



INTRODUCTION TO Programming USING **PYTHON**

001_Python

About the Founder,
Versions Python, IDE's

Introduction to Python

Python is a very *simple* language and has a very straightforward syntax. It helps programmers to program without **Boilerplate** (prepared code).

It has efficient high-level data structures and an Elegant syntax and Dynamic typing, together with its Interpreted nature, makes it an ideal language for scripting and rapid application development in many areas on most platforms.

Boilerplate

```
//JAVA
class Test
{
    public static void main(String[] args)
    {
        System.out.println("Hello World.");
    }
}
```

```
#Python
print("Hello World")
```

About the Founder

Python was developed by **Guido Van Rossum** in 1989 while working at National Research Institute at Netherlands.

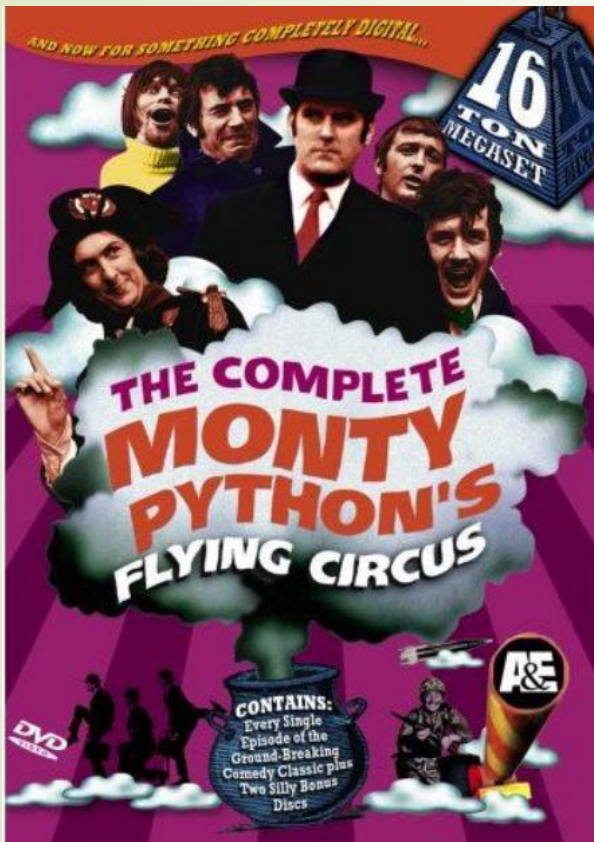
But officially Python was made available to public in 1991. With the official Date of Birth for Python is Feb 20th, 1991.



“Python is an experiment on how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered.”

Guido Van Rossum

Fun fact – About the name



The name Python was selected from the TV Show.

"The Complete Monty Python's Flying Circus",

which was broadcasted in BBC from 1969 to 1974.

This may seem irrelevant, but it does have a subtle impact on how the Python community presents itself.

For instance, you might come across references to Monty Python's Flying Circus within the community.

Examples include nods to the "Cheese Shop" skit or a general sense of playful humor woven into Python's culture and documentation.

Versions of Python



There are two major versions of the Python language, Python 2, which is the widely-deployed legacy language, and Python 3, which is the present and future of the language.



It's now over a decade since the transition from Python 2 to Python 3 was begun, Python 2 is not maintained from the year 2020.



We'll be using Python 3 for this course, and everything we show will work on Python 3.6 or later, and most things will work on Python 3.3 or later.

Where is Python used?

Python programming is a skill that can be used in virtually any industry. From industries like finance, healthcare, and insurance, to fields like aerospace to entertainment – Python is driving innovation and new solutions.

Python has broad uses – but it's becoming more important specifically in the following areas:

- **Data Analytics**

- **Machine learning**

- **Artificial intelligence**

- **Task automation**

- **Computing security**

- **Web development**

Integrated development environment (IDE)

The IDE used in this course is Thonny there is a few reasons why I used this IDE, Then demonstrating python, It lacks unneeded functions, its open source and free of influence of big companies. & of course, my own personal preference.

