Material Imprimible

Curso de Python

Módulo 3

Python If ... Else

Condiciones de Python y sentencias If

Python admite las condiciones lógicas habituales de las matemáticas:

- Es igual a: `a == b`
- No es igual a: `a != b`
- Menos de: `a < b`
- Menos o igual que: `a <= b`
- Mayor que: `a > b`
- Mayor o igual que: a `a >= b`

Estas condiciones se pueden utilizar de varias maneras, más comúnmente en declaraciones "if " y “loops”.

Una "declaración if" se escribe mediante la palabra clave `if.`

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Sangría

Python se basa en la sangría (espacio en blanco al principio de una línea) para definir el ámbito en el código. Otros lenguajes de programación a menudo utilizan llaves para este propósito.

```
a = 33
b = 200
```

```
if b > a:
print("b is greater than a") # obtendrá un error
```

Elif

La palabra clave elif es una forma de decir pythons "si las condiciones anteriores no eran verdaderas, entonces pruebe esta condición".

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

Else

La palabra clave else detecta cualquier cosa que no sea detectada por las condiciones anteriores.

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

También puede tener un sin :elif

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Short Hand If

Si solo tiene una instrucción que ejecutar, puede colocarla en la misma línea que la instrucción if.

```
a = 200
b = 33

if a > b: print("a is greater than b")
```

Short Hand If ... Else

Si solo tiene una instrucción que ejecutar, una para if y otra para ELSE, puede ponerla todo en la misma línea:

```
a = 2
b = 330

print("A") if a > b else print("B")
```

Esta técnica se conoce como **Operadores ternarios** o **Expresiones condicionales.**

También puede tener varias sentencias else en la misma línea:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

And

La palabra clave and es un operador lógico y se utiliza para combinar instrucciones condicionales:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

Or

La palabra clave or es un operador lógico y se utiliza para combinar instrucciones condicionales:

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

Anidado Si

Puede tener instrucciones if dentro de instrucciones if, esto se denomina instrucciones if *anidadas*.

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
        print("and also above 20!")
  else:
        print("but not above 20.")
```

La Declaración de pass

if instrucciones if no pueden estar vacías, pero si por alguna razón tiene una instrucción sin contenido, coloque la instrucción pass para evitar recibir un error.

```
a = 33
b = 200

if b > a:
  pass
# tener una declaración if vacía como esta, generaría un error sin la declaración pass
```

Bucles de Python

Python tiene dos comandos de bucle primitivos:

- While los bucles
- for bucles

El bucle de tiempo

Con el bucle while podemos ejecutar un conjunto de instrucciones siempre que una condición sea verdadera.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Nota: recuerde incrementar i, o de lo contrario el bucle continuará para siempre.

El bucle while requiere que las variables relevantes estén listas, en este ejemplo necesitamos definir una variable de indexación, i, que establecemos en 1.

La Declaración de ruptura (Break)

Con la instrucción break podemos detener el bucle incluso si la condición while es verdadera:

```
i = 1
while i < 6:
  print(i)
  if (i == 3):
        break
  i += 1
```

La Declaración continúe

Con la instrucción continue podemos detener la iteración actual y continuar con la siguiente:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
        continue
  print(i)
```

```
# Faltara el numero 3 en el resultado
```

La declaración de else

Con la instrucción else podemos ejecutar un bloque de código una vez cuando la condición ya no es verdadera:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i ya no es menor que 6")
```

Python para bucles

Un bucle for se utiliza para recorrer en iteración una secuencia (es decir, una lista, una tupla, un diccionario, un conjunto o una cadena).

Esto se parece menos a la palabra clave for en otros lenguajes de programación y funciona más como un método de iterador como se encuentra en otros lenguajes de programación orientados a objetos.

Con el bucle for podemos ejecutar un conjunto de instrucciones, una vez para cada elemento de una lista, tupla, set, etc.

```
frutas = ["manzana", "banana", "cereza"]
for x in frutas:
  print(x)
```

El bucle for no requiere una variable de indexación para establecer de antemano.

Bucle a través de una cadena

Incluso las cadenas son objetos iterables, contienen una secuencia de caracteres:

```
for x in "banana":
  print(x)
```

La Declaración de ruptura

Con la instrucción break podemos detener el bucle antes de que haya atravesado todos los elementos:

```
frutas = ["manzana", "banana", "cereza"]
for x in frutas:
  if x == "banana":
        break
  print(x)
```

La Declaración continua

Con la instrucción continue podemos detener la iteración actual del bucle y continuar con la siguiente:

```
frutas = ["manzana", "banana", "cereza"]
for x in frutas:
  if x == "banana":
        continue
  print(x)
```

La función range()

Para recorrer un conjunto de código un número especificado de veces, podemos usar la función range(),

La función range() devuelve una secuencia de números, a partir de 0 de forma predeterminada, e incrementa en 1 (de forma predeterminada) y termina en un número especificado.

