

Control Statements

if boolean-expression:

#statements

elif boolean-expression:

#statements

elif boolean-expression:

#statements

elif boolean-expression:

#statements

else:

#statements

if-else

- **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: $a \leq b$
- Greater than: $a > b$
- Greater than or equal to: $a \geq b$

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

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

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 = 200
```

```
b = 33
```

```
if b > 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.

One line if statement:

Example

```
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:

One line if else statement:

Example

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

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

One line if else statement, with 3 conditions:

Example

```
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")
```

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")
```

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.")
```

Python While 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 = i + 1
```


The break Statement

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

Example

Exit the loop when i is 3:

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

The else Statement

With the else statement we can run a block of code once when the condition no longer is true

Example

Print a message once the condition is false:

```
i = 1
while i < 6:
    print(i)
    i = i + 1
else:
    print("i is no longer less than 6")
```

Python For Loops

For Loop

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.

```
for <variable> in <sequence>:  
    # body_of_loop that has set of statements  
    # which requires repeated execution
```

Example

Print each fruit in a fruit list:

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

Looping Through a String

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

Example

Loop through the letters in the word "banana":

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

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
```

Break Statement Continued

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)
```


The continue Statement

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

Example

Do not print banana:

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

Pass Statement

```
# An empty loop
for letter in 'HelloAll':
    pass
print ('Last Letter :', letter)
```

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)
```

Note that `range(6)` is not the values of 0 to 6, but the values 0 to 5.

The `range()` function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: `range(2, 6)`, which means values from 2 to 6 (but not including 6):

Example

Using the start parameter:

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

The `range()` function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter:

`range(2, 30, 3)`:

Example

Increment the sequence with 3 (default is 1):

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

Else in For Loop

The `else` keyword in a `for` loop specifies a block of code to be executed when the loop is finished:

Example

Print all numbers from 0 to 5, and print a message when the loop has ended:

```
for x in range(6):  
    print(x)  
else:  
    print("Finally finished!")
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example

Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]  
fruits = ["apple", "banana", "cherry"]
```

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