[ ] Start coding or generate with AI.

## 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,

system scripting.

## What can Python do?

Python can be used on a server to create web applications.

Python can be used alongside software to create workflows.

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.

Python can be used for rapid prototyping, or for production-ready software development.

## Why Python?

Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).

Python has a simple syntax similar to the English language.

Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

Python can be treated in a procedural way, an object-oriented way or a functional way.

## Python Syntax compared to other programming languages

Python was designed for readability, and has some similarities to the English language with influence from mathematics.

Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.

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.

#Python Syntax

## Execute Python Syntax

As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

## 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!")
```

```
Five is greater than two!
```

Double-click (or enter) to edit

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

Example

Syntax Error:

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

```
  File "/tmp/ipython-input-3793329317.py", line 2
    print("Five is greater than two!")
    ^
IndentationError: expected an indented block after 'if' statement on line 1
```

## 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.

Example

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

```
Five is greater than two!
Five is greater than two!
```

## Python Variables

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

Example

Variables in Python:

Python has no command for declaring a variable.

## Variables

Variables are containers for storing data values.

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"
print(x)
print(y)
```

```
5
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)
```

```
Sally
```

## 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

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

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

```
<class 'int'>
<class 'str'>
```

## 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.

Example

This will create two variables:

```
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:

## 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.

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

## Example

Illegal variable names:

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

```
  File "/tmp/ipython-input-2185399839.py", line 1
    2myvar = "John"
    ^
SyntaxError: invalid decimal literal
```

Double-click (or enter) to edit

```
myVariableName = "John"
```

## 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

Each word, except the first, starts with a capital letter:

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

```
Orange
Banana
Cherry
```

## One Value to Multiple Variables

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

Double-click (or enter) to edit

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

```
Orange
Orange
Orange
```

Double-click (or enter) to edit

# Python - Output Variables

## Output Variables

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

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

```
Python is awesome
```

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

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

```
Python is awesome
```

## We can also use the + operator to output multiple variables:

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

```
Python is awesome
```

```

```

## 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:

## Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

Double-click (or enter) to edit

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

```
Hello, World!
```

## 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!")
```

```
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!")
```

```
Hello, World!
```

```
Start coding or generate with AI.
```

## Python 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` |

| Example | Data Type | Try it |
|---|---|---|
| `x = "Hello World"` | str | Try it » |
| `x = 20` | int | Try it » |
| `x = 20.5` | float | Try it » |
| `x = 1j` | complex | Try it » |
| `x = ["apple", "banana", "cherry"]` | list | Try it » |
| `x = ("apple", "banana", "cherry")` | tuple | Try it » |
| `x = range(6)` | range | Try it » |
| `x = {"name" : "John", "age" : 36}` | dict | Try it » |
| `x = {"apple", "banana", "cherry"}` | set | Try it » |
| `x = frozenset({"apple", "banana", "cherry"})` | frozenset | Try it » |
| `x = True` | bool | Try it » |

| | | |
|---|---|---|
| `x = b"Hello"` | bytes | Try it » |
| `x = bytearray(5)` | bytearray | Try it » |
| `x = memoryview(bytes(5))` | memoryview | Try it » |
| `x = None` | NoneType | Try it » |

## Python Lists

mylist = ["apple", "banana", "cherry"]

## List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

## ExampleGet your own Python Server

Create a List:

thislist = ["apple", "banana", "cherry"]

print(thislist)

## List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

## Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

## Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

## Example

Lists allow duplicate values:

Lists allow duplicate values:

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry', 'apple', 'cherry']
```

## List Length

To determine how many items a list has, use the len() function:

Example

Print the number of items in the list:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

```
3
```

## List Items - Data Types

List items can be of any data type:

Example

String, int and boolean data types:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]

print(list1)
print(list2)
print(list3)
```

```
['apple', 'banana', 'cherry']
[1, 5, 7, 9, 3]
[True, False, False]
```

## A list can contain different data types:

Example

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]

print(list1)
```

```
['abc', 34, True, 40, 'male']
```

## type()

From Python's perspective, lists are defined as objects with the data type 'list':

<class 'list'>

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

```
<class 'list'>
```

## The list() Constructor

It is also possible to use the list() constructor when creating a new list.

Example

Using the list() constructor to make a List:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

## Python - Remove List Items

## Remove Specified Item

The remove() method removes the specified item.

ExampleGet your own Python Server

Remove "banana":

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

```
['apple', 'cherry']
```

## If there are more than one item with the specified value, the remove() method removes the first occurrence:

Example

Remove the first occurrence of "banana":

```
thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
```

```
['apple', 'cherry', 'banana', 'kiwi']
```

## Remove Specified Index

The pop() method removes the specified index.

## Example

Remove the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

```
['apple', 'cherry']
```

If you do not specify the index, the pop() method removes the last item.

Example

Remove the last item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

```
['apple', 'banana']
```

The del keyword also removes the specified index:

Example

Remove the first item:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

```
['banana', 'cherry']
```

The del keyword can also delete the list completely.

Example

Delete the entire list:

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

Example

Clear the list content:

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

```
[]
```

## Python - Loop Lists

### Loop Through a List

You can loop through the list items by using a for loop:

Print all items in the list, one by one:

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

```
apple
banana
cherry
```

### Python Tuples

mytuple = ("apple", "banana", "cherry")

### Tuple

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and unchangeable.

Tuples are written with round brackets.

### Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

### Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

### Allow Duplicates

Since tuples are indexed, they can have items with the same value:

Double-click (or enter) to edit

```
#Example
##Tuples allow duplicate values:

thistuple = ("apple", "banana", "cherry", "apple", "cherry")
```

```
print(thistuple)
```

```
('apple', 'banana', 'cherry', 'apple', 'cherry')
```

## Tuple Length

To determine how many items a tuple has, use the len() function:

```
#Example
##Print the number of items in the tuple:

thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

```
3
```

## Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
#Example
##One item tuple, remember the comma:

thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

```
<class 'tuple'>
<class 'str'>
```

## Tuple Items - Data Types

Tuple items can be of any data type:

### Example

String, int and boolean data types:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

## A tuple can contain different data types:

### Example

A tuple with strings, integers and boolean values:

```
tuple1 = ("abc", 34, True, 40, "male")
print(tuple1)
```

```
('abc', 34, True, 40, 'male')
```

## Access Tuple Items

We can access tuple items by referring to the index number, inside square brackets:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```