Ciclo I

Fundamentos de la programación

Semana 03

Entornos de desarrollo utilizados

Visual Studio Code

Enlace de descarga:

https://code.visualstudio.com/Download

Extensiones y temas para descargar:

- Python
- indent-rainbow
- Dark

Tipos de datos

En Python, todo valor que pueda ser asignado a una variable tiene asociado un tipo de dato, porque en Python todo es un objeto.

Donde los tipos de datos serían las clases y las variables serían las instancias de los tipos de datos.

En definitiva, **un tipo de dato establece qué valores puede tomar una variable y qué operaciones se pueden realizar sobre la misma**.

Tipos de datos básicos

Los *booleanos*, los *numéricos* (enteros, punto flotante y complejos) y las *cadenas de caracteres*.

Python también define otros tipos de datos, entre los que se encuentran:

- Secuencias: Los tipos *list* y *range*
- Mapas: El tipo *dict*
- Conjuntos: El tipo *set*
- Iteradores
- Clases
- Instancias
- Excepciones

Booleanos

El tipo booleano sólo puede tener dos valores: `True` (verdadero) y `False` (falso). Estos valores son especialmente importantes para las expresiones condicionales y los bucles

Clase	Tipo	Notas	Ejemplo
`bool`	Números	Valor booleano falso.	`False`
`bool`	Números	Valor booleano verdadero.	`True`

Convertir a booleanos

Para convertir a *tipos booleanos* debe usar la función **bool()** la cual esta integrada en el interprete Python.

Numéricos

Estos tipos de datos se crean mediante literales numéricos y se devuelven como resultados por operadores aritméticos y funciones aritméticas integradas. Los objetos numéricos son inmutables; Una vez creado su valor nunca cambia.

Por supuesto, los números de Python están fuertemente relacionados con los números matemáticos, pero están sujetos a las limitaciones de la representación numérica en las computadoras.

Python distingue entre enteros, números de punto flotante y números complejos:

Clase	Tipo	Notas	Ejemplo
`int`	Números	Número entero con precisión fija.	`42`
`long`	Números	Número entero en caso de overflow.	`42L` ó `456966786151987643L`
`float`	Números	Coma flotante de doble precisión.	`3.1415927`
`complex`	Números	Parte real y parte imaginaria *j*.	`(4.5 + 3j)`

Convertir a numéricos

Para convertir a tipos numéricos debe usar las siguientes funciones integradas en el interprete Python:

- La función **int()** devuelve un tipo de datos **número entero**.
- La función **long()** devuelve un tipo de datos **número entero long**.
- La función **float()** devuelve un tipo de datos **número entero float**.
- La función **complex()** devuelve un tipo de datos **número complejo**.

Cadenas de caracteres

Las cadenas de caracteres, son secuencias inmutables que contienen caracteres encerrado entre comillas.

Cadenas cortas

Son caracteres encerrado entre comillas simples (`'`) o dobles (`"`).

```
>>> 'Hola Mundo'
'Hola Mundo'
```

Cadenas largas

Son caracteres encerrados entre grupo comillas triples simples (`'''`) o dobles (`"""`), están son generalmente son referenciadas como *cadenas de triple comillas*.

```
>>> """Clase que representa una Persona"""
'Clase que representa una Persona'
>>> '''Clase que representa un Supervisor'''
'Clase que representa un Supervisor'
```

Convertir a cadenas de caracteres

Para convertir a *tipos cadenas de caracteres* debe usar la función **str()** la cual **esta integrada** en el interprete Python.

Ámbitos de variables y constantes

Variables

Es un nombre que se refiere a un objeto que reside en la memoria. El objeto puede ser de alguno de los tipos vistos (número o cadena de texto), o alguno de los otros tipos existentes en Python.

Cada variable debe tener un nombre único llamado **identificador**.

Eso es muy de ayuda pensar las variables como contenedores que contienen data el cual puede ser cambiado después a través de técnicas de programación.

Alcance de las variables

Las variables en Python son locales por defecto. Esto quiere decir que las variables definidas y utilizadas en el bloque de código de una **función**, sólo tienen existencia dentro de la misma, y no interfieren con otras variables del resto del código.

A su vez, las variables existentes fuera de una **función**, no son visibles dentro de la misma.

En caso de que sea conveniente o necesario, una variable local puede convertirse en una variable global declarándola explícitamente como tal con la sentencia **global**.

Ejemplos de variables

A continuación, se presentan algunos ejemplos del uso de *variables*:

Ejemplo de asignar valor a variable

A continuación, se creará un par de variables a modo de ejemplo. Una de **tipo cadenas de caracteres** y una de **tipo entero**:

```
>>> c = "Hola Mundo" # cadenas de caracteres
>>> type(c) # comprobar tipo de dato
<type 'str'>
>>> e = 23 # número entero
>>> type(e) # comprobar tipo de dato
<type 'int'>
```

Como puede ver en Python, a diferencia de muchos otros lenguajes, no se declara el tipo de la variable al crearla. En *Java*, por ejemplo, definir una variable seria así:

```
String c = "Hola Mundo";
int e = 23;
```

También nos ha servido el pequeño ejemplo para presentar los comentarios en linea en Python: cadenas de caracteres que comienzan con el carácter `#` y que Python ignora totalmente.

