Para estudiantes de Educación Básica y Media
UNIVERSIDAD DE CHILE

Clase 2: Ciclos y funciones en Python

FM849 - Programación Científica para Proyectos de Inteligencia Artificial (IA)

5 de enero de 2026

Condicionales y ciclos

Python tiene condicionales lógicos, ciclos y otros operadores de flujo para controlar el orden de ejecución de instrucciones.

Tipo de operador	Uso	Operadores
Condicionales	Ejecutar código según condiciones	`if`, `elif`, `else`
Ciclos (*loops*)	Recorrer conjuntos de datos	`for`, `while`
Controladores de ciclo	Funcionalidades útiles	`continue`, `break`, `pass`

Vamos a ver más ejemplos en código: 🔗 Material complementario.

Ciclos: `for`

El ciclo `for` se utiliza para recorrer elementos en un iterable (p. ej., tuplas, listas, vectores, etc.).

Sintaxis general

```
>>> for variable in iterable:
...     # Código a ejecutar en cada iteración.
```

Ejemplo

```
for i in range(5):
    print(i)
```

Salida

```
0
1
2
3
4
```

Ciclos: `while`

El ciclo `while` ejecuta un bloque de código mientras una condición sea cierta.

Sintaxis general

```
>>> while condition:
...     # Código a ejecutar mientras condition == True.
```

Ejemplo

```
k = 0
while k < 5:
    print(i)
    k += 1 # k = k + 1.
```

Salida

```
0
1
2
3
4
```

Control de ciclo: `continue`

La instrucción `continue` genera un salto a la siguiente iteración del ciclo, ignorando el código que sigue.

Sintaxis general

```
>>> while condition1:
...     if condition2:
...         continue
...     # Código no ejecutado si condition2 == True.
```

Ejemplo

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

Salida

```
0
1
3
4
```

Control de ciclo: `break`

La instrucción `break` termina completamente el ciclo.

Sintaxis general

```
>>> while condition1:
...     if condition2:
...         break
... # Si condition2 == True, break se activa y continuamos acá.
```

Ejemplo

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

Salida

```
0
1
2
```

Control de ciclo: pass

La instrucción `pass` es un comando que indica no hacer nada.

Ejemplo (supondremos que `x` ya está definido)

```
>>> if x < 0:
...     print("¡Negativo!")
... elif x == 0:
...     # TODO: agregar el caso x == 0.
...     pass
... else:
...     print("¡Positivo!")
```

Funciones

Una función es un bloque de código "reutilizable" (cuidado con este concepto, porque existen las funciones anónimas). Algunas características:

- Tiene un objetivo especifico.
- Usualmente, recibe parámetros, también conocidos como *inputs* en la jerga de programación.
- Comúnmente, entregará un resultado.

Ejemplo

```
>>> # Función que retorna la suma de dos números.
>>> def my_function(x, y):
...     return x + y
```

Funciones

- Las funciones se definen usando la palabra clave `def`.
- Luego de `def`, se escribe el nombre de la función y, entre paréntesis, los parámetros que ésta recibe.
- En las siguientes líneas de código se escriben todas las funcionalidades de la función.
- Cuando se alcanza una línea que tenga el comando `return`, la función devuelve el valor después de `return` en el contexto en el cual fue llamada.

Sintaxis general

```
def nombre_funcion(parametros):
    # Cuerpo de la función.
```

Ejemplo

```
def suma(a, b):
    return a + b
resultado = suma(3, 5)
print(resultado)
```

Funciones

- Las funciones ayudan a organizar y reutilizar código en Python.
- Como regla general, si sabemos que una parte del código será usada múltiples veces, podemos crear una función que la abstraiga.

Sin funciones

```
>>> import numpy as np
>>> l = [2, 3, 4, 5, 2]
>>> max_valor = -np.inf
>>> for i in range(len(l)):
...     if l[i] > max_valor:
...         max_valor = l[i]
>>> l = [255, 313, 42, 53, 20]
>>> max_valor = -np.inf
>>> # Se repite lo mismo aquí...
```

Con funciones

```
>>> import numpy as np
>>> def calc_max(l):
...     max_valor = -np.inf
...     for i in range(len(l)):
...         if l[i] > max_valor:
...             max_valor = l[i]
...     return max_valor
>>> l = [2, 3, 4, 5, 2]
>>> max_valor = calc_max(l)
```

Clases

- Las clases proporcionan un medio para agrupar datos y funcionalidades.
- Cuando creamos una clase nueva, podemos generar objetos con nuevas características.
- Las clases tienen dos elementos fundamentales: atributos y métodos.

```
>>> class Complex:
...     def __init__(self, real_part, imag_part):
...         self.r = real_part
...         self.i = imag_part
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

Referencias

GeeksforGeeks. Loops in Python, 2024. URL https://www.geeksforgeeks.org/python/loops-in-python/. Accessed: 2026-01-05.

Wes McKinney. *Python for Data Analysis*. O'Reilly Media, 3 edition, 2022.