Introducción a las ciencias de la computación
y programación en Python

Control de flujo e iteraciones

Rodrigo Chang

Banco de Guatemala

Rodrigo Chang <rrcp@banguat.gob.gt>

Abstract

“I taught myself how to program computers when I was a kid, bought my first computer when I was 10, and sold my first commercial program when I was 12.”
— Elon Musk

Veremos como obtener una entrada del usuario y generar una salida a través de la línea de comandos. Hablaremos de las condicionales y de las estructuras de iteración.

Cadenas de texto (introducción)

- Son objetos de tipo no escalar que permiten manejar números, letras, espacios y caracteres especiales.
- Se encierran en comillas dobles o simples.

```
hello = "Hola mundo"
hi = 'Hola amigos del PES'
```

- Podemos concatenar cadenas de texto con el operador +:

```
nombre = "Antonio"
saludo = hello + " " + nombre
```

- Podemos efectuar otras operaciones:

```
molesto = hello + " " + nombre*3
```

Salida en consola

- Utilizamos la función `print`.

```
print("Hola mundo")
```

- Podemos enviar varios argumentos a la función para imprimirlos todos. Podemos separar los elementos con el caracter especificado en el argumento `sep=' '`.

```
pi=3.1415
print(pi, 2, "amigos")
print(1, 'Python Tricks', 'Dan Bader', sep=',')
```

Combinar variables en el texto

- Será útil en muchas ocasiones, combinar en un mensaje de salida una o más variables.
- Para esto, utilizamos el operador `%`.

```
pi=3.1415
print("Hola %s, Pi=%0.4f" % ('Rodrigo', pi))
```

- También es posible utilizando diccionarios:

```
print("Pi=%(pi)s" % {'pi' : pi})
```

- Otras especificaciones posibles:
  - `%d` para números enteros.
  - `%s` para cadenas de texto.
  - `%f` para números de punto flotante.

https://realpython.com/python-string-formatting/

Entrada vía línea de comandos

- Utilizamos el la función `input("")`.
- Devuelve el valor escrito por el usuario a una variable:

```
nombre = input("Ingresa tu nombre: ")
print("Hola " + nombre)
```

- `input` devuelve objetos de tipo `str`. Al trabajar con números se deben convertir los tipos:

```
r = float(input("Ingresa el radio: "))
area = 3.14159*(r**2)
```

Condicionales

- Podemos decidir si ejecutar un grupo de instrucciones con la siguiente estructura. <condition> debe ser booleana.

```
if <condition>:
    <expression>
    <expression>
    ...
```

- La cláusula `else` indica las instrucciones a evaluar si la condicion es falsa.

```
if <condition>:
    <expression>
    ...
else:
    <expression>
    ...
```

La indentación es IMPORTANTE, esta es la forma en que Python estructura los bloques.

Condicionales

- La cláusula `elif` permite añadir condiciones adicionales.

```
if <condition>:
    <expression>
    ...
elif <condition>:
    <expression>
    ...
else:
    <expression>
    ...
```

Bloques de código

```
x = float(input("Enter a number for x: "))
y = float(input("Enter a number for y: "))
if x == y:
    print("x and y are equal")
    if y != 0:
        print("therefore, x / y is", x/y)
elif x < y:
    print("x is smaller")
elif x > y:
    print("y is smaller")
print("thanks!")
```

Ciclo `while`

- Cuando queremos repetir una expresión hasta que se cumpla una condición.

```
while <condition>:
    <expression>
    <expression>
    ...
```

- Nuevamente, <condition> debe ser de tipo booleano.
- Si la condición es `True`, el bloque se ejecuta.

Ciclo `while`

- Veamos un ejemplo con ciclo `while`.

```
ans = input("Programar es divertido. ")
while ans != "Es genial":
    ans = input("Puedes pensarlo nuevamente?: ")
print("Excelente, ahora eres programador@.")
```

Ciclos `for`

- Cuando queremos iterar sobre números de una secuencia.

```
# more complicated with while loop
n = 0
while n < 5:
    print(n)
    n = n+1

# shortcut with for loop
for n in range(5):
    print(n)
```

Ciclos `for`

- En cada iteración, <variable> toma un valor.

```
for <variable> in range(<some_num>):
    <expression>
    ...
```

- La primera vez, toma el valor más pequeño, `0`.
- La siguiente vez, toma el valor anterior + 1.

Función `range`

- Los argumentos `start` y `step` son opcionales.
- Tienen valores por defecto: `start = 0` y `step = 1`.
- El ciclo llega hasta `stop - 1`.

```
# Sumamos números de 7 a 9
mysum = 0
for i in range(7, 10):
    mysum += i
    print(mysum)

# Sumamos números impares
mysum = 0
for i in range(5, 11, 2):
    mysum += i
    print(mysum)
```

Note el uso del operador abreviado +=.

Sentencia `break`

- **Fuerza** la salida del ciclo actual.
- Instrucciones posteriores son ignoradas.
- Solamente actúa en el ciclo más interno.

```
while <condition_1>:
    while <condition_2>:
        <expression_A>
        break
        <expression_B>
    <expression_C>
```

Sentencia `break`

```
mysum= 0
for i in range(5, 11, 2):
    mysum += i
    if mysum == 5:
        break
        mysum += 1
print(mysum)
```

¿Qué pasa en este programa?

Ciclos `for` vs. `while`

`for`

- Se conoce el número de iteraciones.
- Admite `break`.
- Utiliza un contador o lista.
- Se puede reescribir un ciclo `for` utilizando `while`.

`for`

- Número de iteraciones no acotado.
- Admite `break`.
- Puede utilizar contador pero debe inicializarlo y actualizarlo en el ciclo.
- Puede no ser posible reescribir un ciclo `while` utilizando `for`.