Introducción

Una tarea común es desarrollar programas que no solo puedan mostrar información en una pantalla o una consola, sino que también reciban información de los usuarios o incluso de otros programas. En este módulo, construirás tu primer programa en Python para aprender a manejar la entrada y la salida en la consola. También aprenderás conceptos de programación de Python como variables y conversión entre tipos de datos.

Escenario: Un programador junior creando sus primeras aplicaciones

En este escenario, eres un joven cadete espacial en una nave espacial en algún momento en el futuro. Tu misión es encontrar y explorar nuevos sistemas solares. Estás capacitado como ingenierx y estás buscando desarrollar una serie de programas de servicios públicos para el personal más alto a bordo.

¿Qué aprenderás?

- Utilizar funciones para administrar la entrada y la salida a la consola.
- Crear variables para almacenar datos.
- Distinguir entre tipos de datos.
- Utilizar la conversión de tipos para convertir entre tipos de datos.

¿Cuál es el objetivo principal?

Utiliza elementos básicos de programación para crear tus primeros programas Python.

Trabajar con salidas

Cuando comienzas a aprender cualquier lenguaje de programación, primero pruebas algunas declaraciones de código en un bucle de lectura-evaluación-impresión (REPL). Sin embargo, pronto querrás pasar a un desarrollo serio. Eso significa aprender más sobre el idioma, cómo estructurar un programa y más.

Un programa Python

Para crear un programa en Python, debes almacenarlo en un archivo. El archivo debe tener la extensión .py.

La idea de un programa es hacer algo, llevar a cabo una tarea. Para que el programa haga algo, debe agregar instrucciones de código que realicen instrucciones. Una instrucción podría imprimir algún texto o calcular algo, por ejemplo. Un programa de ejemplo puede tener un aspecto similar al siguiente:

```
# program.py
```

```
sum = 1 + 2
print(sum)
```

Ejecutar un programa

Supongamos que has creado un programa que consta de instrucciones. Para ejecutarlo hay que invocar el programa ejecutable de Python, seguido del nombre del programa. Aquí hay un ejemplo de tal invocación:

```
python3 program.py
```

La ejecución de un programa de este tipo mostraría el siguiente resultado en la consola:

```
3
```

La función print()

Una de las primeras cosas que es probable que hagas es imprimir en una consola. Una consola es una aplicación de línea de comandos que te permite interactuar con el sistema operativo. En la consola, puedes ejecutar comandos y programas. También puedes ingresar información y mostrar información como texto en la pantalla.

Para escribir información en la consola, puedes utilizar la función e implementarla como función principal. Debido a que es una función central, tendrás acceso a ella si Python está instalado. Para usarla dale un argumento: `print()print()`

```
print('Hola desde la consola')
```

In [ ]:
```
# Impresión en pantalla print('Hola desde la consola')
```

Observa cómo el comando anterior invoca mediante paréntesis. Así es como invocas una función. Si usaras corchetes () en lugar de paréntesis, no funcionaría:print[]

Variables

Para avanzar en la programación, debes comprender que estás operando con datos. Como tu programa está trabajando en datos, es posible que deba recordar un cierto valor a lo largo de la ejecución del programa. Para eso, se utilizan variables.

En el ejemplo siguiente se realiza un cálculo y se almacena en variables:

```
sum = 1 + 2 # 3
product = sum * 2
print(product)
```

In [ ]:
```
# Tu turno, prueba el fragmento de código anterior
```

Tipos de datos

Una variable asume un tipo de datos. En el programa anterior, obtiene el tipo . Pero hay muchos más tipos de datos. Aquí hay algunos que es probable que encuentre: `sum` `int`

Tipo	Descripción	Ejemplo
Tipo numérico	Número, con o sin decimales	`int, float, complex, no = 3`
Tipo de texto	Cadena de caracteres	`str = "a literal string"`
Tipo booleano	Booleano	`continue = True`

Hay tipos más complejos, pero comencemos con estos.

Aquí hay un fragmento de código que muestra algunos de los tipos anteriores:

```
planetas_en_el_sistema_solar = 8 # int, plutón era considerado un
planeta pero ya es muy pequeño
distancia_a_alfa_centauri = 4.367 # float, años luz
puede_despegar = True
transbordador_que_aterrizo_en_la_luna = "Apollo 11" #string
```

¿Cómo sabes qué tipo tiene algo? Si ve los datos asignados a la variable como se muestra en el código siguiente, puede detectarlos:

```
distancia_a_alfa_centauri = 4.367 # Parece un decimal flotante
```

La otra forma es usar la función:type()

```
type(distancia_a_alfa_centauri)
```

In [1]:
```
# Declaramos la variable
distancia_a_alfa_centauri = 4.367

# Descubrimos su tipo de dato
type(distancia_a_alfa_centauri)
```

Out[1]:
```
float
```

Operadores

Los operadores le permiten realizar cálculos sobre variables y sus valores. La idea general es que tienes un lado izquierdo y un lado derecho y un operador en el medio:

```
<left side> <operator> <right side>
```

Así es como se vería un ejemplo real del código de marcador de posición anterior:

```
left_side = 10
right_side = 5
```

```
left_side / right_side # 2
```

En este ejemplo se utiliza una barra diagonal (/) para dividir el valor por el valor.
`left_side` `right_side`

Hay muchos más operadores.