Thonny
Python IDE for beginners

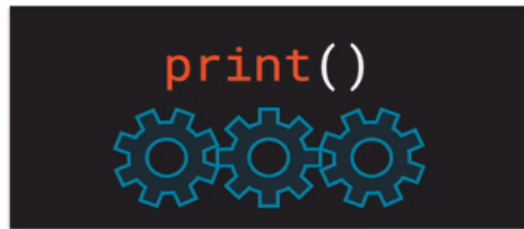
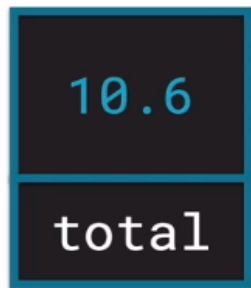
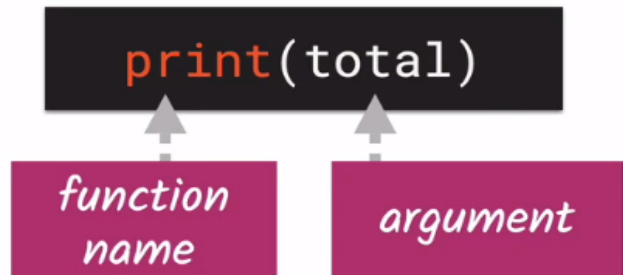


Download version 4.1.7 for
Windows • Mac • Linux

<https://thonny.org/>

Authors note:

You can use any IDE of your own personal preference,
(VS code, PyCharm, notepad++, ...)



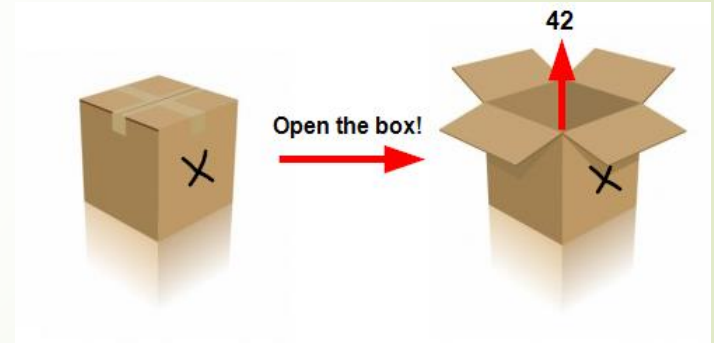
*You can think of this function
as a black box machine.
We don't know how it works...*

*But we do know it
will output the given
value to the screen.*

Variables, Constants and Identifiers

Variables

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data that can be changed later in the program.



```
#Operator  
number = 42
```

Here, we have created a variable named number. We have assigned the value 42 to the variable.

Assigning values to Variables in Python

As you can see from the above example, you can use the assignment operator = to assign a value to a variable.

Another great thing about Python is that the value as well as the data type of the variable can be changed during the code execution.

As you can see in the output, from the next slide, the data type of the variable was changed in the same code execution.

The Python function "type" is used to get the information about the data type of the variable.

```
print(type(42))  
print(type(42.01))  
print(type("hello"))  
print(type(True))  
print(type(1+1j))  
  
print(type([]))  
print(type(()))  
print(type(set()))  
print(type(frozenset()))  
print(type({}))
```

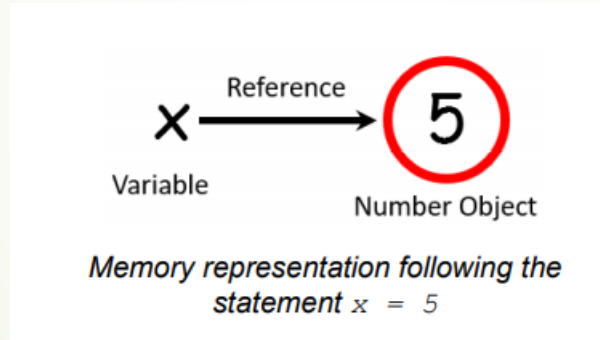
How is a value represented in memory?

In Python, we don't assign values to the variables. Instead, Python gives the reference of the object(value) to the variable.

The values are represented in memory as objects and a reference links the variable to the object.

Memory Representation

The diagrams below illustrate what happens when we initialise the variable `x` to 5.



We will revisit this later, when we learn about immutability of objects

Assigning multiple values to multiple variables

There are ways to Assigning multiple values to multiple variables in on statement.

One way is by a Unpacking a Tuple (more on this latter), you need a variable name for each of the values.

Then you can all so referencing this, is more like aliasing the variable , as such they haver the same ID (more on this latter)

Multiple Assigning can be on one line if Statement with “;”

```
a = 1; b = 2; c = 3  
print(a,b,c)
```

```
# Unpacking  
a,b,c =5,2.1,"Hello"  
print(a,b,c)
```

```
#referencing  
x=y = "Same"  
print(x,y)  
  
print(id(x))  
print(id(y))  
print(x is y)
```


Constants

A constant is a type of variable whose value should not be changed. It is helpful to think of constants as containers that hold information which cannot be changed later.


