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

```
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

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!")
```

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!")
```

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 = 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 = 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:

```
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 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:

```
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

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)
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
```

```
```

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:

```
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:

```
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:

```
#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.
```

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

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.

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:

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:

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:

```
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:

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

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")
```

```
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.

```
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.

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

Example

Remove the last item:

Remove the last item.

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

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

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 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:

Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

Example

Clear the list content:

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'>
```

```
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
```

```
n = 4
for i in range(0, n):
    print(i)
```

```
0
1
2
3
```

Example: Iterating Over List, Tuple, String and Dictionary Using for Loops in Python

```
li = ["ICAR", "for", "Soybean"]
for x in li:
    print(x)

tup = ("ICAR", "for", "Soybean")
for x in tup:
    print(x)

s = "ICAR"
for x in s:
    print(x)

d = dict({'x':123, 'y':354})
for x in d:
    print("%s  %d" % (x, d[x]))

set1 = {10, 30, 20}
for x in set1:
    print(x),
```

```
ICAR
for
Soybean
ICAR
for
Soybean
I
C
A
R
x  123
y  354
10
20
30
```

Iterating by Index of Sequences

```
ICAR
for
Soybean
ICAR
```

```
for
Soybean
I
C
A
R
x  123
y  354
10
20
30
```

Iterating by Index of Sequences

```
Soybean
```

While Loop

In Python, a while loop is used to execute a block of statements repeatedly until a given condition is satisfied. When the condition becomes false, the line immediately after the loop in the program is executed.

[7] ✓ 0s

```
cnt = 0
while (cnt < 3):
    cnt = cnt + 1
    print("Hello ICAR")
```

```
Hello ICAR
Hello ICAR
Hello ICAR
```

Infinite While Loop

If we want a block of code to execute infinite number of times then we can use the while loop in Python to do so.

Code given below uses a 'while' loop with the condition "True", which means that the loop will run infinitely until we break out of it using "break" keyword or some other logic.

Do not use it

[ ]

```
while (True):
    print("Hello ICAR")
```

Nested Loops

Python programming language allows to use one loop inside another loop which is called nested loop. Following example illustrates the concept.

[8] ✓ 0s

```
for i in range(1, 5):
    for j in range(i):
        print(i, end=' ')
    print()
```

```
1
2 2
3 3 3
4 4 4 4
```

Loop Control Statements

Loop control statements change execution from their normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

Continue Statement

The continue statement in Python returns the control to the beginning of the loop.

Double-click (or enter) to edit

[9]
✓ 0s

```
for letter in 'ICARFORSOYBEAN':
    if letter == 'e' or letter == 's':
        continue
    print('Current Letter :', letter)
```

Continue Statement

The continue statement in Python returns the control to the beginning of the loop.

Double-click (or enter) to edit

[9]
✓ 0s

```
for letter in 'ICARFORSOYBEAN':
    if letter == 'e' or letter == 's':
        continue
    print('Current Letter :', letter)
```

```
Current Letter : I
Current Letter : C
Current Letter : A
Current Letter : R
Current Letter : F
Current Letter : O
Current Letter : R
Current Letter : S
Current Letter : O
Current Letter : Y
Current Letter : B
Current Letter : E
Current Letter : A
Current Letter : N
```

Break Statement

Pass Statement

We use pass statement in Python to write empty loops. Pass is also used for empty control statements, functions and classes.

[12]

```
for letter in 'ICARFORSOYBEAN':
    pass
print('Last Letter :', letter)
```

```
Last Letter : N
```

Pass Statement

We use pass statement in Python to write empty loops. Pass is also used for empty control statements, functions and classes.

[12]

```
for letter in 'ICARFORSOYBEAN':
```

```
    pass
print('Last Letter :', letter)
```

```
Last Letter : N
```