What is Python?
Python is a popular programming language. It was created in 1991 by Guido van Rossum.
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.

Good to know

The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.

In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thorny, Pycharm, Net beans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

Python was designed to 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:

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:*

```
C:\Users\Your Name>python myfile.py
```

Python Indentations:

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

Python uses indentation to indicate a block of code

Example:

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

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:

Example:

Comments in Python:

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

Docstrings:

Python also has extended documentation capability, called docstrings.
Docstrings can be one line, or multiline.
Python uses triple quotes at the beginning and end of the docstring:

Example
Docstrings are also comments:

"""This is a multiline docstring."""

print("Hello, World!")

Python Variables

Creating Variables:

Unlike other programming languages, Python has no command for declaring a variable.

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

Example:

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

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

Example:

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

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)

Output Variables:

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the + character:

Example 1:

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

ANSWER?

Example 2:

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

ANSWER?

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

Example:

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

If you try to combine a string and a number, Python will give you an error:

Example:

```
x = 5
y = "JOHN"
print(x + y)
```

Python Numbers:

Python Numbers:

There are **three** *numeric types in Python:*

- **int**
- **float**
- **Complex**

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

Example:

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

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

Example

```
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.

Example:

Integers:

```
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.

Example

Floats:

```
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.

Example

Floats:

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

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

Complex:

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

Example

Complex:

```
x = 3+5j
y = 5j
z = -5j

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

Python 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 rounding down to the previous whole number), 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

Example:

Integers:

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

Floats:

```
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
```

Strings:

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

Python Strings

Example:

•Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"
print(a[1])
```

Substring:

- Get the characters from position 2 to position 5 :

```
b = "Hello, World!"
print(b[2:5])
```

The **strip()** method removes any whitespace from the beginning or the end:

```
a = " Hello,  World! "
print(a.strip())
# returns "Hello, World!"
```

len():

The **len()** method returns the length of a string:

```
a = "Hello, World!"
print(len(a))
```

•lower():

- The lower() method returns the string in lower case:
- **a = "Hello, World!"**
  print(a.lower())

Example

•upper():

The upper() method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())
```

replace()

The replace() method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

Example:

split()

The split() method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!“

print(a.split(","))

# returns ['Hello', 'World!']
```

Command-line String Input:

Python allows for command line input.
That means we are able to ask the user for input.
The following example asks for the user's name, then, by using the input() method, the program prints the name to the screen:

```
x = input("Enter your name:")
print("Hello, " + x)
```

Python Operators

Python Operators:
Operators are used to perform operations on variables and values.
Python divides the operators in the following groups:

Arithmetic operators
Assignment operators
Comparison operators
Logical operators
Identity operators
Membership operators
Bitwise operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example	
+	Addition	x + y	
-	Subtraction	x - y	
*	Multiplication	x * y	
/	Division	x / y	
%	Modulus	x % y	
**	Exponentiation	x ** y	
//	Floor division	x // y	

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As	
=	x = 5	x = 5	
+=	x += 3	x = x + 3	
-=	x -= 3	x = x - 3	
*=	x *= 3	x = x * 3	
/=	x /= 3	x = x / 3	
%=	x %= 3	x = x % 3	
//=	x //= 3	x = x // 3	
**=	x **= 3	x = x ** 3	
&=	x &= 3	x = x & 3	
|=	x |= 3	x = x | 3	
^=	x ^= 3	x = x ^ 3	
>>=	x >>= 3	x = x >> 3	
<<=	x <<= 3	x = x << 3	

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Python List
s

Python Collections (Arrays)

There are four collection data types in the Python programming language:

List is a collection which is ordered and changeable. Allows duplicate members.

Tuple is a collection which is ordered and unchangeable. Allows duplicate members.

Set is a collection which is unordered and unindexed. No duplicate members.

Dictionary is a collection which is unordered, changeable and indexed. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

List

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Example

Create a List:

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

Access Items

You access the list items by referring to the index number:

Example:

Print the second item of the list:

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

Change Item Value:

To change the value of a specific item, refer to the index number:

Example

Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = “MANGO"
print(thislist)
```

Loop Through a List

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

Example

Print all items in the list, one by one:

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

Check if Item Exists

To determine if a specified item is present in a list use the `in` keyword:

Example

Check if "apple" is present in the list:

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

if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

List Length

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

Example:

Print the number of items in the list:

