PYTHON

Vikas Jaat

INTRODUCTION

AGENDA-WW01D01

- What is python
- Why Python
- Python Use cases
- Python Installation
- VS and Jupyter Notebook Setup
- Executing First Python Script

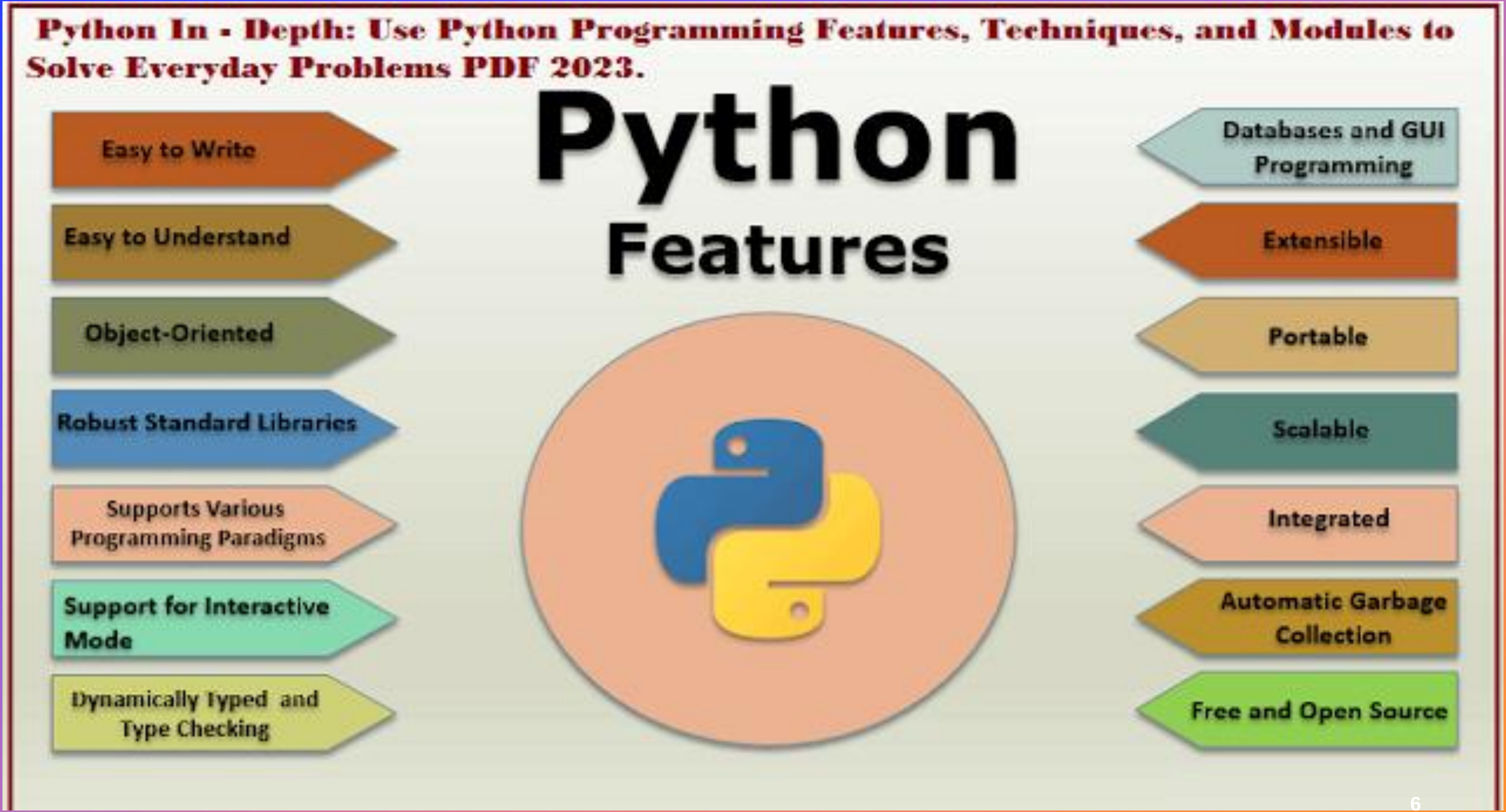
WHAT IS PYTHON

WHY PYTHON

- Python is a general-purpose, interpreted, object-oriented, high-level programming language.
- It is an open-source programming language with more than 1 lakh libraries and more than 2 lakh active contributors.
- Python is also known for its simplicity as compared to other programming languages.

PYTHON USE CASES

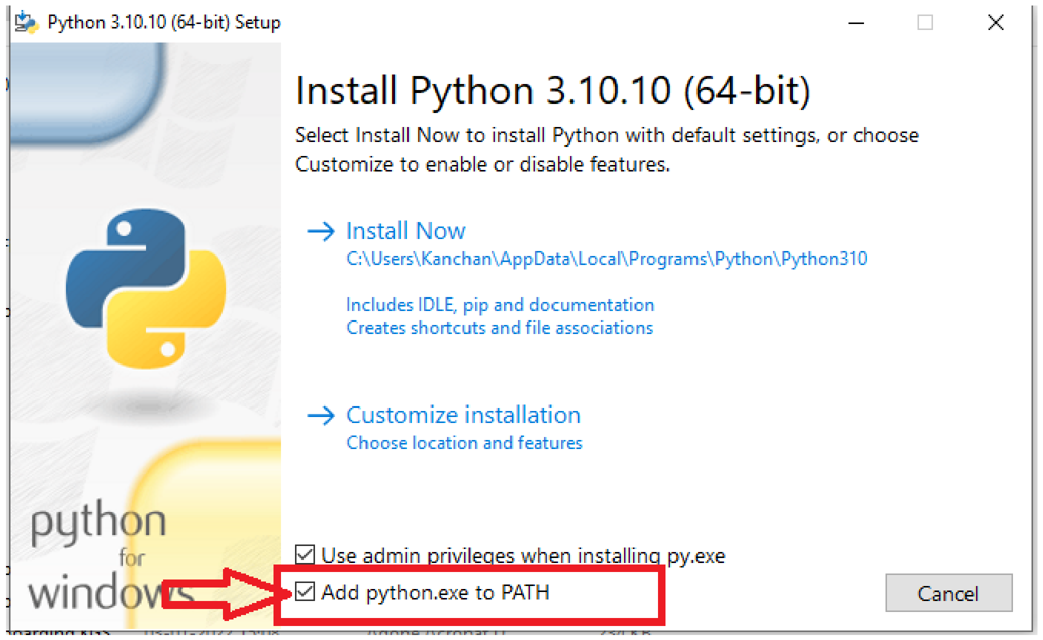
- Be it Artificial Intelligence or Web Development or IoT or Big Data Analysis or Cloud Application Development or Automation domain you can find n number of use cases for Python
- For example:
  - Instagram is coded in Python.
  - It is one of the main languages used by Google engineers.
  - Netflix uses Python to build its recommendation algorithms.
  - The Dropbox desktop application is developed in Python.
  - Reddit is coded in Python.



Python In - Depth: Use Python Programming Features, Techniques, and Modules to Solve Everyday Problems PDF 2023.

Python
Features
Easy to Write
Easy to Understand
Object-Oriented
Robust Standard Libraries
Supports Various Programming Paradigms
Support for Interactive Mode
Dynamically Typed and Type Checking
Databases and GUI Programming
Extensible
Portable
Scalable
Integrated
Automatic Garbage Collection
Free and Open Source

PYTHON INSTALLATION



Python 3.10.10 (64-bit) Setup

Install Python 3.10.10 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\Kanchan\AppData\Local\Programs\Python\Python310

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Use admin privileges when installing py.exe
☑ Add python.exe to PATH

- https://www.python.org/downloads/
- System > About
- VS IDE
- Jupyter Notebook Plugin
- ipykernel

EXECUTING FIRST PYTHON SCRIPT

- **Code**—The standard code cell. This one is reserved for writing and running Python code! You can run your code by clicking on the Run button, as seen previously.

- **Markdown**—A text cell that is mainly used for notebook documentation, to write comments, titles, equations, etc. This type lets you structure your text using HTML tags or Markdown syntax.
- For example, we could create a title. To do this, simply add a # in front of your text. A single # will correspond to a level one title, two ## to a level two title, etc.

