INTRODUCTION TO PYTHON

Definition of Python

* Python is a high level, general – purpose programming language with an elegant syntax that allows programmers to focus more on problem- solving than on syntax errors.
* Python has become a big buzz in the field of modern software development infrastructure management and especially in data science and artificial intelligence .

Python advantages over other languages:

* When compared to any other high level programming language such as c++ , java, it requires the program to develop the source codes
* Python is designed to be simple and easy to read and write. Python can be utilized for a wide range of applications such as scripting, developing, & testing.
* Python can be easily compiles to byte code
* Python also provides features to support functional and structured programming
* Python supports an interactive mode which allows interactive testing and debugging for snippets of code.
* In python there are no editing, debugging testing compilation steps, so it is very fast.

Features of python:

* It is simple to use
* It as elegant syntax and readability
* It has large library
* It has a cross- platform
* It has a free and open source program language.
* Python programing code are line by line at a time using the interpreter
* Object- oriented programing can be achieved through python class
* It can be integrated with other programing languages like c, C++, java, and many more.
* This will teach you many more porogrm concepts

Applications of Python:

* Web applications
* Creating software proto types
* Scientific & Numeric Computing
* Network programming
* Games and 3D Applications
* A good language to teach programming o freshers.

Characterstics of Python:

* There is a rich collection of python data types.
* Python is a platform independent – scripted language
* Python provides more run- time flexibility.
* Libraries available in python are cross- platform Compatible.

**A Simple Python Cheat Sheet**

note : I'm sorry if there are mistakes in words and meaning, hopefully you guys can understand it, thanks in advance !!!

Important methods in Python:

SET:

* Add()
* Clear()
* Pop()
* Union()
* Issuperset()
* Issubset
* Intersection()
* Difference()
* Isdisjoint()
* Setdiscard()
* Copy()

LIST;

* Append)
* Copy()
* Count()
* Insert()
* Reverse()
* Remove()
* Sort()
* Pop()
* Extend()
* Index()
* Clear()

DICTIONARY:

* Copy()
* Clear()
* fromkeys()
* items()
* get()
* keys()
* pop()
* values()
* update()
* setdefault()
* popitem()

**Table Of Contents**

* Main Python Data Types
* Built-in Function in Python
* Math Operator
* List
* Tuples
* Dictionaries
* Define a Function
* If statements in Python
* Python Loops
* Class
* Dealing with Python Excpetions (Errors)

**Main Python Data Types**

Every value in python is called an **object**. And every object has a spesific data type. The three most-used data types are as follows:

**1. Integers (int)**

In Python, integers are zero, positive or negative whole numbers without a fractional part and having unlimited precision, e.g. 0, 100, -10. The followings are valid integer literals in Python. Integers can be binary, octal, and hexadecimal values. All integer literals or variables are objects of the int class.

**Example :**

In [27]:

*# Zero, Negative, and Positive*

number\_1, number\_2, number\_3 **=** 0, **-**2, 50

*# Binary, Octal, and Hexadesimal*

number\_4, number\_5, number\_6 **=** 0b11011000, 0o12, 0x12

print(type(number\_1), type(number\_2), type(number\_3))

print(type(number\_4), type(number\_5), type(number\_6))

<class 'int'> <class 'int'> <class 'int'>

<class 'int'> <class 'int'> <class 'int'>

**2. Floating-point numbers (float)**

In Python, floating point numbers (float) are positive and negative real numbers with a fractional part denoted by the decimal symbol . or the scientific notation E or e, e.g. 1234.56, 3.142, -1.55, 0.23.

**Example :**

In [28]:

*# Float 1, 2, 3*

number\_1, number\_2, number\_3 **=** 0.0, **-**2.0, 50.87

*# Float with many number, Float with underscore, and Scientific notation*

number\_4, number\_5, number\_6 **=** 123.123, 123\_123.3221, 1e5

print(type(number\_1), type(number\_2), type(number\_3))

print(type(number\_4), type(number\_5), type(number\_6))

<class 'float'> <class 'float'> <class 'float'>

<class 'float'> <class 'float'> <class 'float'>

**3. Strings (str)**

In Python, string is an immutable sequence data type. It is the sequence of Unicode characters wrapped inside single, double, or triple quotes.

**Example :**

In [29]:

str\_1**=**'This is a string in Python' *# With single quote*

str\_2**=**"This is a string in Python" *# With double quote*

*# Multiline String with single quote*

str\_3**=**'''This is

the first

Multi-line string.