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

- **Add Items**
- To add an item to the end of the list, use the append() method:
- Example
- Using the **append()** method to append an item:

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

thislist.append("orange")

print(thislist)
```

insert()

To add an item at the specified index, use the **insert()** method:

Example

Insert an item as the second position:

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

Remove Item

- There are several methods to remove items from a list:
- Example

The **remove()** method removes the specified item:

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

The pop() method removes the specified index, (or the last item if index is not specified):

```
Thislist=["apple", "banana", "cherry"]

thislist.pop()

print(thislist)
```

del

he del keyword removes the specified index:

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

clear()

method empties the list:

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

thislist.clear()

print(thislist)
```

list() constructor

- Using the **list() constructor** to make a List:
- **thislist = list(("apple", "banana", "cherry"))** # note the double round-brackets
  print(thislist)

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable) , to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python List sort() Method

- Sort the list alphabetically:

```
cars = ['Ford', 'BMW', 'Volvo']
cars.sort()
```

- Sort the list descending:

```
cars = ['Ford', 'BMW', 'Volvo']
cars.sort(reverse=True)
```

- Sort the list by the length of the values:
- # A function that returns the length of the value:

```
def myFunc(e):
  return len(e)

cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']

cars.sort(key=myFunc)
```

- Sort a list of dictionaries based on the "year" value of the dictionaries:
- # A function that returns the 'year' value:
-
```
def myFunc(e):
 return e['year']

cars = [
  {'car': 'Ford', 'year': 2005},
  {'car': 'Mitsubishi', 'year': 2000},
  {'car': 'BMW', 'year': 2019},
  {'car': 'VW', 'year': 2011}
]

cars.sort(key=myFunc)
```

- *Sort the list by the length of the values and reversed*:
- # A function that returns the length of the value:

```
def myFunc(e):
  return len(e)

cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']

cars.sort(reverse=True, key=myFunc)
```

- Python **List index()** Method
- What is the position of the value "cherry":

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

x = fruits.index("cherry")
```

Python **List count()** Method

- Return the number of times the value " cherry" appears int the fruits list:
- **fruits = ['apple', 'banana', 'cherry']**

  x = fruits.count("cherry")

Python **List extend()** Method

- Add the elements of cars to the fruits list:
- **fruits = ['apple', 'banana', 'cherry']**

  cars = ['Ford', 'BMW', 'Volvo']

  fruits.extend(cars

- Add a tuple to the fruits list:
- **fruits = ['apple', 'banana', 'cherry']**

  points = (1, 4, 5, 9)

  fruits.extend(points)

Python List **reverse()** Method

- Example
- Reverse the order of the fruit list:
- **fruits = ['apple', 'banana', 'cherry']**

  fruits.reverse()

Python List copy() Method

- Example
- Copy the fruits list:
- **fruits = ['apple', 'banana', 'cherry', 'orange']**

  x = fruits.copy()

Python Tuples

Tuple

- A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round brackets.
- Example
- Create a Tuple:
- **thistuple = ("apple", "banana", "cherry")
  print(thistuple)**

Access Tuple Items

- You can access tuple items by referring to the index number, inside square brackets:
- Example
- Return the item in position 1:
- **thistuple = ("apple", "banana", "cherry")
print(thistuple[1])**

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**.

Example

You cannot change values in a tuple:

```
thistuple = ("apple", "banana", "cherry")
thistuple[1] = "blackcurrant"
# The values will remain the same:
print(thistuple)
```

Loop Through a Tuple

- You can loop through the tuple items by using a for loop.
- Example
- Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Check if Item Exists

- Check if "apple" is present in the tuple:

