Date: / /

Python: python is a general purpose interpreted, interactive, object oriented, and high-level programming language.

It was created by Guido Van Rossum during 1985 - 1990.

Advantages

① python is interpreted - processed @ runtime. by interpreter. no need for compiling before execution.

② Interactive: can interact directly with a python interpreter via command prompt.

③ Object oriented:-

④ Python is a free language for both programmers and supports the development of a wide range of applications from simple text processing to web browsers & games.

papergrid

Date: / /

Naming conventions

Classes - start with uppercase & all other identifiers lowercase.

Starting an identifier with a single leading underscore indicates that the identifier is private.

two underscores leading an identifier clearly it's super private.

If the identifier also ends with two trailing underscores then it's a language defined special name

Lines & indentation

- python provides no braces to indicate blocks of code for class & function definitions or flow control.

- Blocks of code are denoted by line indentation, which is strictly enforced

```
if True:
    print("True")
else:
    print("False")
```

papergrid

Date: / /

→ check for python interpreter.
- cmd > python --version.

program to find largest to 3 no.

```
# numbers = []
for i in range(0,3):
    numbers.append(int(input("enter numbers:")))
print("largest of the 3 no. is:", max(numbers))
```

Variable names

a variable name can have short name (like x & y) or more descriptive name (age, carname etc).

Rules for variables:

- A variable name must start with a letter or the underscore character.

- A variable name cannot start with a no.

- Alphanumeric & '_' can be used.

- Variable names are case sensitive.

→ variables are containers for storing data values.
→ unlike other programming languages, Python has no command for declaring a var.
→ untyped. A variable is created the moment you first assign value to it.
Eg: x = 5
y = "John".

Data types.

str : String.
int, float, complex : numeric.
list, tuple, range (sequence).
dictionary : mapping type.
set, frozenset : Set type.
bool : boolean.
bytes, bytearray, memoryview : Binary types.

Python Strings
- Strings in python are identified as contiguous set of characters.
- represented in quote characters.
- python allows either pairs of single or double quotes.
- subsets of strings can be taken using slice operator ([] and [:])
- with indexes starting at 0 in the beginning of the string & working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator.
- and the asterisk (*) is the repetition operator.

Eg: ~~print~~ - str = "Hello world!"
- print (str)
- print (str[0])
- print (str[2:5])
- print (str[2:])
- print (str * 2)
- print (str + "TEST")

Python Lists

- Lists are the most versatile of python's compound data types.
- A list contains items separated by commas and enclosed within square [] brackets.
- To some extent, lists are similar to arrays in C.
- One difference between them is that all the items belonging to a list can be of different types.
- The value stored in a list can be accessed using the slice operator.
- With indexes starting at 0 in the beginning of the list and working their way to -1 at the end.
- The (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

```
list = ['abcd', 786, 'john', 70.2]
tinylist = [123, 'john']
print(list)
print(list[0])
print(list[1:3])
print(list[2:])
print(tinylist * 2)
print(list + tinylist)
```


Python tuples

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas.
- Unlike lists, however, tuples are enclosed within ( ).
- Unlike lists, tuples cannot be updated.
- Tuples can be thought of as a read only list.

Python dictionary

- Python's dictionaries are kind of hash table type.
- They consists of key-value pairs.
- A dictionary key can be almost any Python type, but are usually numbers or strings.
- Dictionaries are enclosed by curly braces { } and values can be assigned and accessed using square braces [ ]

```
dict = {'one': "This is one"}
dict[2] = "This is two"
print(dict['one'])
print(dict[2])
print(dict.keys())
print(dict.values())
```

Data type conversion.
- to convert b/w types, you simply use the type name as a function.
- There are several built in functions to perform conversions from one data type to another.
These functions return a new object representing the converted value.
- int(x[,base])
- long(x[,base])
- float(x)
- complex(real[,imag])
- str(x)
- repr(x) - equivalent to str()
- eval(str) - evaluates a string & returns an object.
- tuple(s) - converts to a tuple.
- list(s) - converts to a list.
- set(s) - converts to a set.
- dict(d) - creates a dictionary. d must be a sequence of (key, value) tuples.
- frozenset(s) - converts to a frozen set.

Python numbers
3 numeric types in python.
- int eg: a = 1
- float eg: y = 2.8
- complex eg: 3 = 1j

To verify any object in python we use the type() function

Python booleans
Booleans value is either true or false.

print(100 > 200)

O/P false.

The bool() function allows you to evaluate any value and give you true or false in return.

Python collections
There are four collection datatypes in python.

1) List - is a collection which is ordered & 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

Operators in python

① Arithmetic
+ , - , * , / , % , **.

