

Python Indentations

Where in other programming languages the indentation in code is for readability only, in Python the indentation is very important.

Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:  
    print("Five is greater than two!")
```

Python will give you an error if you skip the indentation:

Example

```
if 5 > 2:  
print("Five is greater than two!")
```

Python Variables

In Python variables are created the moment you assign a value to it:

Example

Variables in Python:

```
x = 5  
y = "Hello, World!"
```

Python has no command for declaring a variable.

Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

Example

Comments in Python:

```
#This is a comment.  
print("Hello, World!")
```

String Literals

String literals in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the `print()` function:

Example

```
print("Hello")  
print('Hello')
```

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

```
a = "Hello"  
print(a)
```

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example

You can use three double quotes:

```
a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)
```

Or three single quotes:

Example

```
a = '''Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua.'''  
print(a)
```

Note: in the result, the line breaks are inserted at the same position as in the code.

Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

Example

Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"  
print(a[1])
```

Example

Substring. Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"  
print(b[2:5])
```

Example

The strip() method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "  
print(a.strip()) # returns "Hello, World!"
```

Example

The len() method returns the length of a string:

```
a = "Hello, World!"  
print(len(a))
```

Example

The lower() method returns the string in lower case:

```
a = "Hello, World!"  
print(a.lower())
```

Example

The upper() method returns the string in upper case:

```
a = "Hello, World!"  
print(a.upper())
```

Example

The replace() method replaces a string with another string:

```
a = "Hello, World!"  
print(a.replace("H", "J"))
```

Example

The split() method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!"  
print(a.split(",")) # returns ['Hello', ' World!']
```

Python Variables

Creating Variables

Variables are containers for storing data values.

Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Example

```
x = 5
y = "John"
print(x)
print(y)
```

Variables do not need to be declared with any particular type and can even change type after they have been set.

Example

```
x = 4 # x is of type int
x = "Sally" # x is now of type str
print(x)
```

String variables can be declared either by using single or double quotes:

Example

```
x = "John"
# is the same as
x = 'John'
```

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Remember that variable names are case-sensitive

Assign Value to Multiple Variables

Python allows you to assign values to multiple variables in one line:

Example

```
x, y, z = "Orange", "Banana", "Cherry"  
print(x)  
print(y)  
print(z)
```

And you can assign the *same* value to multiple variables in one line:

Example

```
x = y = z = "Orange"  
print(x)  
print(y)  
print(z)
```

Output Variables

The Python `print` statement is often used to output variables.

To combine both text and a variable, Python uses the `+` character:

Example

```
x = "awesome"  
print("Python is " + x)
```

You can also use the `+` character to add a variable to another variable:

Example

```
x = "Python is "  
y = "awesome"  
z = x + y  
print(z)
```

For numbers, the `+` character works as a mathematical operator:

Example

```
x = 5  
y = 10  
print(x + y)
```

If you try to combine a string and a number, Python will give you an error:

Example

```
x = 5  
y = "John"  
print(x + y)
```

Python 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 <= 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, using whitespace, 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
b = 200
if b > a:
    print("b is greater than a") # you will get an error
```

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 to `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.

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:

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

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

Example

One line if else statement, with 3 conditions:

```
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`:

```
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`:

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

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

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 += 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)
```

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

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
```

Example

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)
```

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)
```

Introduction to the if Statement

We'll start by looking at the most basic type of if statement. In its simplest form, it looks like this:

```
if <expr>:
    <statement>
```

In the form shown above:

- <expr> is an expression evaluated in Boolean context, as discussed in the section on [Logical Operators](#) in the Operators and Expressions in Python tutorial.
- <statement> is a valid Python statement, which must be indented. (You will see why very soon.)

If <expr> is true (evaluates to a value that is “truthy”), then <statement> is executed. If <expr> is false, then <statement> is skipped over and not executed.

Note that the colon (:) following <expr> is required. Some programming languages require <expr> to be enclosed in parentheses, but Python does not.

Here are several examples of this type of if statement:

```
>>>
>>> x = 0
>>> y = 5

>>> if x < y:                                     # Truthy
...     print('yes')
...
yes
>>> if y < x:                                     # Falsy
...     print('yes')
...
...

```

```
>>> if x:                                     # Falsy
...     print('yes')
...
>>> if y:                                     # Truthy
...     print('yes')
...
yes

>>> if x or y:                               # Truthy
...     print('yes')
...
yes

>>> if x and y:                             # Falsy
...     print('yes')
...

>>> if 'aul' in 'grault':                   # Truthy
...     print('yes')
...
yes

>>> if 'quux' in ['foo', 'bar', 'baz']:     # Falsy
...     print('yes')
...

```

Infinity is a special case of floating point numbers. It can be obtained by `float('inf')`.