-
  ```
  thistuple = ("apple", "banana", "cherry")
  if "apple" in thistuple:
    print("Yes, 'apple' is in the fruits tuple")
  ```

Tuple Length

- To determine how many items a tuple has, use the **len()** method:
- Example
- Print the number of items in the tuple:
- **thistuple = ("apple", "banana", "cherry")
  print(len(thistuple))**

Add Items

- Once a tuple is created, you cannot add items to it. Tuples are **unchangeable**.
- Example
- You cannot add items to a tuple:
- **thistuple = ("apple", "banana", "cherry")**
  thistuple[3] = "orange" # This will raise an error
  print(thistuple)

Remove Items

- Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely:
- Example
- The del keyword can delete the tuple completely:
- **thistuple = ("apple", "banana", "cherry")**
  del thistuple
  print(thistuple) #this will raise an error because the tuple no longer exists

The tuple() Constructor

- It is also possible to use the tuple() constructor to make a tuple.
- Example
- Using the tuple() method to make a tuple:
- **thistuple = tuple(("apple", "banana", "cherry"))**

  print(thistuple)

Python Sets

Set

- A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.
- Example
- Create a Set:
- **thisset = {"apple", "banana", "cherry"} print(thisset)**

Access Items

- You cannot access items in a set by referring to an index, since sets are unordered the items has no index.
- But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using thein keyword.
- Example
- Loop through the set, and print the values:
- **thisset = {"apple", "banana", "cherry"}**

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

Add Items

- To add one item to a set use the add() method.
- To add more than one item to a set use the update() method.
- Example
- Add an item to a set, using the add() method:
- **thisset = {"apple", "banana", "cherry"}**

  thisset.add("orange")

  print(thisset)

update()

- Example
- Add multiple items to a set, using the update() method:
- **thisset = {"apple", "banana", "cherry"}**
  thisset.update(["orange", "mango", "grapes"])
  print(thisset)

pop()

- Remove the last item by using the pop() method:
- **thisset = {"apple", "banana", "cherry"}**

  x = thisset.pop()

  print(x)

  print(thisset)

Python Set difference() Method

- Example
- Return a set that contains the items that only exist in set x, and not in set y:
- **x = {"apple", "banana", "cherry"}**
  y = {"google", "microsoft", "apple"}
  z = x.difference(y)
  print(z)

Remove Item

- Remove Item
- To remove an item in a set, use the **remove()**, or the **discard()** method.
- Example
- Remove "banana" by using the remove() method:
- **thisset = {"apple", "banana", "cherry"}**

```
thisset.remove("banana")
 thisset.discard("banana")
 print(thisset)
```

clear()

- The **clear()** method empties the set:
- **thisset = {"apple", "banana", "cherry"}**

  thisset.clear()

  print(thisset)

del

- The **del** keyword will delete the set completely:
- **thisset = {"apple", "banana", "cherry"}**

  del thisset

  print(thisset)

set()

- The set() Constructor
- It is also possible to use the set() constructor to make a set.
- Example
- Using the set() constructor to make a set:
- **thisset = set(("apple", "banana", "cherry"))**
  # note the double round-brackets
  print(thisset)

symmetric_difference() Method

- Example
- Return a set that contains all items from both sets, except items that are present in both sets:

-
  ```
  x = {"apple", "banana", "cherry"}
  y = {"google", "microsoft", "apple"}

  z = x.symmetric_difference(y)

  print(z)
  ```

Set symmetric_difference_update() Method

- Example
- Remove the items that are present in both sets, AND insert the items that is not present in both sets:
- **x = {"apple", "banana", "cherry"}**
  y = {"google", "microsoft", "apple"}

  x.symmetric_difference_update(y)

  print(x)

intersection() Method

- Example
- Return a set that contains the items that exist in both set x, and set y:
-
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.intersection(y)

print(z)
```

Set union() Method

Example

- Return a set that contains all items from both sets, duplicates are excluded:
- **x = {"apple", "banana", "cherry"}**
  y = {"google", "microsoft", "apple"}

  z = x.union(y)

  print(z

Set isdisjoint() Method

- Example
- Return True if no items in set x is present in set y:

-
  ```
  x = {"apple", "banana", "cherry"}
  y = {"google", "microsoft", "facebook"}

  z = x.isdisjoint(y)

  print(z)
  ```

Set issubset() Method

- Example
- Return True if all items set x are present in set y:
-
```
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b", "a"}

z = x.issubset(y)

print(z)
```

Set issuperset() Method

- Example
- Return True if all items set y are present in set x:

-
  ```
  x = {"f", "e", "d", "c", "b", "a"}
  y = {"a", "b", "c"}

  z = x.issuperset(y)

  print(z)
  ```

Python Dictionaries

Dictionary:

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Example

Create and print a dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Accessing Items

- You can access the items of a dictionary by referring to its key name, inside square brackets:

Example:

- Get the value of the "model" key:
- **x = thisdict["model"]**

get()

- There is also a method called **get()** that will give you the same result:
- Example
- Get the value of the "model" key:
- **x = thisdict.get("model")**