'''

*# Multiline String with double quote*

str\_4**=**"""This is

the second

Multi-line

string."""

print(type(str\_1), type(str\_2))

print(type(str\_3), type(str\_4))

<class 'str'> <class 'str'>

<class 'str'> <class 'str'>

**Built-in functions in Python**

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order. You already know the most popular function in Python, yap you're right, is a **print()** function. Now let's take a look at its equally popular cousins that are in-built in the platform, I'll not showing you guys all of this built in function, if you want to know more, I'll give you the link below.

Here's the link : <https://docs.python.org/3/library/functions.html>

**1. input() Function**

Allows user to input values.

**Example :**

In [30]:

name **=** input("Hi!, what's your name? ")

print("Nice to meet you " **+** name **+** "!")

age **=** input("How old are you? ")

print("So you already " **+** str(age) **+** "years old, that's nice!")

Nice to meet you Azmi!

So you already 18years old, that's nice!

**2. len() Function**

Returns the length of the object. Returns total elements in an iterable or number of chars in a string.

**Example :**

In [31]:

len\_1 **=** "Hope you are enjoying mu Cheat-Sheet!"

print("The length of the string is :", len(len\_1))

The length of the string is : 37

**3. max() and min() Function**

* max() : Returns the largest value from the specified iterable or multiple arguments.
* min() : Returns the lowest value from the specified iterable or provided multiple arguments.

**Example :**

In [32]:

my\_max\_num **=** [5, 10, 40]

print(f"My max num is: {max(my\_max\_num)}")

my\_min\_num **=** [5, 10, 40]

print(f"My min num is: {min(my\_max\_num)}")

My max num is: 40

My min num is: 5

**Math Operator**

Arithmetic operators are used to perform mathematical operations like **addition**, **subtraction**, **multiplication** and **division**.

There are **7 arithmetic operators in Python** :

1. **Addition**
2. **Subtraction**
3. **Multiplication**
4. **Division**
5. **Modulus**
6. **Exponentiation**
7. **Floor division**

* **Addition Operator** : In Python, + is the addition operator. It is used to add 2 values or more.
* **Subtraction Operator** : In Python, - is the subtraction operator. It is used to subtract 2 values or more.
* **Multiplication Operator** : In python, to multiply number, we will use the asterisk character ” \* ” to multiply number.
* **Division Operator** : Divides the number on its left by the number on its right and returns a floating point value.
* **Modulus Operator** : The % symbol in Python is called the Modulo Operator. It returns the remainder of dividing the left hand operand by right hand operand.
* **Exponentation Operator** : One important basic arithmetic operator, in Python, is the exponent operator. It takes in two real numbers as input arguments and returns a single number. The operator that can be used to perform the exponent arithmetic in Python is \*\* .
* **Floor Division Operator** : Divides the number on its left by the number on its right, rounds down the answer, and returns a whole number.

**Example :**

In [33]:

*# (\*\*) = Exponent*

print('Exponent')

print(2 **\*\*** 4)

*# (%) = Modulus/Remainder*

print('Modulus/Remainder')

print(25 **%** 3)

*# (//) = Floor Division*

print('Floor Division')

print(25 **//** 3)

*# (/) = Division*

print('Division')

print(25 **/** 3)

*# (\*) = Multiplication*

print('Multiplication')

print(4 **\*** 9)

*# (-) = Subtraction*

print('Subtraction')

print(5 **-** 4)

*# (+) = Addition*

print('Addition')

print(9 **+** 4)

Exponent

16

Modulus/Remainder

1

Floor Division

8

Division

8.333333333333334

Multiplication

36

Subtraction

1

Addition

13

**List**

A **list** is an ordered and mutable Python container, being one of the most common data structures in Python. To create a **list**, the elements are placed inside square brackets **([])**, separated by commas. As shown above, **lists** can contain elements of different types as well as duplicated elements.

**Example :**

In [34]:

my\_list **=** [1, 2, 3]

my\_list2 **=** ['a', "b", 'c']

my\_list3 **=** ['a', 1, **True**, 0.5]

print(my\_list)

print(my\_list2)

print(my\_list3)

[1, 2, 3]

['a', 'b', 'c']

['a', 1, True, 0.5]

* Use **append()** function if you want to add a new items to list.
* Use **remove()** function if you want to remove an items on a list.
* Or you can use **pop()** function, it will remove the last item, if, no index specified.

For more information about list, you can check this python site : <https://docs.python.org/3/tutorial/datastructures.html>

**Tuple**

Python **tuples** are a data structure that store an ordered sequence of values. **Tuples are immutable**. This means you cannot change the values in a **tuple**. Tuples are similar to lists, they allow you to display an ordered sequence of elements. The advantage of using tuples over lists is that the former are slightly faster. So it's a nice way to optimize your code.