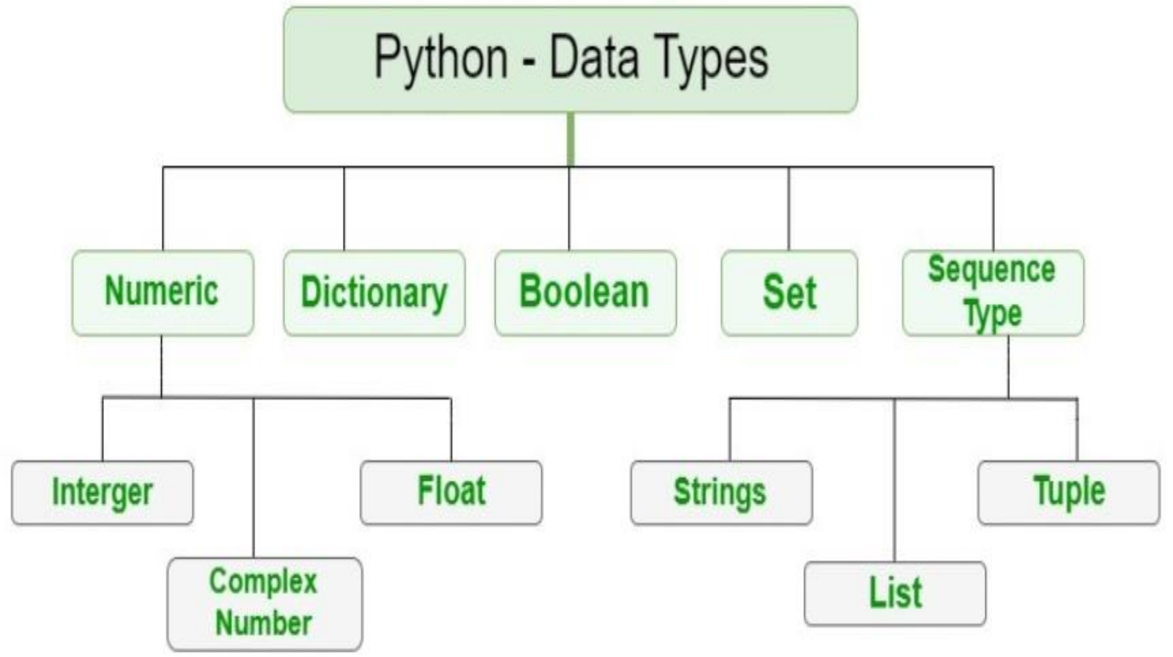
AGENDA-WW01D02

- What is Jupyter Notebook, and How to use it?
- Markdown Vs Coding in Jupyter Notebook
- Why do we need Data Types
- Built-in Data Types.
- What are different uses of Data Types.

VISUAL STUDIO CODE
AND
JUPYTER NOTEBOOK

https://code.visualstudio.com/docs/datascience/jupyter-notebooks

BUILT-IN DATA TYPES



DATA TYPE CONVERSION **(TYPECASTING)**

- You can force the conversion of a variable into a well-defined type
- int() : for integers
- float() : for decimals
- str(): for strings

AGENDA- WW01D03

VS Code and Data Types | Revision

RECAP

print()

print() is a built-in function in Python that is used to output messages or values to the console or other standard output devices. It can be used to display strings, numbers, variables, and expressions.

BREAK

DATA STRUCTRES

- Various data structures are used to organize and manipulate data. Some common data structures in Python include lists, tuples, dictionaries, sets, arrays, and queues.

- To access an element in a data structure, you can use its index. The index of an element refers to its position in the data structure.

AGENDA-WW01D04

- Variable Assignment
- How does an Object is related to the name of the Object
- How are Objects stored in Memory in Python?

COMMENTS

- Single-line comments : These are comments that start with a hash symbol (#) and continue until the end of the line.
- Multi-line comments : These are comments that span multiple lines and are enclosed in triple quotes (either single or double quotes).

INDENTATION

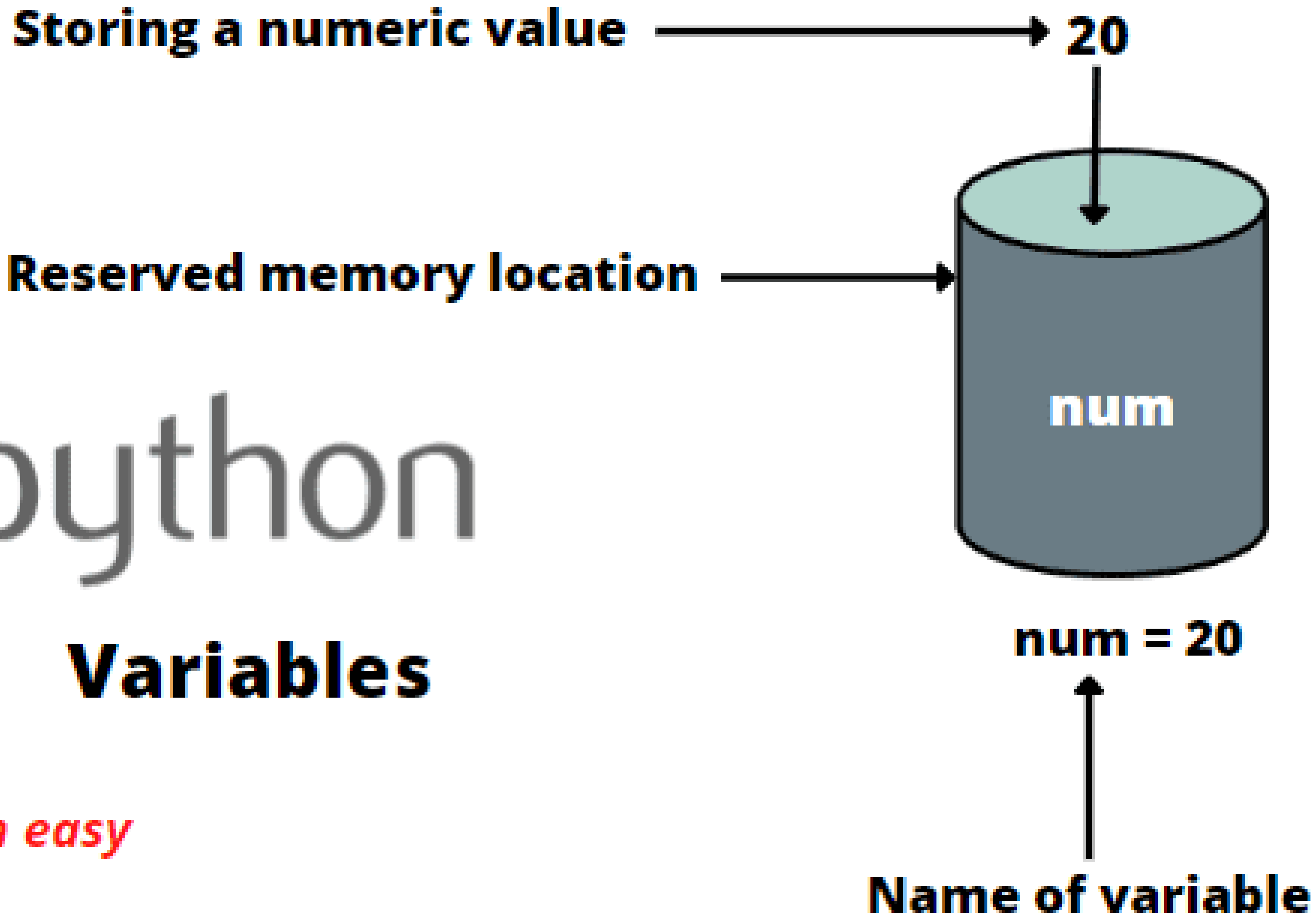
- Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.
- The most common use of indentation is in control flow statements like if/else statements, for and while loops, and function definitions

UNDERSTAND WHAT A VARIABLE IS

- **Variables** are one of the concepts found in all programming languages. You might as well say that without a variable, you can't program, and that's not an exaggeration.
- **Think of a variable as a kind of box containing a value**
- A **value** is what you will store in a variable.

VARIABLE ASSIGNMENT

- Assign values to variables using the equals sign (=)
- variable_name = value
- Python is a dynamically typed language, which means that you don't have to specify the type of a variable when you declare it. The type of the variable is inferred from the value you assign to it.


Storing a numeric value
20
Reserved memory location
num
num = 20
Name of variable

Variables

SOME GENERAL RECOMMENDATIONS

Use clear variable names

- It sounds tedious to do, but it is beneficial for you and the people you will share your code with—it makes it easier to read and maintain your code.
- For example, savingsAccount and checkingAccount are much more explicit names than account1 and account2 .