Change Values

You can change the value of a specific item by referring to its key name:

Example

Change the "year" to 2018:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

Loop Through a Dictionary

- Example
- Print all key names in the dictionary, one by one:
- for x in thisdict:
  print(x)

Example

- Print all *values* in the dictionary, one by one:
- for x in thisdict:
  print(thisdict[x])

- **Example**
- You can also use the values() function to return values of a dictionary:
- for x in thisdict.values():
  print(x)

Example

- Loop through both *keys* and *values*, by using the items() function:

-
  ```
  for x, y in thisdict.items():
    print(x, y)
  ```

Adding Items

- Adding an item to the dictionary is done by using a new index key and assigning a value to it:
- Example

-
  ```
  thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
  }
  thisdict["color"] = "red"
  print(thisdict)
  ```

Removing Items

Example

- The pop() method removes the item with the specified key name:

-
  ```
  thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
  }
  thisdict.pop("model")
  print(thisdict)
  ```

Example

- The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

-
  ```
  thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
  }
  thisdict.popitem()
  print(thisdict)
  ```

Python If ... Else

Python Conditions and If statements

Example

- If statement:
- a = 33
  b = 200
  if b > a:
    print("b is greater than a")

Indentation

- Example
- If statement, without indentation (will raise an error):
- a = 33
  b = 200
  if b > a:
  print("b is greater than a") # you will get an error

Elif

- The elif keyword is pythons way of saying " if the previous conditions were not true, then try this condition".
- Example
-
```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

Else

- The else keyword catches anything which isn't caught by the preceding conditions.
- Example
-
```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Short Hand If

- Example
- One line if statement:
- if a > b: print("a is greater than b")

Short Hand If ... Else

- If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example

- One line if else statement:
- print("A") if a > b else print("B")

Python While Loops

The while Loop

- With the while loop we can execute a set of statements as long as a condition is true.
- Example
- Print i as long as i is less than 6:
-
```
i = 1
while i < 6:
  print(i)
  i += 1
```

The break Statement

- With the break statement we can stop the loop even if the while condition is true:

Example

- Exit the loop when i is 3:

-
  ```
  i = 1
  while i < 6:
    print(i)
    if i == 3:
      break
    i += 1
  ```

The continue Statement

- With the continue statement we can stop the current iteration, and continue with the next:

Example

- Continue to the next iteration if i is 3:

-
  ```
  i = 0
  while i < 6:
    i += 1
    if i == 3:
      continue
    print(i)
  ```

Python For Loops

For Loops

Example

- Print each fruit in a fruit list:
-
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Looping Through a String

- Example
- Loop through the letters in the word "
  banana":
- for x in "banana":
    print(x)

Python Functions

Creating a Function

- In Python a function is defined using the def keyword:
- Example
- def my_function():
    print("Hello from a function")

Calling a Function

- To call a function, use the function name followed by parenthesis:
- Example
- def my_function():
    print("Hello from a function")

  my_function()

Parameters

Example

- def my_function(**fname**):
    print(fname + " Refsnes")

  my_function(**"Emil"**)
  my_function(**"Tobias"**)
  my_function(**"Linus"**)

Return Values

- To let a function return a value, use the return statement:
- Example
- def my_function(x):
    return 5 * x

  print(my_function(3))
  print(my_function(5))
  print(my_function(9))

Python Lambda

lambda function

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.
- Syntax
- lambda *arguments* : *expression*
- The expression is executed and the result is returned:
- Example
- A lambda function that adds 10 to the number passed in as an argument, and print the result:
- **x = lambda a : a + 10**
  print(x(5))

- Lambda functions can take any number of arguments:
- Example
- A lambda function that multiplies argument a with argument b and print the result:

- **x = lambda a, b : a * b**
  print(x(5, 6))
- **x = lambda a, b, c : a + b + c**
  print(x(5, 6, 2))

Python Classes and Objects

Create a Class

- To create a class, use the keyword class:
- Example
- Create a class named MyClass, with a property named x:
- **class MyClass:**

```
class MyClass:
  x = 5
```

Create Object

- Now we can use the class named myClass to create objects:
- Example
- Create an object named p1, and print the value of x:
- **class MyClass:**

```
class MyClass:
  x = 5

p1 = MyClass()
print(p1.x)
```