Ejemplo de asignar múltiples valores a a múltiples variables

A continuación, se creará múltiples variables **(entero, coma flotante, cadenas de caracteres)** asignando múltiples valores:

```
>>> a, b, c = 5, 3.2, "Hola"
>>> print(a)
5
>>> print(b)
3.2
>>> print(c)
'Hola'
```

Si usted quiere asignar el mismo valor a múltiples variables al mismo tiempo, usted puede hacer lo siguiente:

```
>>> x = y = z = True
>>> print(x)
True
>>> print(y)
True
>>> print(z)
True
```

El segundo programa asigna el mismo valor booleano a todas las tres variables `x`, `y`, `z`.

Constantes

Una constante es un tipo de variable la cual no puede ser cambiada. Eso es muy de ayuda pensar las constantes como contenedores que contienen información el cual no puede ser cambiado después.

En Python, las constantes son usualmente declaradas y asignadas en un módulo. Aquí, el módulo significa un nuevo archivo que contiene variables, funciones, etc; el cual es importada en el archivo principal.

Dentro del módulo, las constantes son escritas en letras MAYÚSCULAS y separadas las palabras con el carácter *underscore* `_`.

Constantes integradas

Un pequeño número de constantes vive en el espacio de nombres incorporado. Son las siguientes:

- None
- NotImplemented
- Ellipsis
- False
- True

Ejemplo de constantes

A continuación, se presentan algunos ejemplos del uso de constantes:

Ejemplo de constantes desde un módulo externo

Se crea un archivo llamado `constantes.py` con el siguiente contenido:

```
PORT_DB_SERVER = 3307
USER_DB_SERVER = "root"
PASSWORD_DB_SERVER = "123456"
DB_NAME = "nomina"
```

Crear un archivo llamado `main.py` con el siguiente contenido:

```
print("scp -v -P {0} {1}@{2}:/{3}/{4}/{4}.sql /srv/backup".format(
    str(constantes.PORT_DB_SERVER), constantes.USER_DB_SERVER,
    constantes.IP_DB_SERVER, constantes.USER_DB_SERVER,
    constantes.DB_NAME))
```

Luego ejecuta el programa de la siguiente forma:

```
python3 main.py
```

Cuando usted ejecuta el programa, la salida será:

```
scp -v -P 3307 root@127.0.0.1:/root/webapp/db.sql /srv/backup
```

En el programa anterior, existe un archivo de módulo `constantes.py`. Entonces en este se asignan los valores de constantes `IP_DB_SERVER`, `PORT_DB_SERVER`, `USER_DB_SERVER`, `PASSWORD_DB_SERVER` y `DB_NAME`. Ademas, existe el archivo de módulo `main.py` el cual importa el módulo `constantes`. Finalmente, se imprime una linea de conexión del comando `scp` de Linux usando la función integrada en la librería estándar Python llamada **format()**.

Palabras reservadas

Existen ciertas palabras que tienen significado especial para el intérprete de Python. Estas no pueden utilizarse para ningún otro fin (como ser nombrar valores) excepto para el que han sido creadas.

Puede verificar si una palabra esta reservada utilizando el módulo integrado `keyword`, de la siguiente forma:

```
import keyword
print(keyword.iskeyword('as'))
True
print(keyword.iskeyword('x'))
False
```

Para obtener una lista de todas las palabras reservadas.

```
import keyword
print(keyword.kwlist)
['and', 'as', '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',
'with', 'yield']
```

Reglas y convención de nombres

Algunas reglas y convenciones de nombres para las **variables** y **constantes**:

- Nunca use símbolos especiales como !, @, #, $, %, etc.
- El primer carácter no puede ser un número o dígito.
- Las constantes son colocadas dentro de módulos Python y significa que no puede ser cambiado.
- Los nombres de constante y variable debería tener la combinación de letras en minúsculas (de *a* a la *z*) o MAYÚSCULAS (de la *A* a la *Z*) o dígitos (del *0* al *9*) o un `underscore` (_).

  Por ejemplo:

  - snake_case
  - MACRO_CASE
  - camelCase
  - CapWords
- Los nombres que comienzan con **guión bajo (simple `_` o doble `__`)** se reservan para variables con significado especial
- No pueden usarse como identificadores, las palabras **reservadas**.

Operadores

De asignaciones

Existe en Python todo un grupo de operadores los cuales le permiten básicamente asignar un valor a una variable, usando el operador **«`=`»**. Con estos operadores pueden aplicar la técnica denominada asignación aumentada.

Operador =

El operador *igual a*, (`=`), es el más simple de todos y asigna a la variable del lado izquierdo cualquier variable o resultado del lado derecho.