**Example :**

In [35]:

my\_tuple **=** (1, 2, 3, 4, 5)

my\_tuple[0:3] *# You can slice a tuple*

Out[35]:

(1, 2, 3)

**How to Slide a Tuple**

**The process is similar to slicing lists.**

In [36]:

numbers **=** (0, 1, 3, 4, 5, 6, 1, 4, 2, 4, 5, 6, 8, 1)

print(numbers[1:11:2])

(1, 4, 6, 4, 4)

For more infomation about how to use Tuple, here's the link : <https://docs.python.org/3/tutorial/datastructures.html>

**Dictionary**

**Dictionary** in Python is an unordered collection of data values, used to store data values like a map, which, unlike other Data Types that hold only a single value as an element, **Dictionary** holds **key:value** pair. **Key-value** is provided in the **dictionary** to make it more optimized.

**Example :**

In [37]:

new\_dict **=** {

'Name': 'Knightbearr',

'Age': 18,

'Hobbies': 'Read Books' *# Actually nah XD*

}

print(new\_dict)

{'Name': 'Knightbearr', 'Age': 18, 'Hobbies': 'Read Books'}

**How to Acces a Value in Dictionary**

**You can acces any of the values in your dictionary the following way :**

In [38]:

x **=** new\_dict['Name']

**You can also use the following methods to accomplish the same.**

* **dict.keys()** isolates keys
* **dict.values()** isolates values
* **dict.items()** returns items in a list format of (key, value) yuple pairs

**Change item Value**

**To change one of the items, you need to refer to by its key name:**

In [39]:

*# Change the "Hobbies" to Jogging*

new\_dict **=** {

'Name': 'Knightbearr',

'Age': 18,

'Hobbies': 'Read Books' *# Actually nah XD*

}

new\_dict['Hobbies'] **=** 'Jogging'

print(new\_dict)

{'Name': 'Knightbearr', 'Age': 18, 'Hobbies': 'Jogging'}

For more infomation about how to use Dictionary, here's the link : <https://docs.python.org/3/tutorial/datastructures.html>

**Define a Function**

A Python **function** is a group of code. To run the code in a **function**, you must call the **function**. **Functions** can return a value using a return statement. **Functions** are a common feature among all programming languages. They allow developers to write blocks of code that perform specific tasks.

**Example :**

In [40]:

**def** name():

**pass** *# pass is a null statement, nothing happens and the statement results into no operation.*

In [41]:

**def** name():

print("What's your name?") *# Now you can call/invoke the function*

name()

What's your name?

**Now, let's take a look at a defined function with a parameter.**

Let's make function to calculate the perimeter and area of a square.

In [42]:

**def** square(width, length):

around **=** 2 **\*** (width **+** length)

large **=** (width **\*** length)

**return** around, large

around\_result, large\_result **=** square(10, 20)

print(f'Around result: {around\_result}, Large result: {large\_result}')

Around result: 60, Large result: 200

For more information about Python Function, here's the link : <https://www.w3schools.com/python/python_functions.asp>

**If Statements In Python**

Conditional Statement in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false. Conditional statements are handled by IF statements in Python.

Just like other programming languages, Python support the basic logical conditions from math:

* **Equals**: a **==** b
* **Not Equals:** a **!=** b
* **Less than:** a **<** b
* **Less than or equal to:** a **<=** b
* **Greater than:** a **>** b
* **Greater than or equal to:** a **>=** b

You can leverage these conditions in various ways. But most likely, you'll use them in **"if statements"** and **loops**

**Example :**

In [43]:

*# The goal of conditional statements is to check if it's True or False.*

**if** 100 **>** 1:

print("That's true")

That's true

**Nested If Statements**

In [44]:

x **=** 40

**if** x **>** 20:

print('Above twenty')

**if** x **>** 30:

print('Also above 30!')

Above twenty

Also above 30!

For more informations about If statements, here's the link : <https://www.w3schools.com/python/python_conditions.asp>

**Python Loops**

Python has two simple loop commands that are good to know:

* For loops
* While loops

Let's take a look at each of these.

**1. For Loops**

For-loop is a control flow statement for specifying iteration, which allows code to be executed repeatedly. Various keywords are used to specify this statement: descendants of ALGOL use "for", while descendants of Fortran use "do".

In [45]:

**for** i **in** 'nikola tesla':

print(i)

n

i

k

o

l

a

t

e

s

l

a

**2. While loops**

Python while Loop Statements, A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is True.

In [46]:

i **=** 0

**while** i **<** 5:

print(i)

i **+=** 1

0

1

2

3

4

**How to break a loop**

In [47]:

i **=** 0

**while** i **<** 5:

**if** i **==** 3:

**break**

print(i)

i **+=** 1

0

1

2

For more information about Python loops, here's the link:

* While Loops : <https://www.w3schools.com/python/python_while_loops.asp>
* For Loops : <https://www.w3schools.com/python/python_for_loops.asp>

**Class**

A Python class is like an outline for creating a new object. An object is anything that you wish to manipulate or change while working through the code. Every time a class object is instantiated, which is when we declare a variable, a new object is initiated from scratch.