Use explicit variable names

- Avoid abbreviations and acronyms, if possible, even if an abbreviation may seem obvious.
- For example, **annualIncome** is better than **annualInc**

Follow a typographic convention

- One of the most common typographic conventions is called camel case (also known as ***camel caps***). It involves writing variable names containing several words without spaces or punctuation—the first word is written in lowercase, then each word is written with the first letter in upper case, as shown above.

MULTIPLE ASSIGNMENT

- Many Values to Multiple Variables
- x, y, z = "Orange", "Banana", "Cherry“

- One Value to Multiple Variables
- x = y = z = "Orange"

BREAK

OBJECT

- An object is an instance of a class that encapsulates data and behavior.
- Every value in Python is an object, including simple types like integers and floats, as well as more complex types like lists, dictionaries, functions, and classes themselves.
- When you create an object in Python, you are creating an instance of a class.
- An object and its name are related through a process called variable assignment. When you assign a value to a variable in Python, you are creating a reference to an object in memory.

- For example, if you create a string object "s" by assigning a value to a variable, you are creating an instance of the built-in "str" class.

OBJECT

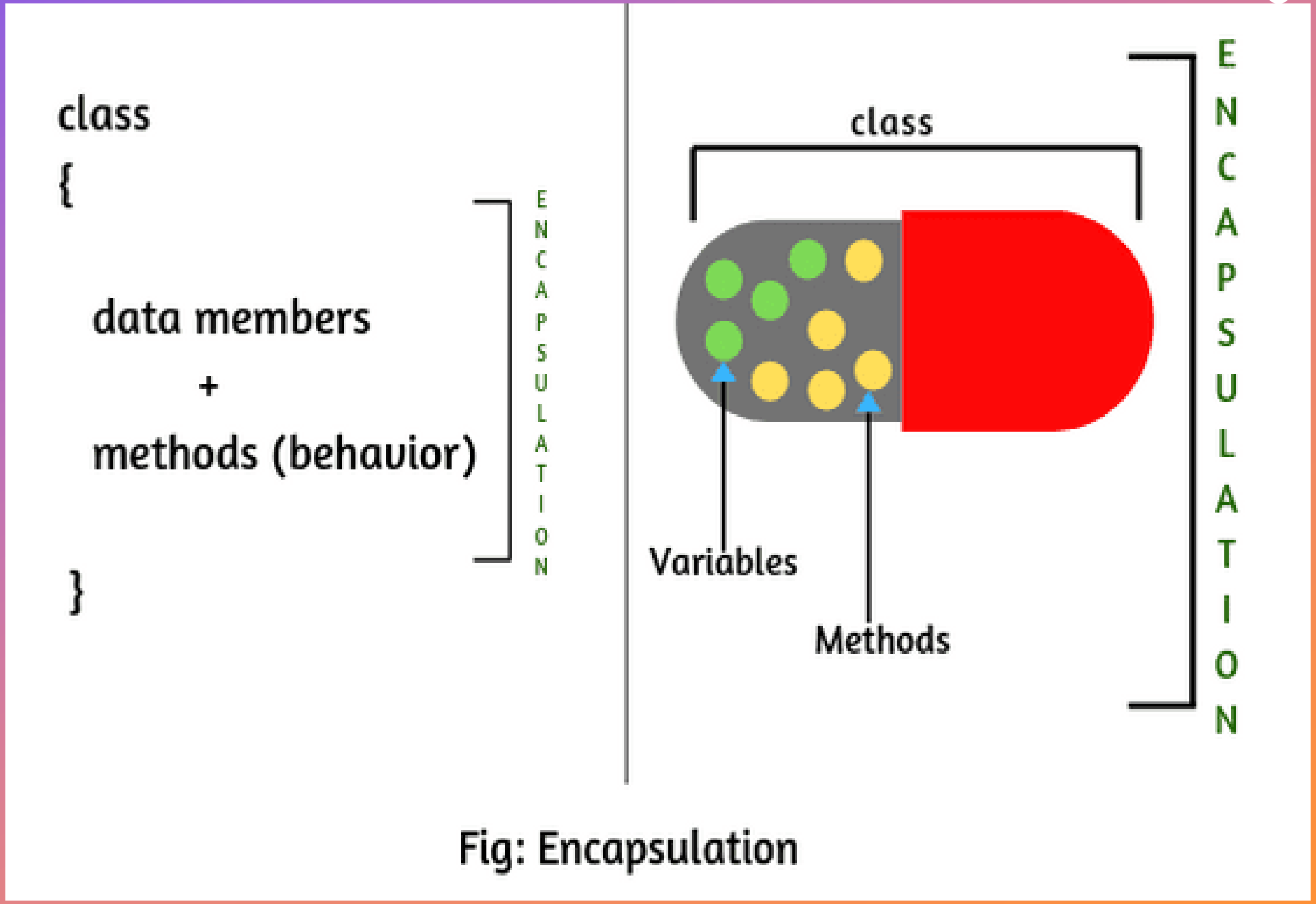
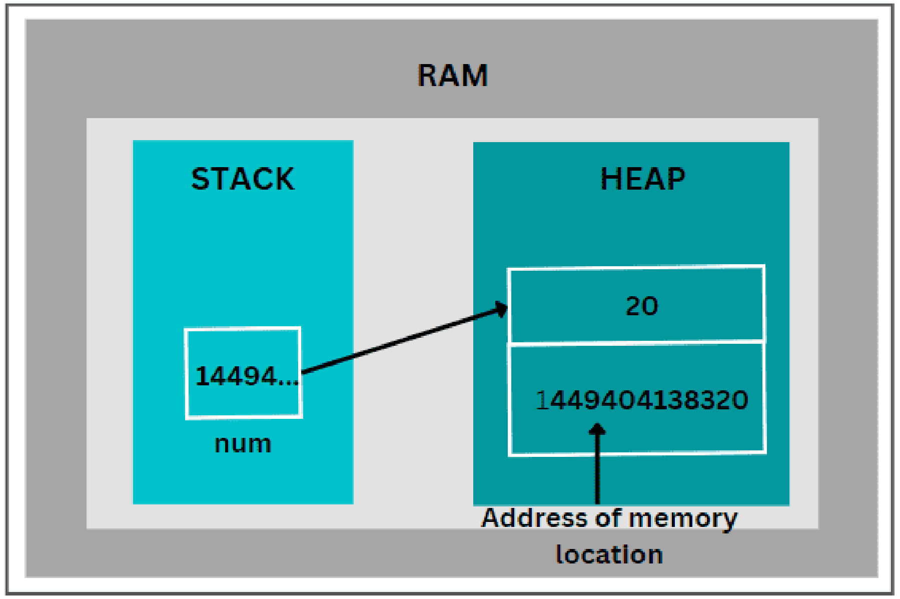


Fig: Encapsulation

HOW ARE OBJECTS STORED IN MEMORY



- Objects are stored in memory dynamically.
- When you create an object, Python allocates memory to store the object's data and behavior.
- The amount of memory allocated depends on the type and size of the object
- Python uses a private heap to store objects in memory.
- In addition to the heap, Python also uses a stack to keep track of function calls and other operations.

AGENDA-WW01D05

Immutability

Operators

IMMUTABILITY

- In Python, **objects** can be classified into two categories: mutable and immutable.
- Immutable **objects** are those whose value cannot be changed once they are created.
- The following are examples of immutable objects.
  - Numbers (int, float, bool,…)
  - Strings
  - Tuples
- Mutable objects, on the other hand, are those whose value can be changed once they are created.
- The following are examples of mutable objects:
  - Lists
  - Sets
  - Dictionaries