Operador +=

El operador `+=` suma a la variable del lado izquierdo el valor del lado derecho.

```
>>> r = 5
>>> r += 10
>>> r
15
```

En el ejemplo anterior si la variable «`r`» es igual a `5` y `r += 10`, entonces la variable «`r`» sera igual a `15`. Su equivalente seria el siguiente:

```
>>> r = 5
>>> r = r + 10
>>> r
15
```

Operador -=

El operador `-=` resta a la variable del lado izquierdo el valor del lado derecho.

```
>>> r = 5
>>> r -= 10
>>> r
-5
```

En el ejemplo anterior si la variable «`r`» es igual a `5` y `r -= 10`, entonces la variable «`r`» sera igual a `-5`. Su equivalente seria el siguiente:

```
>>> r = 5
>>> r = r - 10
>>> r
-5
```

Operador *=

El operador `*=` multiplica a la variable del lado izquierdo el valor del lado derecho.

```
>>> r = 5
>>> r *= 10
>>> r
50
```

En el ejemplo anterior si la variable «`r`» es igual a `5` y `r *= 10`, entonces la variable «`r`» sera igual a `50`. Su equivalente seria el siguiente:

```
>>> r = 5
>>> r = r * 10
>>> r
50
```

Operador /=

El operador `/=` divide a la variable del lado izquierdo el valor del lado derecho.

```
>>> r = 5
>>> r /= 10
>>> r
0
```

En el ejemplo anterior si la variable «`r`» es igual a `5` y `r /= 10`, entonces la variable «`r`» sera igual a `0`. Su equivalente seria el siguiente:

```
>>> r = 5
>>> r = r / 10
>>> r
0
```

Operador `**=`

El operador `**=` calcula el exponente a la variable del lado izquierdo el valor del lado derecho.

```
>>> r = 5
>>> r **= 10
>>> r
9765625
```

En el ejemplo anterior si la variable «`r`» es igual a `5` y `r **= 10`, entonces la variable «`r`» sera igual a `9765625`. Su equivalente seria el siguiente:

```
>>> r = 5
>>> r = r ** 10
>>> r
9765625
```

Operador //=

El operador `//=` calcula la división entera a la variable del lado izquierdo el valor del lado derecho.

```
>>> r = 5
>>> r //= 10
>>> r
0
```

En el ejemplo anterior si la variable «`r`» es igual a `5` y `r //= 10`, entonces la variable «`r`» sera igual a `0`. Su equivalente seria el siguiente:

```
>>> r = 5
>>> r = r // 10
>>> r
0
```

Operador %=

El operador `%=` devuelve el resto de la división a la variable del lado izquierdo el valor del lado derecho.

```
>>> r = 5
>>> r %= 10
>>> r
5
```

En el ejemplo anterior si la variable «`r`» es igual a `5` y `r %= 10`, entonces la variable «`r`» sera igual a `5`. Su equivalente seria el siguiente:

```
>>> r = 5
>>> r = r % 10
>>> r
5
```

Asignación aumentada

Es frecuente que una variable tenga que ser definida de nuevo en función de sí misma. Normalmente usted escribir la siguiente sintaxis:

```
>>> contador = contador + 1
```

El código anterior, se puede abreviar a su equivalente, usando la asignación aumentada, de la siguiente manera:

```
>>> contador += 1
```

El código anterior, no sólo es más corto de escribir, sino también más eficiente en tiempo de ejecución.

Aritméticos

- Suma

  ```
  >>> 3 + 2
  5
  ```

- Resta

  ```
  >>> 4 - 7
  -3
  ```

- Negación

  ```
  >>> -7
  -7
  ```

- Multiplicación

  ```
  >>> 2 * 6
  12
  ```

- Exponente

  ```
  >>> 2 ** 6
  64
  ```

- División

  ```
  >>> 3.5 / 2
  1.75
  ```

- División entera

```
>>> 3.5 // 22
1.0
```

- Módulo

```
>>> 7 % 2
1
```

Relacionales

Operador ==

```
>>> 5 == 3
False
>>> 5 == 5
True
>>> "Plone" == 5
False
>>> "Plone" == "Plone"
True
>>> type("Plone") == str
True
```

Operador !=

```
>>> 5 != 3
True
>>> "Plone" != 5
True
>>> "Plone" != False
True
```

Operador <

```
>>> 5 < 3
False
```

Operador >

```
>>> 5 > 3
True
```

Operador <=

```
>>> 5 <= 3
False
```

Operador >=

```
>>> 5 >= 3
True
```

Métodos de entrada y salida

Para pedir información al usuario, debe utilizar las funciones integradas en el interprete del lenguaje, así como los argumentos de línea de comandos.

Entrada

La función **input()** siempre devuelve un valor numérico:

```
>>> edad = input('Ana: ¿Que edad tiene usted?: ')
Ana: ¿Que edad tiene usted?: 38
>>> print(edad)
38
```

Salida

En la sentencia `print` se pueden usar el formato de impresión alternando las cadenas de caracteres y variables:

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> valor = 2**0.5
>>> print("el resultado de", tipo_calculo, "es:", valor)
el resultado de raíz cuadrada de dos es: 1.41421356237
```