② Assignment operators.
= , += , -= , *= , /= , %= ,
**= , //=

③ comparison operators.
== , != , < , > , <= , >=

④ Logical operators.
And, Or, not.

⑤ Identity operators.
- is
- returns true if both the variables are the same object.

Eg: x = 10
y = 20
z = x.

print(x is y) # false.
print(x is z) # True.

'is' operator compares the object references for equality whereas the comparison operator == checks for the equality of the values of the objects

'is not': returns true if both object references are different.

List

List is a collection which is ordered and changeable.

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

Access list items by refering to the index number:

```
print(thislist[1])
```

Negative indexing means beginning from end, -1 refers to the last item, -2 refers to the second last item etc.

```
print(thislist[-1])
```

Range of indexes: You can specify a range of indexes by specifying where to start and where to end the range.

```
print(thislist[2:5])
```

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

```
thislist[1] = "blackcurrant"
```

Loop through List: You can loop through list items by using a for loop:

```
for x in thislist:
    print(x)
```

Check if item exists:

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

List Length:
To determine how many items a list has the len() function:

```
print(len(thislist))
```

Add items: To add an item to the end of the list use the append() method:

```
thislist.append("orange")
```

To add item at specified index use insert.

```
thislist.insert(1, "orange")
```

Remove Items: There are several methods to remove items from a list:

```
# remove method removes specified item
thislist.remove("banana")
```

- The pop() method removes specified index,

thislist.pop()

- The del() keyword removes the specified index:

del thislist[0]

del thislist

- clear method empties the list:

thislist.clear()

- Copy a list:

copy() method.

mylist = thislist.copy()

- list() method

mylist = list(thislist)

Join two lists: There are several ways to join or concatenate two or more lists in python.

One of the easiest ways are by using + operator

list3 = list1 + list2

You can use extend method which purpose is to add elements from one list to another list:

list3 = list1.extend(list2)

List methods

~~append()~~ .

Tuples

is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

Eg: thistuple = ("apple", "banana", "cherry")

Access tuple items can be accessed by referring to the index number.

- print(thistuple[1])

Negative indexing.

- print(thistuple[-1])

Range of indexes.

print(thistuple[2:5])

Change tuple values
- a tuple is normally unchangable, but there is a work around.
- convert the tuple to a list using the list() function

Loop through tuple.
- for x in thistuple:
  print(x)

Check if item exists
if "apple" in thistuple:
  print("yes "apple" is in the fruits tuple").

Check if item exits.
- print(len(thistuple))

Delete tuple
- del(thistuple)

Join Loops
tuple3 = tuple1 + tuple2

Python Sets
Set is a collection which is unordered and unindexed. In python sets are written with curly brackets.
- thisset = {"apple", "banana", "cherry"}

Always item: since sets are unordered. items cannot be referred by its index.

Use the for-in loop.
- for x in thisset:
  print(x)

Change Items: once a set is created you cannot change items. but you can add new items

Add items: to add one item to a set use add() method.
to add more than one item to a set use update() method

- thisset.add("orange")
- thisset.update(["apple", "mango", "grapes"])

Length of a Set.

- print(len(thisset))

Remove Item: to remove an item in a set use the remove() or the discard() method.

- thisset.remove("banana")

> if item to be removed doesn't exist, remove() will raise an error.

- thisset.discard("banana")

> discard() will not raise an error in case of missing item.

- thisset.clear()
  # empties the set.
- del(thisset)

Join two Sets.

The union() method returns a new set with all items from both sets.

- set3 = set1.union(set2)

The update() method inserts the items in set2 into set1:

set1.update(set2)

* Python dictionaries

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

thisdict = {"brand": "ford", "model": "mustang", "year": 1964}

Access Items: you can access the items of a dictionary by refering to its key name, inside square brackets.

ex: thisdict["model"]

There is also a get() method that will give you the same result.
x = thisdict.get("model")

Change values
- thisdict["year"] = 2018

Loop through a dictionary.

while loops on a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

- # keys.
for x in thisdict:
print(x)

- # values
for x in thisdict:
print(thisdict[x])

- for x in thisdict.values():
print(x)

key value pair.
- for x,y in this dict.items():
print(x, y)

Check if key exist in dictionary
- if "model" in thisdict:
print("yes", 'model' is one of the key in the dictionary).

Dictionary length
- print(len(thisdict))

adding items
- thisdict["color"] = red

removing items
- thisdict.pop("model")

popitem() method removes the last inserted item.

- thisdict.popitem().
- del thisdict["model"]

* clear() method empties the dictionary.
  thisdict.clear()

Copy a dictionary.

