Chapter 2: Intro to python

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- Web development (server-side),
- Software development,
- Mathematics,
- Machine learning.

What can Python do?

- Python can be used on a server to create web applications.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, etc.).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with few lines.
- Python runs on an interpreter system, meaning that code can be executed fast as soon as it is written.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Python Syntax compared to other programming languages

1. Python was designed for readability, and has some similarities to the English language with influence from mathematics.
2. Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
3. Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

1. Python 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!")
```

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

The number of spaces is up to programmer, the most common use is four, but it has to be at least one.

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

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

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

print() method

```
print("test") same as print('test')
print(2024)
print(5+6)
print("a", "b", "c", sep="|")
print(10*"test\n")
~~print("a" + 5)~~, print("a", 5) or print("a" + str(5))
```

input

```
name = input("Enter name: ")
print("hello" + name)
or
print("Enter name")
name = str(input())
print("hello" + name)
```

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

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

```
x = 4       # x is of type int
x = "Jo"    # 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 = "Jo"
print(type(x))      # int
print(type(y))      # str
```

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

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

Variable names are case-sensitive.

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

3. 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)
- A variable name cannot be any of the Python keywords listed below.

Keyword	Description
`and`	A logical operator
`as`	To create an alias
`assert`	For debugging
`break`	To break out of a loop
`class`	To define a class
`continue`	To continue to the next iteration of a loop
`def`	To define a function
`del`	To delete an object
`elif`	Used in conditional statements, same as else if
`else`	Used in conditional statements
`except`	Used with exceptions, what to do when an exception occurs
`False`	Boolean value, result of comparison operations
`finally`	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
`for`	To create a for loop
`from`	To import specific parts of a module
`global`	To declare a global variable
`if`	To make a conditional statement
`import`	To import a module

in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
True	Boolean value, result of comparison operations
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To end a function, returns a generator

Legal variable names:

```
myvar = "Jo"
my_var = "Jo"
_my_var = "Jo"
MYVAR = "Jo"
myvar2 = "Jo"
```

Illegal variable names:

```
2myvar = "Jo"
my-var = "Jo"
my var = "Jo"
```

Many Values to Multiple Variables

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

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

One Value to Multiple Variables

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

```
x = y = z = "Orange"
print(x, y, z, sep="+")      #Orange+ Orange+ 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
print(x)
print(y)
print(z)
```

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 easy'
print(x, y)     # add space between variables by default
print(x + y)    # no space between variables
```

You can also use the `+` operator to output multiple variables having same type. For numbers, the `+` character works as a mathematical operator:

```
x, y = 5, 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:

```
x = 5
y = ”John”
print(x + y)     #error

# or you can make both variables as same type

print(str(x) + ' ' + y)
```

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 = "easy"

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

display()
display()
```

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 display():
  x = "fantastic"
  print("Python is " + x)

display()
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 `global` keyword.

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

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

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

```
x = "awesome"

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

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

4. Comments

Python has commenting capability for the purpose of in-code documentation. Comments can be used to explain Python code and to make the code more readable. Comments can be used to prevent execution when testing code.

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

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

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


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


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