar las siguientes operaciones:
   - Mostrar el diccionario animales.
   - Mostrar el género del gato.
   - Mostrar la lista de animales (lista de claves).
   - Agregar corvina, género pez.
   - Agregar sapo, género reptil, mostrar género del sapo; corregir, el sapo es un batracio, no un reptil; mostrar el género del sapo corregido.