OPERATORS

- **Arithmetic operators**

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

- **Assignment operators**

Assignment operators are used to assign values to variables

- **Comparison operators**

Comparison operators are used to compare two values

- **Logical operators**

Logical operators are used to combine conditional statements

- **Identity operators**

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

- **Membership operators**

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

- **Bitwise operators**

Bitwise operators are used to compare (binary) numbers

AGENDA-WW01D06

- Conditional Statements
- Nested If-else
- for loop Basic Introduction

Last session remaining topics

1- Recap

2- Bitwise Operators

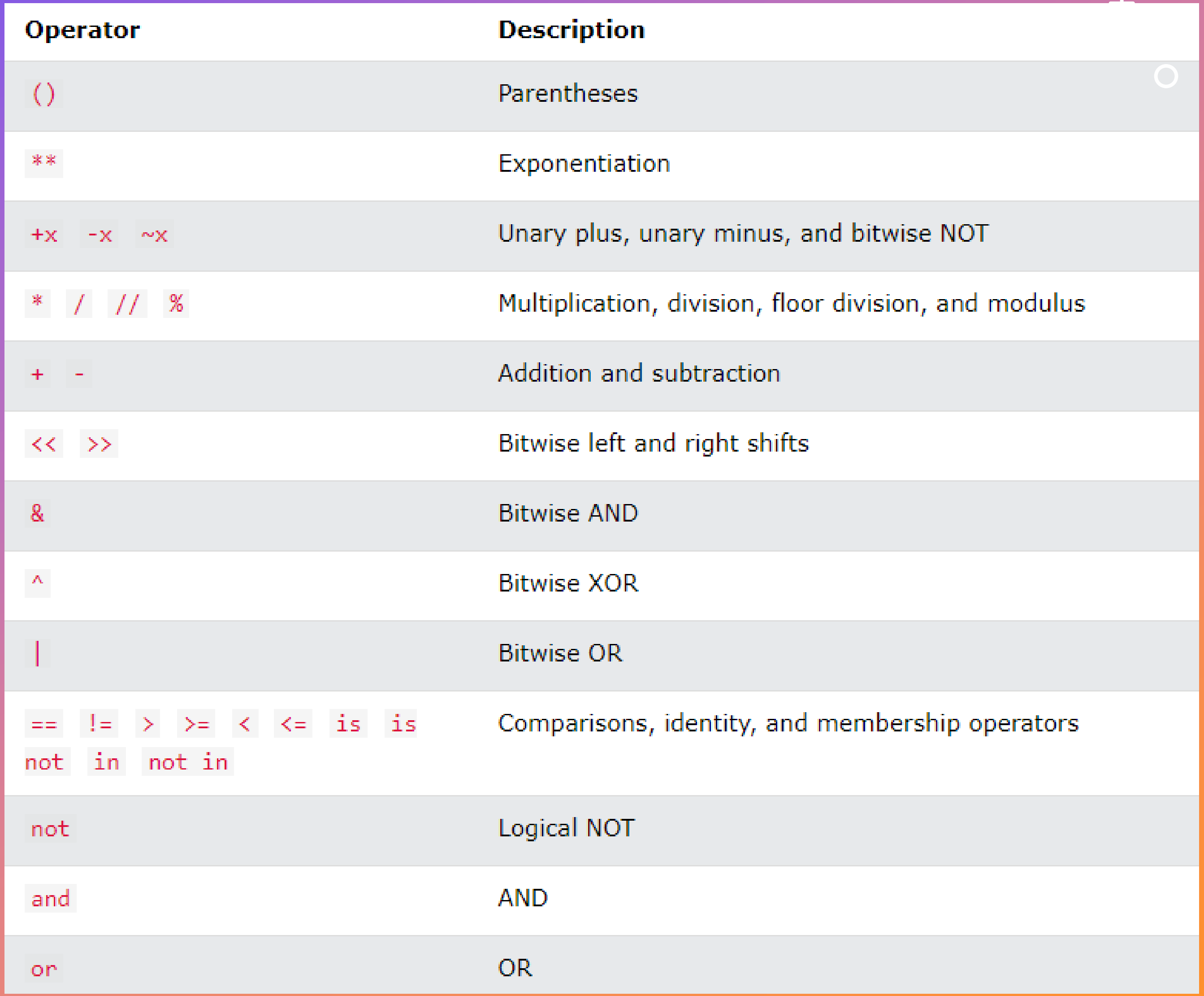
3- Operator Precedence

4- Few Key points related to Set and Dictionary wrt Immutability

OPERATOR PRECEDENCE

Operator precedence describes the order in which operations are performed.

NOTE: If two operators have the same precedence, the expression is evaluated from left to right.

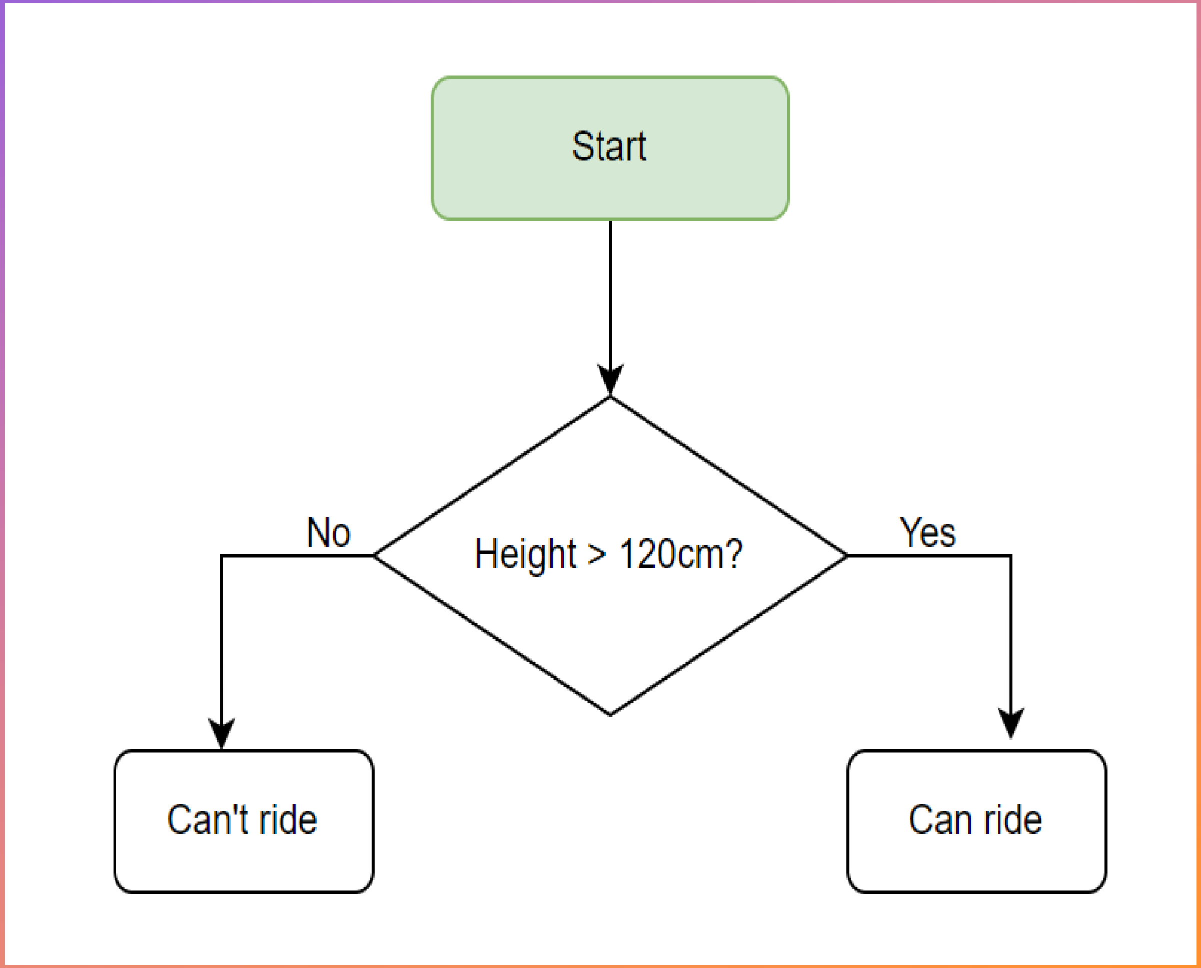


Operator	Description
`()`	Parentheses
`**`	Exponentiation
`+x` `-x` `~x`	Unary plus, unary minus, and bitwise NOT
`*` `/` `//` `%`	Multiplication, division, floor division, and modulus
`+` `-`	Addition and subtraction
`<<` `>>`	Bitwise left and right shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==` `!=` `>` `>=` `<` `<=` `is` `is not` `in` `not in`	Comparisons, identity, and membership operators
`not`	Logical NOT
`and`	AND
`or`	OR