For example, The number of seconds in one hour is a constant as well; it will never change: there will always be 3,600 seconds in an hour.

Example : Declaring and assigning value to a constant

```
PI = 3.14  
GRAVITY = 9.8  
MAX_VALUE = 10
```

Authors note:

Naming them in all capital letters is a convention to separate them from variables, however, it does not actually prevent reassignment.



An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

Identifiers in programming can consist of letters (a-z, A-Z), digits (0-9), and underscores (_), but they cannot start with a digit. For example, `variable1` is valid, while `1variable` is not.

Keywords such as `in`, `for`, `if`, and so on cannot be used as identifiers, and special symbols like `!` `@` `#` `$` `%` are not allowed. Although an identifier can be of any length, it is best to keep them concise and meaningful for readability.

Keywords

| 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 |

| Keyword | Description |
|----------|---|
| 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 |

Keywords

| Keyword | Description |
|---------|--|
| 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 return a list of values from a generator |

Keywords are the reserved words in Python.

We cannot use a keyword as a variable name, function name or any other identifier.

They are used to define the syntax and structure of the Python language.

In Python, keywords are case sensitive. All the keywords except True, False and None are in lowercase and they must be written as they are,

Python Statement

Instructions that a Python interpreter can execute are called statements.

For example, `a = 1` is an assignment statement.

if statement, *for* statement, *while* statement, etc. are other kinds of statements which will be discussed later.

```
a = 1; b = 2; c = 3  
print(a,b,c)
```

```
print("Hi"); print("how are you")
```

Multi-line statement

In Python, the end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (`\`).


This is an explicit line continuation. In Python, line continuation is implied inside parentheses `()`, brackets `[]`, and braces `{ }`

We can also put multiple statements in a single line using semicolons, as follows:

```
a = 1; b = 2; c = 3
```

```
my_sum= 1+2+3+4+ \  
        5+6+7+8  
print(my_sum)
```

```
my_sum= (1+2+3+4+  
        5+6+7+8)  
print(my_sum)
```




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.

Indentation refers to the whitespaces (usually 4 spaces or a single tab) that signify the beginning of a suite (block) of code. **All statements indented at the same level belong to the same suite.****