- mydict = thisdict.copy()
- mydict = dict(thisdict)

Nested dictionaries.
  a dictionary can also contain many dictionaries, this is called nested dictionaries.

Eg: myfamily = {"child1": {
                "name": "Emil",
                "year": 2004
            },
            "child2": {
                "name": "Tobias",
                "year": 2007}
            }

python if ... else.

python conditional statements

python support the usual logical conditions from mathematic
equals: a == b
not equals: a != b
less than: a < b
less or equal to: a <= b.
greater than: a > b.
greater than or equal to: a >= b.

These conditions can be used in several ways, most commonly in "if statements" and loops.

and "if statements" is written by using the if keyword.

```
a = 33
b = 200.
if b > a:
    print("b is greater than a")
```

Indentation.

python relies on indentation to define scope in the code. Other programming languages often use curly brackets for this purpose.

Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

```
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.

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a is equal to b")
else:
    print("a is greater than b")
```


Short hand if

If you have only one statement to execute, you can put it on the same line as the 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.

```
a = 2
b = 330
print("A") if a > b else print("B")
```

AND, OR

AND

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

OR

```
a = 200
b = 33
c = 500
if a > b or a > c: print("At least one of the conditions is True")
```

Nested If.
you can have if statements inside if statements, this is called nested if statements.

```
x = 41
if a > 10:
    print ("above ten,")
    if x > 2:
        print("and also above 20!")
    else
        print ("but not above 20.")
```

The pass statement
if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

```
a = 33
b = 200
if b > a:
    pass
```


python while loops
python loops:
python has 2 primitive loop commands:
- while loops
- for loops

The while loop:
with the while loop we can execute a set of statements as long as a condition is true.

Eg:

```
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:

```
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.

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

The else statement.

with the else statement we can run a block of code once when the condition no longer is true:

```
- i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")
```

Python For Loops

A for loop is used for iterating over a sequence.

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-oriented programming languages.

with the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

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

Looping through a String

Even strings are iterable objects, they contain a sequence of characters.

```
- for x in "banana":
      print(x)
```

The break statement.

with the break statement we can stop the loop before it has looped through all the items:

```
- fruits = ["apple", "banana", "cherry"]
  for x in fruits:
      print(x)
      if x == "banana":
          break
```

The continue statement.

with the continue statement we can stop the current iteration of the loop and continue with the next:

- fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        continue
    print(x)

The range() function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default) and ends at a specified number.

for x in range(6):
    print(x)

Python functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

In a python function is defined using the def keyword:

eg: def my_function():
        print("Hello from a function")

Calling a function.

To call a function, use the function name followed by parenthesis:

def my_function():
    print("Hello from a function")

my_function()

~~Argum~~

Arguments

Information can be passed into functions as arguments.

arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate with a comma.

The following example has a function with one argument (fname). When the function is called we pass along a first name, which is used inside the function to print the full name:

```
def my_function(fname):
    print(fname + " Refsnes").
my_function("Emil")
my_function("Tobias")
```


Parameters or arguments?

The terms parameter and argument can be used for the same thing: information that are passed into a function.

From a function's perspective:

A parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that is sent to the function when it's called.

Number of arguments.

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more and not less.

```
def my_function(fname, lname):
    print(fname + " " + lname)
my_function("Emil", "Refsnes")
```

Arbitary arguments, * args.

If you dont know how many arguments that will be passed into your function add a * before the parameter name in the function definition.

eg: def my_funtion (*kids)
print ("The youngest child is" + kids[2])
my_function ("Emil", "Tobias", "Linus")

Keyword arguments

You can also send arguments with key = value syntax.

This way the order of the arguments does not matter.

def my_function (child3, child2, child1)
print ("The youngest child is" + child3)

my_function (child1 = "Emil", child2 = "Tobias",
child3 = "Linus")

Arbitary keyword arguments,

If you dont know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

def my_function(** kid):
print ("His last name is" + kid["lname"])

my_function (fname = "Tobias", lname = "Refsnes")

Default parameter value

The following example shows how to use a default parameter value.

eg: def my_function(country = "Norway"):
print ("I am from" + country)

my_function ("Sweden")
my_function ()
my_function ("Brazil")

Passing a list as an argument.

You can send any types of arguments to a function, and it will be treated as the same data type inside the function.

Eg:

```
def my_function(food):
    for x in food:
        print(x)

fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

Return Values

To let a function return a value, use the return statement:

Eg:

```
def my_function(x):
    return 5 * x

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


The pass Statement.

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

Eg:

```
def myfunction():
    pass
```

Recursion

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

Python strings

String literals.