The __init__() Function

- Example
- Create a class named Person, use the __init__() function to assign values for name and age:
-
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Object Methods

- Objects can also contain methods. Methods in objects are functions that belongs to the object.
- Let us create a method in the Person class:
- Example
- Insert a function that prints a greeting, and execute it on the p1 object:

-
  ```
  class Person:
    def __init__(self, name, age):
      self.name = name
      self.age = age

    def myfunc(self):
      print("Hello my name is " + self.name)

  p1 = Person("John", 36)
  p1.myfunc()
  ```

Modify Object Properties

- You can modify properties on objects like this:
- Example
- Set the age of p1 to 40:
- **p1.age = 40**

Delete Object Properties

- You can delete properties on objects by using the del keyword:
- Example
- Delete the age property from the p1 object:
- **del p1.age**

Delete Object :

del p1

Python - Files I/O

Opening and Closing Files

Syntax

- file object = open(file_name [, access_mode][, buffering])
- **file_name** – The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode** – The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- **buffering** – If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

FILE READ

- Modes & Description
- **r**

Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.

- **rb**
- Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
- **r+**
- Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
- **rb+**
- Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.

FILE WRITE

- **w**
- Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
- **wb**
- Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
- **w+**
- Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
- **wb+**
- Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

FILE APPENDING

- **a**
- Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
- **ab**
- Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
- **a+**
- Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
- **ab+**
- Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file doe

The *file* Object Attributes

- Once a file is opened and you have one *file* object, you can get various information related to that file.
- Here is a list of all attributes related to file object −
- Sr.No.Attribute & Description1**file.closed**
- Returns true if file is closed, false otherwise.
- 2**file.mode**
- Returns access mode with which file was opened.
- 3**file.name**
- Returns name of the file.
- 4**file.softspace**
- Returns false if space explicitly required with print, true otherwise

- The *read()* ,write(),append() Method

- Example
- Let's take a file *foo.txt*, which we created above.
- #!/usr/bin/python # Open a file
- **fo = open("foo.txt", "r")**
- **fo = open("foo.txt", “w")**
- **fo = open("foo.txt", “a")**

Renaming and Deleting Files

- Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.
- To use this module you need to import it first and then you can call any related functions.
- The rename() Method
- The *rename()* method takes two arguments, the current filename and the new filename.
- Syntax
- **os.rename(current_file_name, new_file_name)** Example
- Following is the example to rename an existing file *test1.txt* –
- #!/usr/bin/python import os # Rename a file from test1.txt to test2.txt os.rename("test1.txt", "test2.txt")
  The *remove()* Method
- You can use the *remove()* method to delete files by supplying the name of the file to be deleted as the argument.
- Syntax
- **os.remove(file_name)**

Directories in Python

- All files are contained within various directories, and Python has no problem handling these too.
  The **os** module has several methods that help you create, remove, and change directories.
- The *mkdir()* Method
- You can use the *mkdir()* method of the **os** module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.
- Syntax
- **os.mkdir("newdir")** Example
- Following is the example to create a directory *test* in the current directory −
- #!/usr/bin/python
- **import os   # Create a directory "test“**
- **os.mkdir("test")**

chdir() Method

- The *chdir()* Method
- You can use the *chdir()* method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.
- Syntax
- **os.chdir("newdir")** Example
- Following is the example to go into "/home/newdir" directory −
- #!/usr/bin/python
- **import os # Changing a directory to "/home/ newdir" os.chdir("/home/newdir")**

getcwd()

- The *getcwd()* Method
- The *getcwd()* method displays the current working directory.
- Syntax
- **os.getcwd()** Example
- Following is the example to give current directory −
- #!/usr/bin/python
- **import os** # This would give location of the current directory
- **os.getcwd()**

rmdir()

- The *rmdir()* Method
- The *rmdir()* method deletes the directory, which is passed as an argument in the method.
- Before removing a directory, all the contents in it should be removed.
- Syntax
- **os.rmdir('dirname')** Example
- Following is the example to remove "/tmp/test" directory. It is required to give fully qualified name of the directory, otherwise it would search for that directory in the current directory.
- #!/usr/bin/python
- **import os**
- # This would remove "/tmp/test" directory.
- **os.rmdir**("/tmp/test")

Python - Date & Time

Print time and date

- #!/usr/bin/python
- import time;
- localtime = time.asctime(time.
  localtime(time.time()))
- print ("Local current time :", localtime)