```
for x in range(6):
  print(x)
```

Tenga en cuenta que range(6) no es los valores de 0 a 6, sino los valores 0 a 5.

El valor predeterminado de la función range() es 0 como valor inicial, sin embargo, es posible especificar el valor inicial añadiendo un parámetro: range(2, 6),lo que significa valores de 2 a 6 (pero sin incluir 6):

```
for x in range(2, 6):
```

```
  print(x)
```

La función range() por defecto incrementa la secuencia en 1, sin embargo, es posible especificar el valor de incremento añadiendo un tercer parámetro: range(2, 30, **3**):

```
for x in range(2, 30, 3):
  print(x)
```

Else in For Loop

La palabra clave else de un bucle especifica un bloque de código que se ejecutará cuando finalice el bucle: for

```
for x in range(6):
  print(x)
else:
  print("Finalmente Terminado!")
```

Bucles anidados

Un bucle anidado es un bucle dentro de un bucle.

El "bucle interno" se ejecutará una vez para cada iteración del "bucle externo":

```
adj = ["rojo", "grande", "sabrosa"]
frutas = ["manzana", "banana", "cereza"]

for x in adj:
  for y in frutas:
    print(x, y)
```

La Declaración de pass

for Los bucles no pueden estar vacíos, pero si por alguna razón tiene un bucle sin contenido, coloque la instrucción para evitar obtener un error.for pass

```
for x in [0, 1, 2]:
  pass
# tener un ciclo for vacío como este, generaría un error sin la instrucción pass
```

Funciones de Python

Una función es un bloque de código que solo se ejecuta cuando se llama.

Puede pasar datos, conocidos como parámetros, a una función.

Una función puede devolver datos como resultado.

Creación de una función

En Python, una función se define mediante la palabra clave def:

```
def my_function():
  print("Hello from a function")
```

Llamar a una función

Para llamar a una función, utilice el nombre de la función seguido de paréntesis:

```
def my_function():
  print("Hello from a function")
 my_function()
```

Argumentos

La información se puede pasar a funciones como argumentos.

Los argumentos se especifican después del nombre de la función, dentro de los paréntesis.

Puede agregar tantos argumentos como desee, simplemente sepárelos con una coma.

En el ejemplo siguiente se tiene una función con un argumento (fname). Cuando se llama a la función, pasamos un nombre, que se utiliza dentro de la función para imprimir el nombre completo:

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Los argumentos a menudo se acortan a *args* en la documentación de Python.

¿Parámetros o argumentos?

Los términos *parámetro* y *argumento* se pueden utilizar para la misma cosa: información que se pasa a una función.

Desde la perspectiva de una función:

Un parámetro es la variable que aparece entre paréntesis en la definición de función.

Un argumento es el valor que se envía a la función cuando se llama.

Número de argumentos

De forma predeterminada, se debe llamar a una función con el número correcto de argumentos. Lo que significa que si la función espera 2 argumentos, debe llamar a la función con 2 argumentos, no más, y no menos.

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
#Si intenta llamar a la función con 1 o 3 argumentos, obtendrá un error:
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil")
```

Argumentos arbitrarios, *args

Si no sabe cuántos argumentos se pasarán a la función, agregue un *antes del nombre del parámetro en la definición de función.

De esta manera, la función recibirá una *tupla* de argumentos y podrá acceder a los elementos en consecuencia:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

Los argumentos arbitrarios a menudo se acortan a **args* en la documentación de Python.

Argumentos de palabras clave

También puede enviar argumentos con la sintaxis *de clave - valor*.

De esta manera el orden de los argumentos no importa.

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

La frase *Argumentos de palabra clave* a menudo se acorta a *kwargs* en la documentación de Python.

Argumentos de palabras clave arbitrarias, **kwargs

Si no sabe cuántos argumentos de palabra clave se pasarán a la función, agregue dos asteriscos: antes del nombre del parámetro en la definición de función.**

De esta manera, la función recibirá un *diccionario* de argumentos y podrá acceder a los elementos en consecuencia:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

Valor de parámetro predeterminado

En el ejemplo siguiente se muestra cómo utilizar un valor de parámetro predeterminado.

Si llamamos a la función sin argumento, utiliza el valor predeterminado:

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Pasar una lista como argumento

Puede enviar cualquier tipo de argumento de datos a una función (cadena, número, lista, diccionario, etc.) y se tratará como el mismo tipo de datos dentro de la función.

Por ejemplo, si envía una lista como argumento, seguirá siendo una lista cuando llegue a la función:

```
def my_function(food):
  for x in food:
    print(x)
fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

Valores devueltos

Para permitir que una función devuelva un valor, utilice la instrucción:return

```
def my_function(x):
  return 5 * x
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

La Declaración de pase

function las definiciones no pueden estar vacías, pero si por alguna razón tiene una definición sin contenido, coloque la instrucción para evitar recibir un error.functionpass

```
def myfunction():
  pass
# tener una definición de función vacía como esta, generaría un error sin la instrucción pass
```

Fuentes: https://j2logo.com/bucle-for-en-python/