**Example :**

**How to Create a Class**

Let's create a class named superClass, with one property named s

In [48]:

**class** superClass:

s **=** 100

**How to Create an Object**

As a next step, you can create an object using your class, Here's how it's done:

In [49]:

c1 **=** superClass()

print(c1**.**s)

100

**Further you can assign different attributes and methods to your object. The example below:**

In [50]:

*# Class and Object 1*

**class** Car:

carCount **=** 0

**def** \_\_init\_\_(self, name, color):

self**.**name **=** name

self**.**color **=** color

Car**.**carCount **+=** 1

**def** getCar(self):

**return** "Name : " **+** self**.**name **+** ", Color : " **+** self**.**color

ford1 **=** Car("Ford", "Red")

toyota1 **=** Car("Toyota", "Green")

print(f"Total of Car : {Car**.**carCount}")

print("Car 1 : \n" **+** ford1**.**getCar())

print("Car 2 : \n" **+** toyota1**.**getCar())

*# Class and Object 2*

**class** Person:

**def** \_\_init\_\_(self, name, age):

self**.**name **=** name

self**.**age **=** age

**def** getIdentitas(self):

**return** "Name : " **+** self**.**name **+** ", age : " **+** str(self**.**age)

azmi **=** Person('Azmi', 18)

fujimoto **=** Person('Fujimoto', 20)

print(azmi**.**getIdentitas())

print(fujimoto**.**getIdentitas())

Total of Car : 2

Car 1 :

Name : Ford, Color : Red

Car 2 :

Name : Toyota, Color : Green

Name : Azmi, age : 18

Name : Fujimoto, age : 20

Fo more information about the Python Class, here's the link: <https://docs.python.org/3/tutorial/classes.html>

**Dealing with Python Exceptions (Error)**

Until now error messages haven’t been more than mentioned, but if you have tried out the examples you have probably seen some. There are (at least) two distinguishable kinds of errors: syntax errors and exceptions.

**The Most Common Python Exceptions:**

* **AssertionError** = Raised when the assert statement fails.
* **AttributeError** = Raised on the attribute assignment or reference fails.
* **EOFError** = Raised when the input() function hits the end-of-file condition.
* **FloatingPointError** = Raised when a floating point operation fails.
* **ImportError** = Raised when the imported module is not found.
* **IndexError** = Raised when the index of a sequence is out of range.
* **KeyError** = Raised when a key is not found in a dictionary.
* **KeyboardInterrupt** = Raised when the user hits the interrupt key (Ctrl+c or delete).
* **MemoryError** = Raised when an operation runs out of memory.
* **NameError** = Raised when a variable is not found in the local or global scope.
* **RuntimeError** = Raised when an error does not fall under any other category.
* **StopIteration** = Raised by the next() function to indicate that there is no further item to be returned by the iterator.
* **SyntaxError** = Raised by the parser when a syntax error is encountered.
* **IndentationError** = Raised when there is an incorrect indentation.
* **TabError** = Raised when the indentation consists of inconsistent tabs and spaces.
* **TypeError** = Raised when a function or operation is applied to an object of an incorrect type.
* **UnboundLocalError** = Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
* **UnicodeError** = Raised when a Unicode-related encoding or decoding error occurs.
* **ValueError** = Raised when a function gets an argument of correct type but improper value.
* **ZeroDivisionError** = Raised when the second operand of a division or module operation is zero.

If you want to know more about Error in Python, here's the link :

* <https://www.tutorialsteacher.com/python/error-types-in-python>
* <https://docs.python.org/3/tutorial/errors.html>

**How to Troubleshot the Errors**

Python has a useful statement, design just for the purpose of handling exceptions, **try/except** statement, Here's the code showing you can catch KeyErrors in dictionary using this statement:

**1. Key Error Example**

In [51]:

my\_dict **=** {'a':1, 'b':2, 'c':3}

**try**:

value **=** my\_dict['d']

**except** KeyError:

print("That key doesn't exist!")

That key doesn't exist!

**2. Value Error and Zero Division Error**

In [52]:

print("Division")

**while** **True**:

**try**:

denominator **=** int(input("Input the denominator: "))

numerator **=** int(input("Input the numerator: "))

result **=** denominator**/**numerator

**break**

**except** ValueError:

print("Is not a number")

**except** ZeroDivisionError:

print("The Numerator is Zero, Please choose another number ^-^")

Division