CONDITIONALS TATEMENTS

IF | ELIF | ELSE STATEMENTS

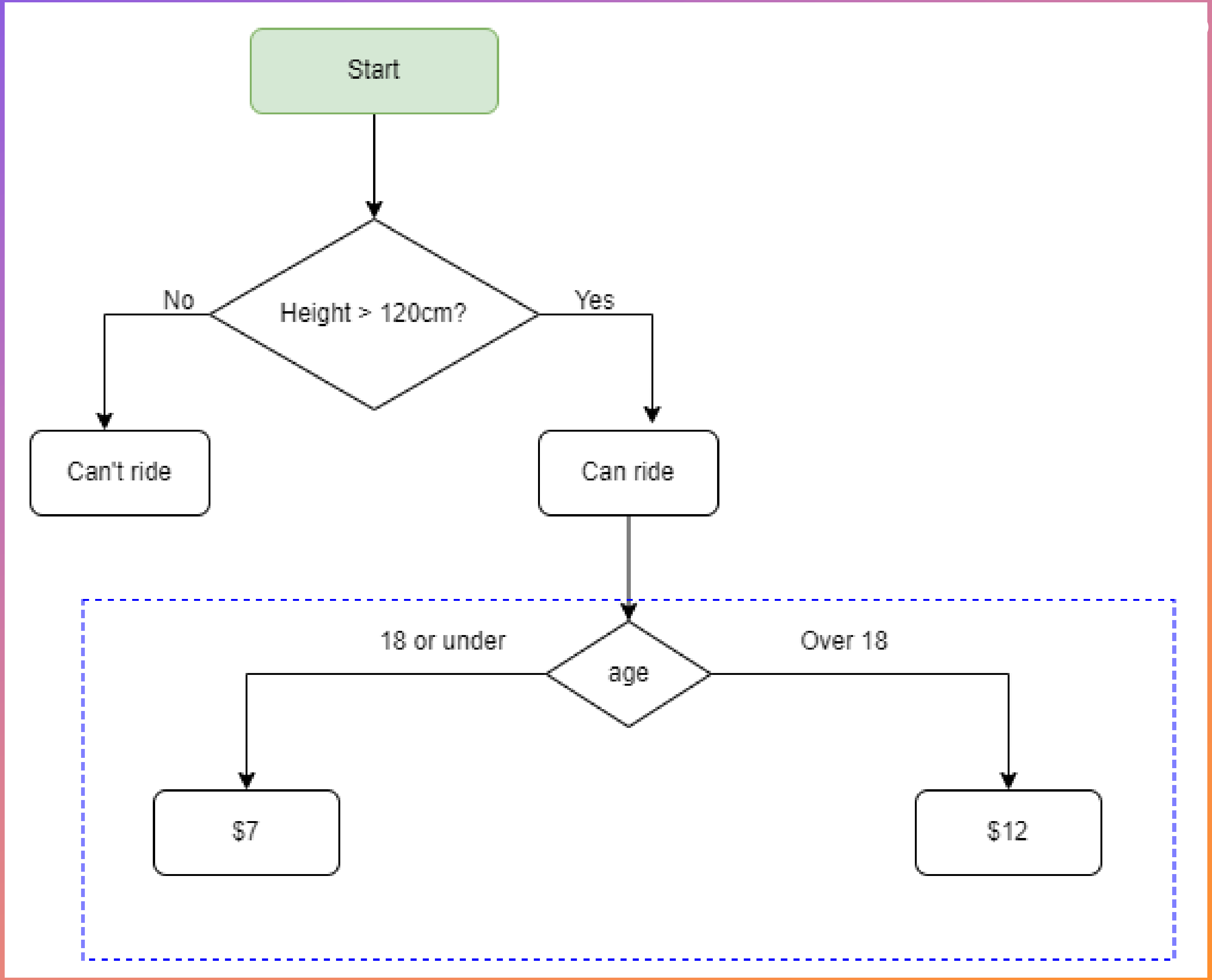
- Conditional Statement in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false. Conditional statements are handled by IF statements in Python



NESTED IF ELSE

Syntax

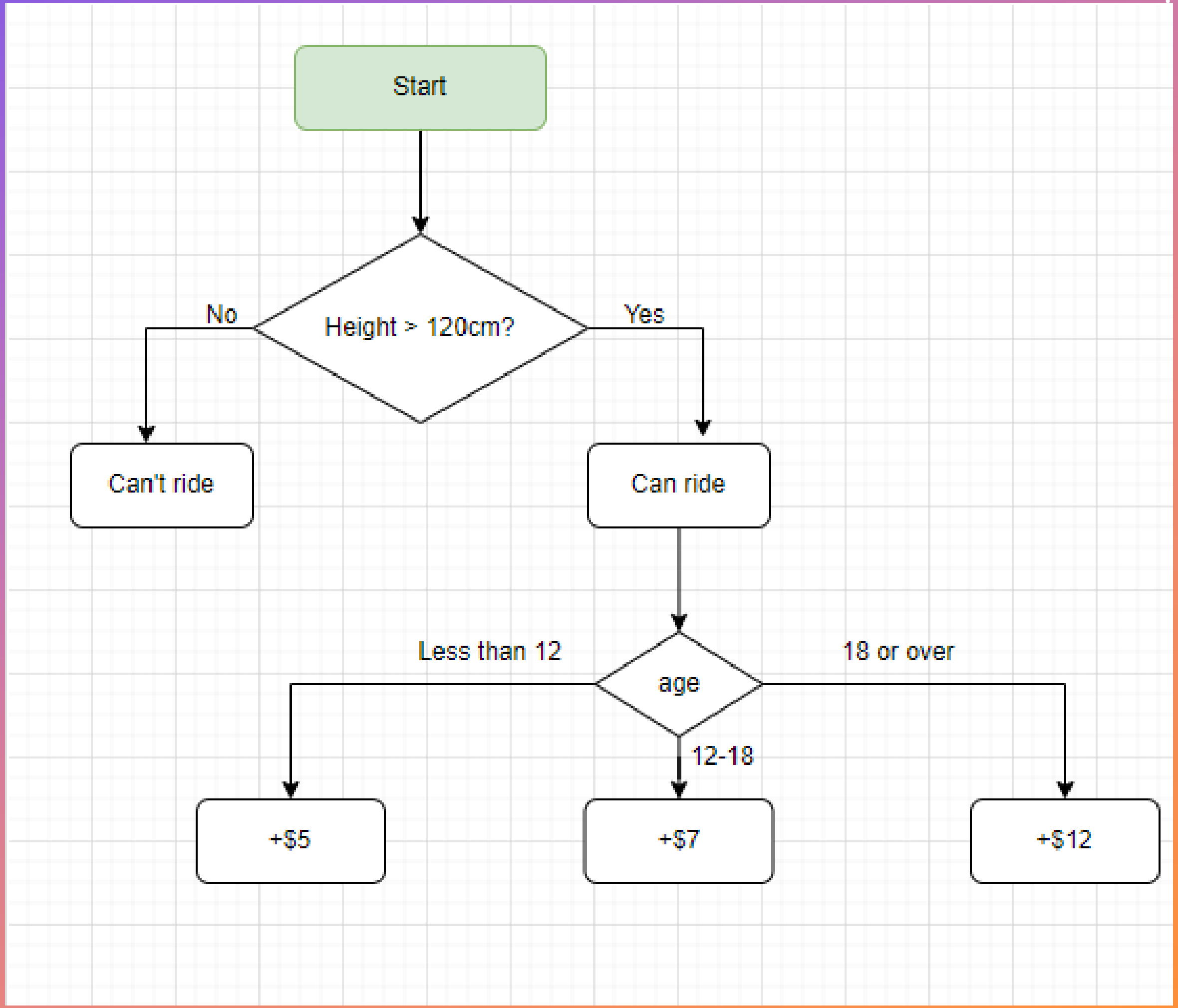
```
if expression:
   statement(s)
else:
   if expression:
      statement(s)
   else:
        statement(s)
```