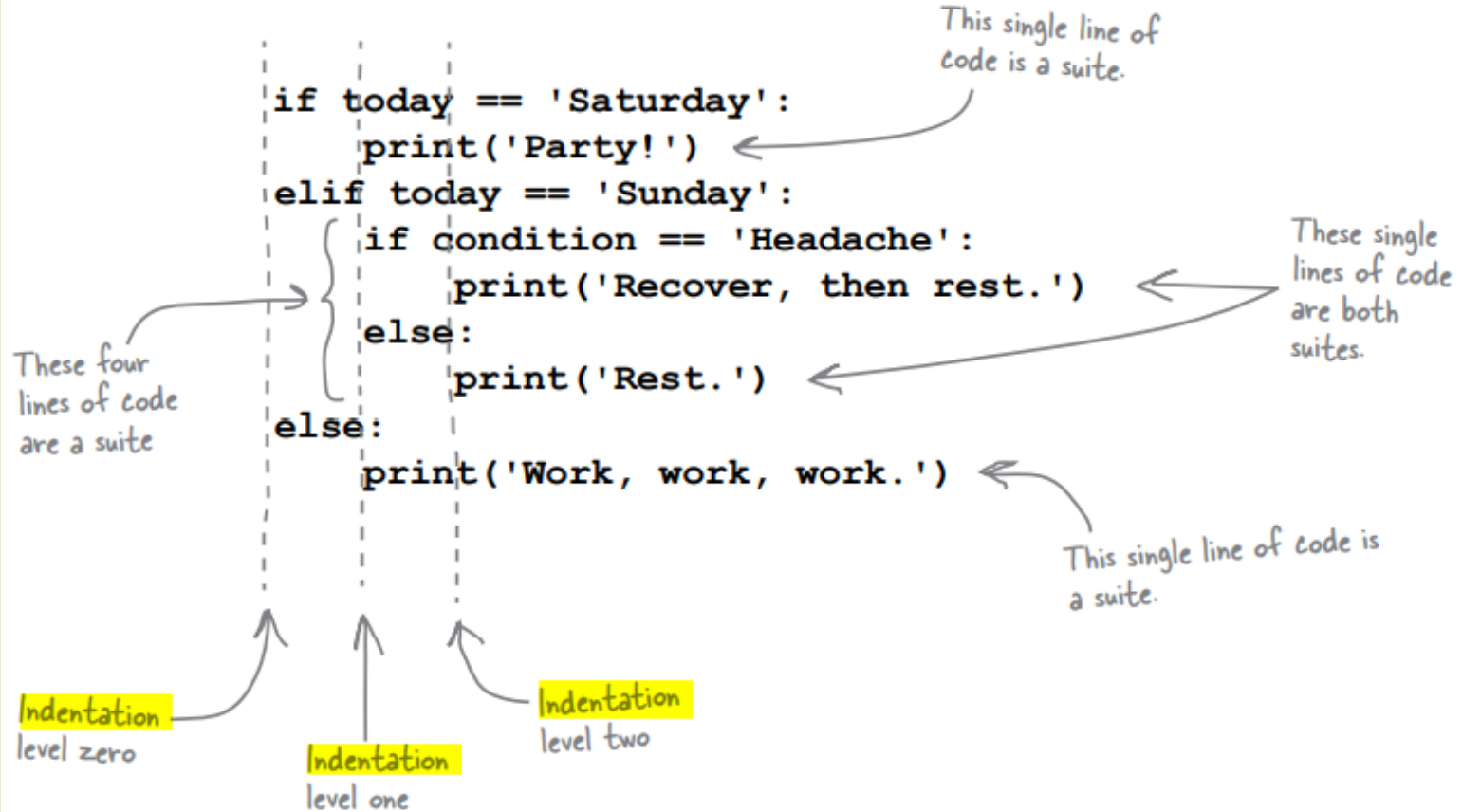
Indentation

The colon (:) introduces a new suite(block) of code that must be intended to right.

The interpreter raises an error if you forgot to indent your statements after a colon.

The end of the suite is specified by unindenting the next line of code(which is not the part of your suite).

Indentation Example



Comments

Comments provide a way to leave an explanation or annotation in your source code. Comments are programmer-readable and are added to make the source code easier for a programmer to understand.

When the Python interpreter comes across a comment, it ignores the comment and moves to the next line of code.

There are two types of commenting features available in Python:

- ▀ These are single-line comments and multi-line comments.

Single-line comment

A single-line comment begins with a hash (#) symbol and is useful in mentioning that the whole line should be considered as a comment until the end of the line.

In the below example program, the first line starts with the hash symbol, so the entire line is considered a comment.

In the second line of code, "number = 50" is a statement, and after the statement, the comment begins with the # symbol. From the # symbol to the end of this line, the line will be treated as a comment.(inline comment)

```
#Defining a variable to store a number  
number = 50 # Store 50 as a value
```

Multi-line comment

Multi-line comment is useful when we need to comment on many lines.

Python doesn't offer a separate way to write multiline comments.

We can use # at the beginning of each line of comment on multiple lines.

Here, each line is treated as a single comment and all of them are ignored.

```
# this is a  
# multiline  
# comment
```

Conventions and Pet Peeves

According to **PEP 8**, comments should always be written in complete sentences with a single space between the # and the first word of the comment:

```
# This comment is formatted to PEP 8.
```

```
#this one isn't
```

For inline comments, PEP 8 recommends at least two spaces between the code and the # symbol

```
message= "Hello world" # This comment is PEP 8 compliant.  
print(message)# This comment isn't
```

Using String Literals to write Multi-line Comments

Here, the multiline string isn't assigned to any variable, so it is ignored by the interpreter.

Even though it is not technically a multiline comment, it can be used as one.

```
"""
```

```
This is our first program  
and it shows how we can use three double quotes  
to use as a Multi-line Comments
```

```
"""
```

```
'''
```

```
This is our first program  
and it shows how we can use three single quotes  
to use as a Multi-line Comments
```

```
'''
```

Docstrings in Python

A docstring is short for documentation string.

Python docstrings (documentation strings) are the string literals that appear right after the definition of a function, method, class, or module. This separates docstrings from multiline comments using triple quotes.

Triple quotes are used while writing docstrings.

```
def double(number):  
    """Function to double the value"""  
    return 2*number
```

Python Comments vs Docstrings

Python Comments

Comments are descriptions that help programmers better understand the intent and functionality of the program.

Python docstrings

As mentioned above, Python docstrings are strings used right after the definition of a function, method, class, or module. They are used to document our code.