String literals in python are surrounded by either single quotation marks, or double marks.

'hello' is the same as "hello".

you can display a string literal with the print() function:

```
print("hello")
print('hello')
```

Assign string to a variable.

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

```
a = "hello"
print(a)
```

Strings are arrays

Like many other popular programming languages, strings in python is an array of bytes representing unicode characters.

However, python does not have a character data type, a single char is the same as a string with length of 1. Square brackets can be used to access elements of the string.

```
a = "hello world!"
print(a[1])
```

Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

```
eg: b = "hello, world!"
    print(b[2:5])
```

negative indexing.

Use negative indexes to start the slice from the end of the string:

```
b = "hello, world!"
print(b[-5:-2])
```

String length.

To get the length of a string, use the len() function.

eg: The len() function returns the length of a string

```
a = "hello, world!"
print(len(a))
```

String methods

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

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

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

Eg: upper method returns the string in upper case:

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

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

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

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

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


String concatenation

To concatenate, or combine, two strings you can use the + operator.

Eg: Merge variable a with variable b into variable c.

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

String Format

As we learned in Python variables chapter, we cannot combine strings and numbers like this:

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

But we can combine strings and numbers by using the format() method!

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

Eg:
age = 36
text = "my name is John and I am {}"
print(text.format(age))

=> The format method takes unlimited no. of arguments and are placed into the respective placeholders!

Eg:
quantity = 3
itemno = 567
price = 49.95
my order = "I would like {} pieces of item {} for {} dollars."

print(myorder.format(quantity, itemno, price))

=> You can use index numbers {0} to be sure the arguments are placed in the correct placeholders.
qty = 3
itemno = 567
price = 49.95
myorder = "I would like to buy {2} dollars for {0} pieces of item {1}."

print(myorder.format(qty, itemno, price))

Escape character
To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a " inside a string that is surrounded by double quotes:

txt = "We are the so-called "Vikings" from the north."

To fix this problem, use the escape character \":

txt = "We are the so-called \"Vikings\" from the north."

Code	Result
\'	Single quote
\\	Backslash
\n	New Line
\r	Carriage return
\t	Tab
\b	Backspace
\f	Form Feed

Code	Result
\ooo	Octal value
\xhh	Hex value

papergrid

Date: / /

Python file handling.

File handling is an important part of any application

Python has several functions for, creating, reading updating & deleting files.

File handling.

The key function for working with files in Python is open() function.
The open() function takes two parameters; filename & mode.
There are four different methods for opening a file.

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist.

"w" - Write - opens a file for writing, creates file if it does not exist

papergrid

Date: / /

"x" - Create - Creates the specified file, returns an error if the file does exist.

In addition you can specify if the file should be handled as binary or text mode.

"t" - Text - Default value. Text mode.
"b" - Binary - Binary mode (e.g. images).

Syntax: To open a file for reading it is enough to specify the name of the file:

f = open("demofile.txt")

The code above is same as:

f = open("demofile.txt", "rt")

Because 'r' for read, and 't' for text are the default values, you only need to specify them.

Assume we are working file is in some folder as python:

demofile.txt
Hello! welcome to demofile.txt.
This file is for testing purpose.
Good luck.

Eg: f = open("demofile.txt", "r")
print(f.read())

if file is in diff location:

f = open("D:\\myfolder\\welcome.txt", "r")
print(f.read())

* By default the read() method returns the whole text, but you can also specify how many characters you want to return:

Eg:
f = open("demofile.txt", "r")
print(f.read(5))

Readlines

By calling the readlines() function you can read the two first lines:

Ex:
f = open("demofile.txt", "r")
print(f.readlines())
print(f.readline())

By looping through the lines of the file you can read the whole file, line by line.

Ex: f = open("demofile.txt", "r")
for x in f:
print(x)

Close files

It is a good practice to always close the file when you are done with it.

Ex:
f = open("demofile.txt", "r")
print(f.readline())
f.close()

Python file write.

write to an existing file.

To write to an exist file you must add a parameter to the open() function:

'a' - append
'w' - write.

Ex:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()
```

Ex:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()
```

Create a new file.

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create.
"a" - Append
"w" - write.

Ex:

```
f = open("myfile.txt", "x")
```

Python delete file

Delete a file.

To delete a file you must import the OS module, and run its os.remove() function:

Ex:

```
import os
os.remove("demofile.txt")
```

check if file exist:

```
import os.
if os.path.exists("demofile.txt"):
    os.remove("demofile.txt")
else:
    print("The file does not exist")
```

delete folder.

To delete an entire folder, use the os.rmdir() method:

```
import os
os.rmdir("myfolder")
```