Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

* Equals: a == b
* Not Equals: a != b
* Less than: a < b
* Less than or equal to: a <= b
* Greater than: a > b
* Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

Example

If statement:

a = 33

b = 200

if b < a:

print("b is greater than a")

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Example

If statement, without indentation (will raise an error):

a = 33

## Elif

The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

### Example

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

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

## Else

The else keyword catches anything which isn't caught by the preceding conditions.

### Example

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")

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

You can also have an else without the elif:

### Example

a = 18  
b = 33  
if 17 > a:  
  print("b is greater than a")  
else:  
  print("b is not greater than a")

## Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

### Example

One line if statement:

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

## Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

### Example

One line if else statement:

a = 2  
b = 330  
print("A") if a > b else print("B")

This technique is known as **Ternary Operators**, or **Conditional Expressions**.

You can also have multiple else statements on the same line:

### Example

One line if else statement, with 3 conditions:

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

And

The and keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, AND if c is greater than a:

a = 200

b = 33

c = 500

if a > b and c > a:

print("Both conditions are True")

Both conditions are True

Or

The or keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, OR if a is greater than c:

a = 200

b = 33

c = 500

if a > b or a > c:

print("At least one of the conditions is True")

Not

The not keyword is a logical operator, and is used to reverse the result of the conditional statement:

Example

Test if a is NOT greater than b:

a = 33

b = 200

if not a > b:

print("a is NOT greater than b")

a is NOT greater than b

Nested If

You can have if statements inside if statements, this is called *nested* if statements.

Example

x = 41

if x > 10:

print("Above ten,")

if x > 20:

print("and also above 20!")

else:

print("but not above 20.")

Above ten,  
and also above 20!

# Python While Loops

## Python Loops

Python has two primitive loop commands:

* while loops
* for loops

## The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

### Example

Print i as long as i is less than 6:

i = 1

while i < 6:

print(i)

i += 1

**Note:** remember to increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

The break Statement

### Example

Exit the loop when i is 3:

With the break statement we can stop the loop even if the while condition is true:

i = 1

while i < 6:

print(i)

if (i == 3):

break

i += 1

## The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

### Example

Continue to the next iteration if i is 3:

i = 0

while i < 6:

i += 1

if i == 3:

continue

print(i)

# Note that number 3 is missing in the result

# Python For Loops

## Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

### Example

Print each fruit in a fruit list:

fruits = ["apple", "banana", "cherry"]

for x in fruits:

print(x)

The for loop does not require an indexing variable to set beforehand.

Looping Through a String

Even strings are iterable objects, they contain a sequence of characters:

Example

for x in "banana":

print(x)

b  
a  
n  
a  
n  
a

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

Example

Exit the loop when x is "banana":

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  print(x)  
  if x == "banana":  
    break

apple  
banana

Exit the loop when x is "banana", but this time the break comes before the print:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  if x == "banana":  
    break  
  print(x)

apple

The continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:

fruits = ["apple", "banana", "cherry"]

for x in fruits:

if x == "banana":

continue

print(x)

apple  
cherry

The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example

Using the range() function:

for x in range(6):

print(x)

0  
1  
2  
3  
4  
5