IF | ELIF I ELSE

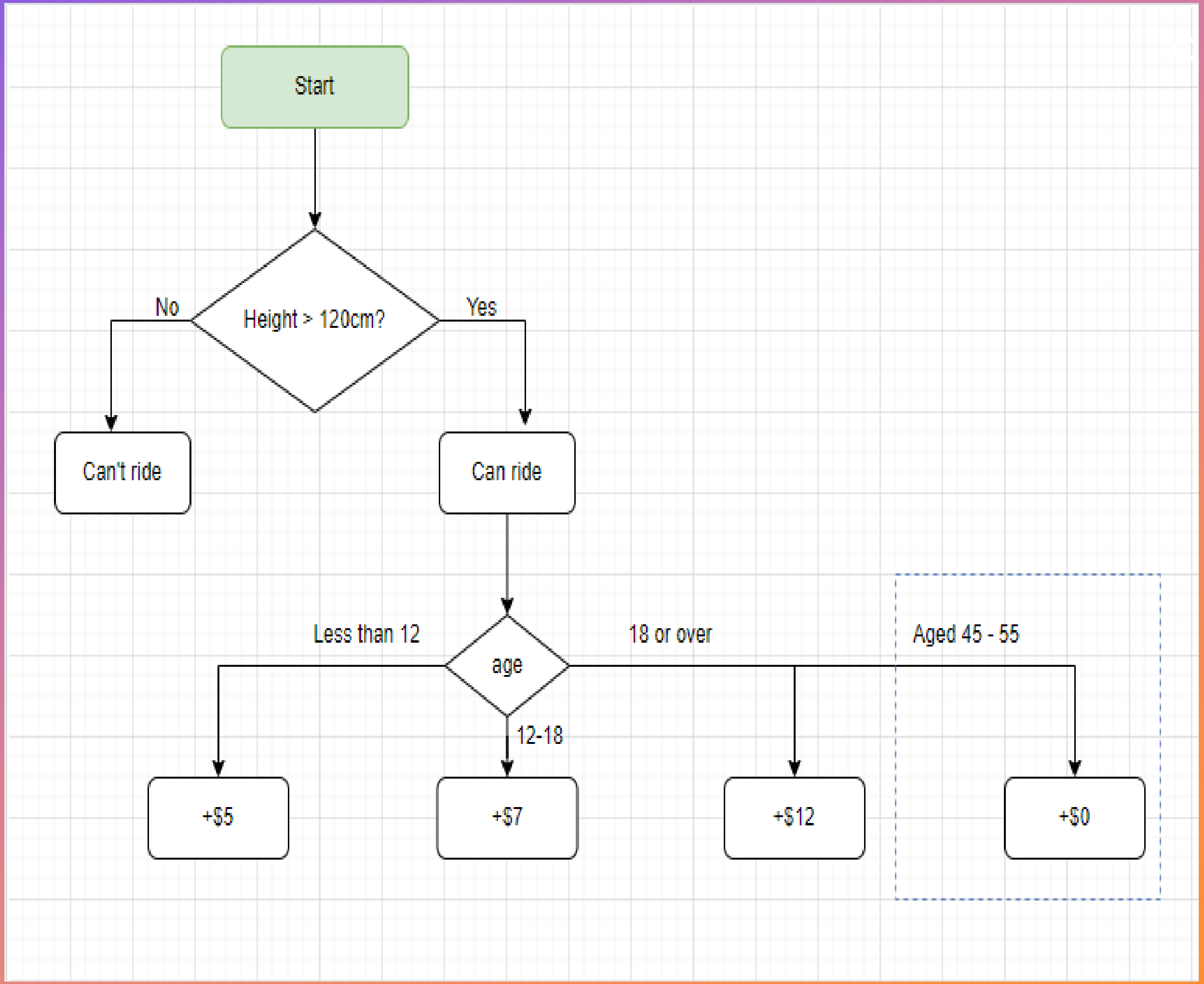
Syntax:

```
if (condition-1):
        statements
 elif (condition-2):
        statements
 else:
        statements
```



BREAK

LOGICAL OPERATORS WITH IF STATEMENT



FOR LOOP

For loops are used for sequential traversal

e.g. for [user_defined_variable] in [iterable_datatype]:

```
for [user_defined_variable] in [iterable_datatype]:
        # code inside for loop
        statements
    # code outside for loop
```

Code:

```
list = [1,2,3]
for i in list:
   print(i)
```

Output:

```
1
2
3
```

AGENDA-WW01D07

- bool()
- slicing
- Iterables
- Practice

bool()

- bool() is a built-in Python function that returns a boolean value representing the truth value of an object.
- The function takes one argument and returns True or False based on the truth value of the argument
- Example 1: Using bool() with numeric values
  - print(bool(0))   # Output: False
  - print(bool(1))   # Output: True
  - print(bool(-1)) # Output: True
- Example 2: Using bool() with strings
  - print(bool(""))   # Output: False
  - print(bool("abc")) # Output: True
- Example 3: Using bool() with lists
  - my_list = []
  - print(bool(my_list)) # Output: False
  - my_list = [1, 2, 3]
  - print(bool(my_list)) # Output: True

Slicing in different data types

- Slicing is a technique in Python that allows you to extract a portion of a sequence, such as a string, list, or tuple.

- **Syntax** :
  - [start:stop:step], where start is the index of the first element, stop is the index of the last element , and step is the step size between elements .

- Example 1: Slicing a string
  - my_string = "Hello, World!"
  - print(my_string[0:5])   # Output: Hello
  - print(my_string[7:12]) # Output: World
  - print(my_string[0:12:2]) # Output: Hlo,Wrd

- Example 2: Slicing a list
  - my_list = [1, 2, 3, 4, 5]
  - print(my_list[1:3])   # Output: [2, 3]
  - print(my_list[::2])   # Output: [1, 3, 5]

- Example 3: Slicing a tuple
  - my_tuple = (1, 2, 3, 4, 5)
- print(my_tuple[1:3])   # Output: (2, 3)
  - print(my_tuple[::2])   # Output: (1, 3, 5)

Iterables

- An iterable is any object capable of returning its elements one at a time, allowing iteration over the elements
- Most common iterables are strings, lists, tuples, and dictionaries, but other objects can be iterated as well, such as sets, files, and generators.
- Example 1: Iterating over a string

```
my_string = "Hello, World!"
for char in my_string:
    print(char)
```

- Example 2: Iterating over a list

```
my_list = [1, 2, 3, 4, 5]
for item in my_list:
    print(item)
```

- Example 3: Iterating over a dictionary

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
for key in my_dict:
    print(key, my_dict[key])
```

Questions to practice for if else

Programming Practice

https://csiplearninghub.com/python-if-else-conditional-statement-practice/

Quiz
https://realpython.com/quizzes/python-conditional-statements/

AGENDA-WW01D08

- for loop
- range(), len()
- help()
- Nested for loop
- for loop practice

FOR LOOP RECAP

range()

- The range() function is a built-in function in Python that generates a sequence of numbers.
- Syntax:

  range(start, stop, step)
  - start: (optional) The starting number of the sequence. If not specified, it defaults to 0.
  - stop: (required) The number at which the sequence stops (but doesn't include in the sequence).
  - step: (optional) The difference between each number in the sequence. If not specified, it defaults to 1.

- Uses of range function
  - Generate sequence of numbers
  - With for loop for iteration

len()

- len() is a built-in function that is used to return the number of items (elements) in an object.
- The object could be a list, tuple, string, dictionary, set, or any other iterable object.
- Syntax:
  - len(iterable_object)

- Usage of len()
  - To calculate size
  - In loops

help()

- help() function is a built-in function in Python that provides documentation on a specific Python object or module.
- It can be used to get information about the function or module's purpose, arguments, return value, and usage examples
  - # Getting help on the 'print' function

```
help(int)
```

BREAK

Nested for loop

A nested for loop in Python is a loop that contains another loop within its block.

It is used to iterate over a collection of elements that are themselves collections, such as a list of lists or a dictionary of lists.

Syntax:

```
for variable1 in iterable1:
    # outer loop code
    for variable2 in iterable2:
        # inner loop code
```

AGENDA-WW01D09

- while
- nested loop
- break
- continue
- pass
- practice

WHILE LOOP

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

```
e.g.  while (condition):
                    # code inside while loop if condition is true
                    statements
      # code outside while loop when condition is false
```

Code:

```
count = 0
while (count<2):
    print(count)
    count += 1
```

Output:

```
0
1
```

Nested loop

Nested loop means using another loop inside a loop.

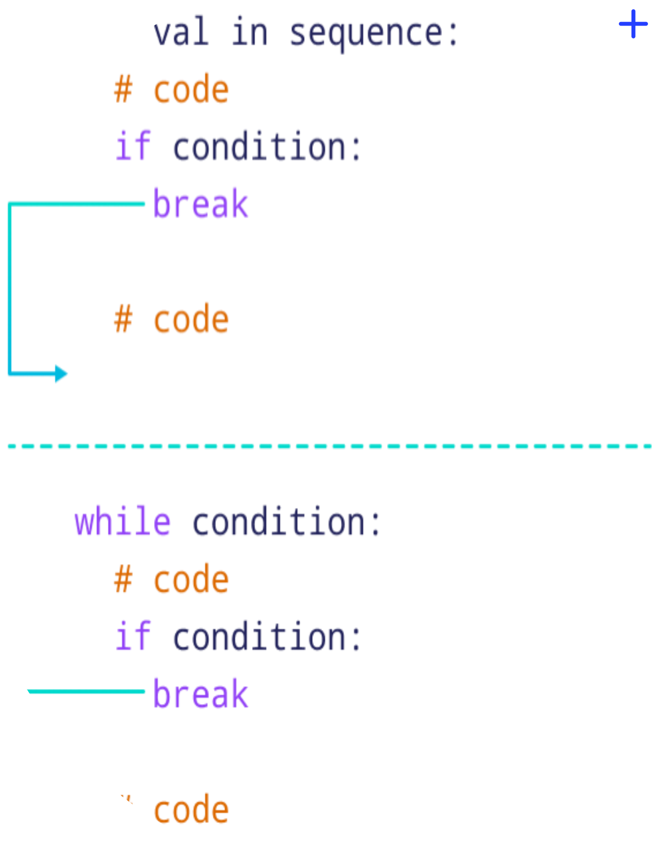
e.g.
```
while (condition):
        # code inside while loop if condition is true
        for [user_defined_vatiable] in [iterable_datatype]:
                # this for loop is a nested loop inside a while loop
                statements #inside for loop
        statements # outside for loop and inside while loop
# code outside while loop when condition is false
```

break

The break statement is used to terminate the loop. After that, the control will pass to the statements that are present after the loop, if available. If the break statement is present in the nested loop, then it terminates only those loops which contains break statement.

Syntax:

```
for [user_defined_vatiable] in [iterable_datatype]:
        statements
        some conditions:
                break
```



```
    val in sequence:
  # code
  if condition:
    break

  # code
```

```
while condition:
  # code
  if condition:
    break

  code
```

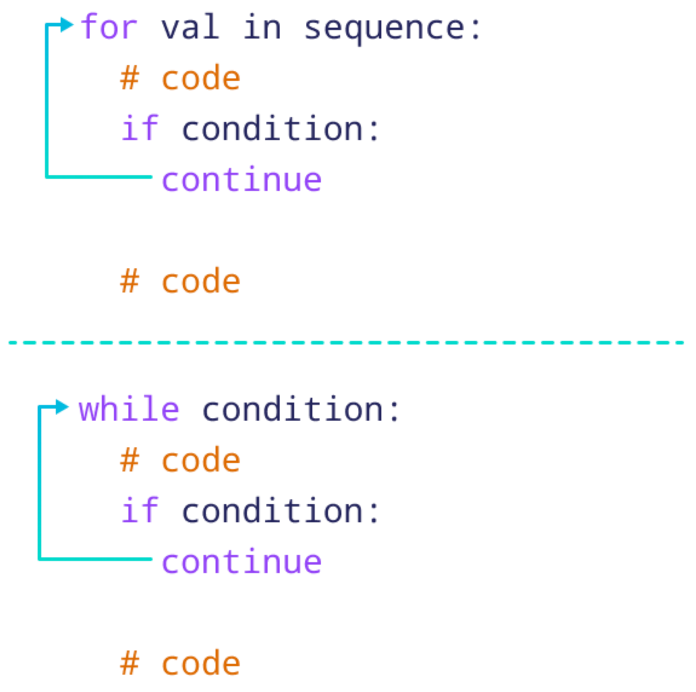
continue

Continue is also a loop control statement just like the break statement. continue statement is opposite to that of break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.

When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

Syntax:

```
for [user_defined_vatiable] in [iterable_datatype]:
        statements
        some conditions:
                continue
```



```
for val in sequence:
    # code
    if condition:
        continue

    # code
```

```
while condition:
    # code
    if condition:
        continue

    # code
```

BREAK

pass

As the name suggests pass statement simply does nothing. The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. It is like null operation, as nothing will happen is it is executed. Pass statement can also be used for writing empty loops. Pass is also used for empty control statement, function and classes.

Syntax:

```
for [user_defined_vatiable] in [iterable_datatype]:
        pass
```

Problems to practice on while loop

https://csiplearninghub.com/questions-of-while-loop-in-python/

AGENDA-WW01D10

Strings

Practice Questions

input(), print()

split and join

builtin functions related to find

replace and count builtin funcitons

upper(),lower(),isupper(),islower(),title() ,istitle(),capitalize()

STRINGS

A string is a sequence of characters. It can be declared in python by using double quotes. Strings are ordered(indexing and slicing operations can be perform).Strings are immutable, i.e., they cannot be changed.

e.g. a = “This is a string.”

input()

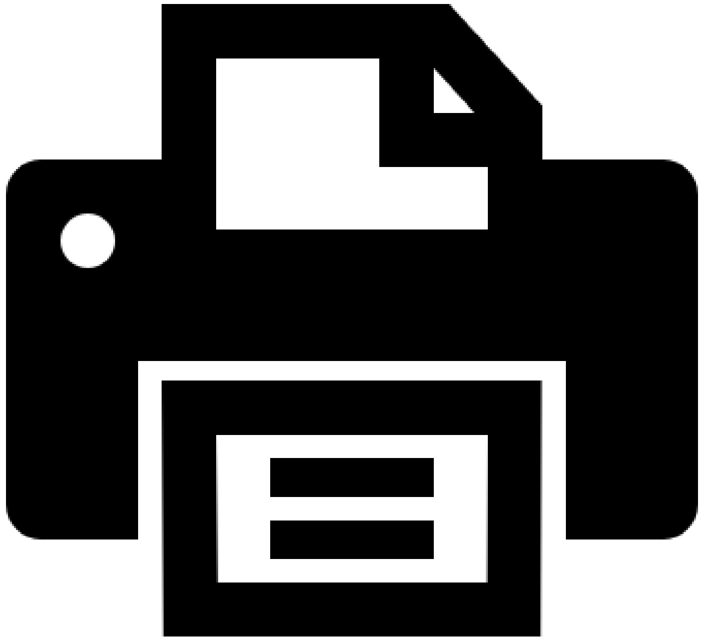
- input() is a built-in Python function used for getting input from the user via the command line or console. It is commonly used to prompt the user to enter some data and then use that data in the program.
- Note that the input() function always returns a string, so if you need to use the input as a different data type, you will need to cast it explicitly.

print()

- print() is a built-in function in Python that is used to output messages or values to the console or other standard output devices. It can be used to display strings, numbers, variables, and expressions.
- **Syntax**:

