Python

Syntax

Python syntax can be executed by writing directly in the Command Line:

```
>>> print("Hello, World!")
Hello, World!
```

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

```
$ python myfile.py
```

Indentation

Indentation refers to the spaces at the beginning of a code line.

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

Python uses indentation to indicate a block of code.

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

- Python will give you an error if you skip the indentation
    - The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.
- You have to use the same number of spaces in the same block of code, otherwise Python will give you an error

Variables

In Python, variables are created when you assign a value to it:

```
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 can be used to:

- Explain Python code
- Make the code more readable
- Prevent execution when testing code

Creating a Comment

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

```
# This is a comment
```

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

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

A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

```
# print("Hello, World!")
print("Cheers, Mate!")
```

Multiline Comments

Python does not really have a syntax for multiline comments.

To add a multiline comment you could insert a # for each line:

```
# This is a comment
# written in
# more than just one line
print("Hello, World!")
```

Or, not quite as intended, you can use a multiline string.

> Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

- As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.

Variables

Creating Variables

Python has no command for declaring a variable.

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

```
x = 5
y = "John"
```

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

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

Casting

If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

Get the Type

You can get the data type of a variable with the `type()` function.

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

Single or Double Quotes?

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

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

Case-Sensitive

Variable names are case-sensitive.

```
a = 4
A = "Sally"
# A will not overwrite a
```

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:

- Must start with a letter or the underscore character
- Cannot start with a number
- 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)
- Cannot be any of the Python keywords.

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

> Remember that variable names are case-sensitive

Multi Words Variable Names

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

- Camel Case: `myVariableName`
- Pascal Case: `MyVariableName`
- Snake Case: `my_variable_name`

Assign Multiple Values

Many Values to Multiple Variables

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

```
x, y, z = "Orange", "Banana", "Cherry"
```

> **Note**: Make sure the number of variables matches the number of values, or else you will get an error.

One Value to Multiple Variables

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

```
x = y = z = "Orange"
```

Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
```

Output Variables

The Python `print()` function is often used to output variables.

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

In the `print()` function, you output multiple variables, separated by a comma:

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

You can also use the + operator to output multiple variables:

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

> Notice the space character after `"Python "` and `"is "` without them the result would be "Pythonisawesome".

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

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

> In the `print()` function, when you try to combine a string and a number with the + operator, Python will give you an error

The best way to output multiple variables in the `print()` function is to separate them with commas, which even support different data types:

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

Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

```
x = "awesome"

def myfunc():
    print("Python is " + x)

myfunc()
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

```
x = "awesome"

def myfunc():
    x = "fantastic"
    print("Python is " + x)

myfunc()

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

The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the `global` keyword.

```
def myfunc():
    global x
    x = "fantastic"

myfunc()

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

Also, use the `global` keyword if you want to change a global variable inside a function.

```
x = "awesome"

def myfunc():
    global x
    x = "fantastic"

myfunc()

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

Data Types

Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type	`str`
Numeric Types	`int`, `float`, `complex`
Sequence Types	`list`, `tuple`, `range`
Mapping Type	`dict`
Set Types	`set`, `frozenset`
Boolean Type	`bool`

Binary Types	`bytes`, `bytearray`, `memoryview`
None Type	`NoneType`

Getting the Data Type

You can get the data type of any object by using the `type()` function:

```
x = 5
print(type(x))
```

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	Data Type
`x = "Hello World"`	`str`
`x = 20`	`int`
`x = 20.5`	`float`
`x = 1j`	`complex`
`x = ["apple", "banana", "cherry"]`	`list`
`x = ("apple", "banana", "cherry")`	`tuple`
`x = range(6)`	`range`
`x = {"name" : "John", "age" : 36}`	`dict`
`x = {"apple", "banana", "cherry"}`	`set`
`x = frozenset({"apple", "banana", "cherry"})`	`frozenset`
`x = True`	`bool`
`x = b"Hello"`	`bytes`
`x = bytearray(5)`	`bytearray`
`x = memoryview(bytes(5))`	`memoryview`
`x = None`	`NoneType`

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
`x = str("Hello World")`	`str`
`x = int(20)`	`int`
`x = float(20.5)`	`float`
`x = complex(1j)`	`complex`
`x = list(("apple", "banana", "cherry"))`	`list`
`x = tuple(("apple", "banana", "cherry"))`	`tuple`
`x = range(6)`	`range`
`x = dict(name="John", age=36)`	`dict`
`x = set(("apple", "banana", "cherry"))`	`set`
`x = frozenset(("apple", "banana", "cherry"))`	`frozenset`
`x = bool(5)`	`bool`
`x = bytes(5)`	`bytes`
`x = bytearray(5)`	`bytearray`
`x = memoryview(bytes(5))`	`memoryview`

Numbers

There are three numeric types in Python:

- `int`
- `float`
- `complex`

Variables of numeric types are created when you assign a value to them:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

To verify the type of any object in Python, use the `type()` function:

```
print(type(x))
print(type(y))
print(type(z))
```

int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

```
x = 1
y = 35656222554887711
z = -3255522
print(type(x))
print(type(y))
print(type(z))
```

float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

```
x = 1.10
y = 1.0
z = -35.59
print(type(x))
print(type(y))
print(type(z))
```

Float can also be scientific numbers with an "e" to indicate the power of 10.

```
x = 35e3
y = 12E4
z = -87.7e100
```

complex

Complex numbers are written with a "j" as the imaginary part:

```
x = 3+5j
y = 5j
z = -5j
print(type(x))
print(type(y))
print(type(z))
```

Type Conversion

You can convert from one type to another with the `int()`, `float()`, and `complex()` methods:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
# convert from int to float:
a = float(x)
# convert from float to int:
b = int(y)
# convert from int to complex:
c = complex(x)
print(a)
print(b)
print(c)
print(type(a))
print(type(b))
print(type(c))
```

Note: You cannot convert complex numbers into another number type.

Random Number

Python does not have a `random()` function to make a random number, but Python has a built-in module called `random` that can be used to make random numbers:

```
import random
print(random.randrange(1, 10))
```

Casting

Specify a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

- **`int()`** - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- **`float()`** - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- **`str()`** - constructs a string from a wide variety of data types, including strings, integer literals and float literals

int

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

float

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
```

string

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```