Python utiliza dos tipos de operadores: aritmética y asignación.

Operadores aritméticos

Con los operadores aritméticos, se realizan cálculos como suma, resta, división y multiplicación. Aquí hay un subconjunto de operadores aritméticos que puede usar:

Tipo	Descripción	Ejemplo
+	Operador de adición que suma dos valores juntos	`1 + 1` .
-	Operador de resta que quita el valor del lado derecho del lado izquierdo	`1 - 2` .
/	Operador de división que divide el lado izquierdo tantas veces como especifique el lado derecho	`10 / 2` .
*	Operador de multiplicación	`2 * 2` .

Operadores de asignación

Los operadores de asignación se utilizan para asignar valores a una variable a lo largo del ciclo de vida de la variable. Estos son algunos operadores de asignación que es probable que encuentres a medida que aprendes a crear programas:

Operador	Ejemplo
=	`x = 2` x ahora contiene 2.
+=	`x += 2` x incrementado en 2. Si antes contenía 2, ahora tiene un valor de 4.
-=	`x -= 2` x decrementado por 2. Si antes contenía 2, ahora tiene un valor de 0.
/=	`x /= 2` x dividido por 2. Si antes contenía 2, ahora tiene un valor de 1.
*=	`x *= 2` x multiplicado por 2. Si antes contenía 2, ahora tiene un valor de 4.

Fechas

Cuando estás creando programas, es probable que interactúes con las fechas. Una fecha en un programa generalmente significa tanto la fecha del calendario como la hora.

Una fecha se puede utilizar en varias aplicaciones, como estos ejemplos:

- **Archivo de copia de seguridad.** Usar una fecha como parte del nombre de un archivo de copia de seguridad es una buena manera de indicar cuándo se realizó una copia de seguridad y cuándo debe realizarse nuevamente.
- **Condición.** Es posible que desee llevar una lógica específica cuando hay una fecha determinada.
- **Métrica.** Las fechas se utilizan para comprobar el rendimiento del código para, por ejemplo, medir el tiempo que se tarda en ejecutar una función.

Para trabajar con una fecha, debe importar el módulo: `date`

```
from datetime import date
```

A continuación, puede invocar las funciones con las que desea trabajar. Para obtener la fecha de hoy, puede llamar a la función: `today()`

```
date.today()
```

Para mostrar la fecha en la consola, puede usar la función. La función toma muchos tipos de datos como entrada. Así es como puedes mostrar la fecha de hoy: `print()`

```
print(date.today())
```

In [1]:
```
# Importamos la biblioteca
from datetime import date

# Obtenemos la fecha de hoy
date.today()

# Mostramos la fecha en la consola
print(date.today())
```

```
2022-01-31
```

Conversión de tipos de datos

Quieres usar una fecha con un mensaje. Ese algo suele ser un problema. Si, por ejemplo, desea mostrar la fecha de hoy en la consola, es posible que tenga un problema:

```
print("Today's date is: " + date.today())
```

Lo que obtienes es un error:

```
---------------------------------------------------------------------
---------
TypeError                                 Traceback (most recent
call last)
<ipython-input-2-e74c8796a0b9> in <module>
----> 1 print("Today's date is: " + date.today())
```

```
TypeError: can only concatenate str (not "datetime.date") to str
```

La última fila del mensaje le indica cuál es el problema. Está intentando usar el operador `+` y combinar dos tipos de datos diferentes, una cadena y una fecha.

Para que este código funcione, debe convertir la fecha en una cadena. Para lograr tal conversión mediante el uso de la función de utilidad: `str()`

```
print("Today's date is: " + str(date.today()))
```

In [ ]:
```
# Tu turno ejecuta el siguiente comando: print("Today's date is: " + str(date.to
```

Recopilar información

Hasta ahora, has aprendido varias construcciones del lenguaje de programación Python. También has escrito un par de programas. Sin embargo, los programas operan con datos, y esos datos provienen de alguna parte. En esta unidad, analizará más de cerca cómo puede recopilar la entrada tanto de la línea de comandos como de la entrada del usuario.

Entrada del usuario

Puede codificarlo para que el programa le diga al usuario que ingrese información. Guarde los datos introducidos en el programa y, a continuación, actúe en consecuencia.

Para capturar información del usuario, utilice la función. Aquí hay un ejemplo: `input()`

```
print("Bienvenido al programa de bienvenida")
name = input("Introduzca su nombre ")
print("Saludos: " + name)
```

In [ ]:
```
# Escribe tu código aquí
```

La ejecución del programa mediante el uso da el siguiente resultado `python input.py` (Si lo haces por consola) :

```
Bienvenido al programa de bienvenida
Enter your name Fernanda
Saludos: Fernanda
```