```
print(*objects, sep= ' ', end = '\n', file=file, flush=flush)
```

- **objects** - object to the printed. * indicates that there may be more than one object
- **sep** - objects are separated by sep. Default value: ' '
- **end** - end is printed at last
- **file** - must be an object with write(string) method. If omitted, sys.stdout will be used which prints objects on the screen.
- **flush** - If True, the stream is forcibly flushed. Default value: False
- It doesn't return any value; returns None



split()

- split() is a built-in function in Python that allows you to split a string into a **list of substrings** based on a specified delimiter.
- Syntax:
  - string.split(separator, maxsplit)
- The separator argument is the delimiter that will be used to split the string. If it is not specified, the default delimiter is a space.
- The maxsplit argument is optional and specifies the maximum number of splits to be performed. If it is not specified, all occurrences of the delimiter will be used for splitting.

join()

- join() is a built-in function in Python used to join together a sequence of strings, where the string used as a separator is inserted between each of the original strings.
- Syntax:
  - separator_string.join(iterable)
- Separator_string is the string that will be used as a separator between the elements of the iterable object.

find()

- This method is used to find the index of the first occurrence of a substring within a string. It returns -1 if the substring is not found.
- Syntax:
  - str.find(substring, start=0, end=len(string))
- substring - It is the substring to be searched in the str string.
- start and end (optional) - The range str[start:end] within which substring is searched.
- find() Return Value
  - The find() method returns an integer value:
  - If the substring exists inside the string, it returns the index of the first occurence of the substring.
  - If a substring doesn't exist inside the string, it returns -1.

index()

- This method is similar to find(), but raises a ValueError exception if the substring is not found

count()

- This method is used to count the number of occurrences of a substring in a string.

in

- This operator is used to check if a substring is present in a string. It returns a boolean value (True or False)

AGENDA-WW01D11

- Different Patterns and Series
- String Questions
- String built-in functions
- Lists
- Negative index and slicing

Palindrome

- A Palindrome is a series of letters, as a word, or a series of numbers, which says the same word when being read both in forward and backward directions.
- **EX**: “radar”, “level”, 3553, 12321, 'Was it a car or a cat I saw'

Fibonacci sequence

- A Fibonacci sequence is the integer sequence of 0, 1, 1, 2, 3, 5, 8....
- The first two terms are 0 and 1. All other terms are obtained by adding the preceding two terms. This means to say the nth term is the sum of (n-1)th and (n-2)th term

Armstrong

- A positive integer is called an Armstrong number of order n if

  abcd... = a^n + b^n + c^n + d^n + ...
- Ex: 153 = 1*1*1 + 5*5*5 + 3*3*3 // 153 is an Armstrong number.

Operations on string

1. lower() : Converts the string in lowercase.

e.g. a.lower() // output: 'this is a string'

2. **Upper()** : Converts the string in uppercase.

e.g. a.upper() // output: 'THIS IS A STRING'

3. **isupper()/islower()** : This gives true/false according to case of string lower or upper.

4. **replace()** : Replaces a word/letter in string.

e.g. variable_name.replace("old:str","new:str")

5. **format()** : We can use format function for formatting string in print statement .

e.g.

num1 = 50

print("Value of num1 is {}".format(num1))

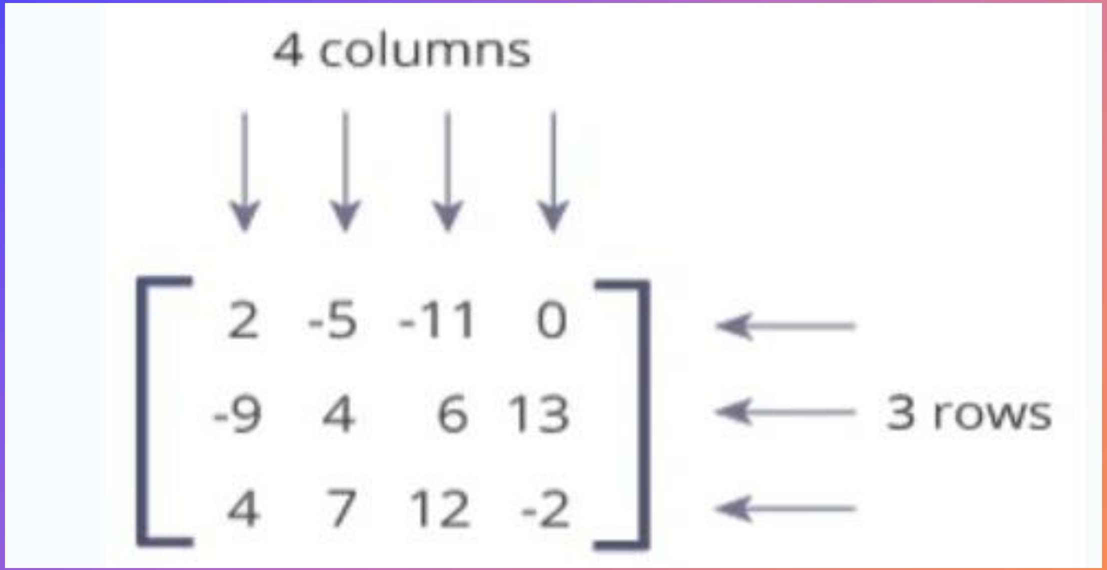
For more such functions prefer python docs.

Link => https://docs.python.org/3/library/stdtypes.html#string-methods

LISTS

Lists are mutable sequences, typically used to store collections of heterogeneous items. Lists are ordered means indexing and slicing.

e.g.
l = [1, 2, 3, “abc”, [0, 1], {‘name’:’abc’, ’roll no’:123}, (a, b, c)]



Nested List

- List inside a List

- A matrix is a two-dimensional data structure where numbers are arranged into rows and columns.

- Eg:

  matirx = [[1,3,3],[2,4,5],[5,6,7],[6,7,8]]

BREAK

NEGATIVE INDEXING

- Use negative indexes to start the slice from the end
- Get the characters from position 5 to position 1, starting the count from the end of the list
- Example

```
myList = ['A', 'B', 'C', 'D', 'E’]
print(myList[-1])
```

NEGATIVE SLICING IN PYTHON

- Similar to negative indexing, Python also supports negative slicing. This means you can start slicing from the end of the sequence.
- For example, let’s access the last 3 values of a list:

```
nums = [1, 2, 3, 4, 5, 6, 7]
part = nums[-4:-1]
print(part)
```

- Here the slicing starts at the index -4 which is the 4th last element of the list. The slicing ends at the last item, which is at the index -1.

- Also, the step size can be a negative value. By doing this, the direction of slicing is reversed. This means the slicing **starts from the sequence[stop + 1] and stops at the sequence[start] value**.

AGENDA-WW01D12

- Lists append, extend, insert, delete
- Concatenation and ord()
- Enumerate
- Questions on Lists
- Builtin Functions
- Sort()

Operations on string

1- Concatenation of String :

We can concatenate strings using the "+" operator or the "join()" method.

```
string1 = "Hello"
string2 = "world"
result = string1 + " " + string2
print(result) # Output: "Hello world“
```

Note: Concatenation is supported by list as well.

2- ord() :

It is a built-in function in Python that takes a single character string as an argument and returns an integer representing the Unicode code point of that character.

```
char = 'A’
code_point = ord(char)
print(code_point) # Output: 65
```

NOTE: that the ord() function only works with a single character string. If you pass a string with more than one character or a non-string argument, Python will raise a TypeError.

https://www.geeksforgeeks.org/ascii-table/

Operations on lists

1. **Add** : To add item in a list.

i) append(): To add an item at the end of list.
e.g. l.append(item)

ii) extend(): To add many items at the end of list.
e.g. l.extend(item1,item2..etc.)

iii) insert(): insert an item at given index.
e.g. l.insert(index_value, item)

2. **Update**: To update or replace items in a list.

e.g. l[index] = new_value

3. **Delete**: To remove an item/element from list.

i) del list[index_value]: To remove an item from a given index value.
ii) remove(item): To remove a specific item.
iii) clear(): To remove/delete all element from list.

For more such functions prefer python docs.
Link https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

index()

- This method is similar to find(), but raises a ValueError exception if the item is not found in list

count()

- This method is used to count the number of occurrences of a element in a list.

in

- This operator is used to check if a element is present in a list. It returns a boolean value (True or False)

sort()

- sort() is a built-in method in Python that can be used to sort a list in ascending order.
- The method sorts the list in place, which means it modifies the original list.
- Ex:

```
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
numbers.sort()
print(numbers) # Output: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
```

- By default, the sort() method sorts the list in ascending order. However, you can also pass a reverse=True argument to sort the list in descending order:
- Ex:

```
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
numbers.sort(reverse=True)
print(numbers) # Output: [9, 6, 5, 5, 5, 4, 3, 3, 2, 1, 1]
```

BREAK

enumerate

A lot of times when dealing with iterators, we also get a need to keep a count of iterations. Python eases the programmers' task by providing a built-in function enumerate() for this task.
Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object. This enumerate object can then be used directly in for loops or be converted into a list of tuples using list() method.

Syntax: enumerate(iterable, start)

Parameters:
i) Iterable: any object that support iteration
ii) start: the index value from which the counter is to be start by default it is 0.

AGENDA-WW01D13

- lstrip, rstrip, strip in Strings
- Tuples
- Lists vs Tuples
- Use cases and Practice
- Commonly Used builtin functions
- Dictionary

lstrip()

- Built-in string method that is used to remove leading whitespace characters from a string
- **Syntax**:
  - string.lstrip([chars])
  - where string is the string that you want to strip the whitespace from, and chars is an optional parameter that specifies the set of characters to remove from the left side of the string.
- **Ex:**

```
text = "   hello world   "
new_text = text.lstrip()
print(new_text)
```

rstrip()

- Built-in string method that is used to remove leading whitespace characters from a string
- **Syntax**:
  - string.rstrip([chars])
  - where string is the string that you want to strip the whitespace from, and chars is an optional parameter that specifies the set of characters to remove from the right side of the string.
- **Ex:**
  - text = "   hello world   “
  - new_text = text.rstrip()
  - print(new_text)

strip()

- Built-in string method that is used to remove leading whitespace characters from the beginning and end of a string
- **Syntax**:
  - string.strip([chars])
  - where string is the string that you want to strip the whitespace from, and chars is an optional parameter that specifies the set of characters to remove from both end of the string.
- **Ex:**
  - text = "   hello world   “
  - new_text = text.strip()
  - print(new_text)

TUPLE

- A tuple is a sequence of immutable(can not update/change values in it) Python objects.
- Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.
- Creating a tuple is as simple as putting different comma-separated values in parentheses ().
- Ex: Days in a week

TUPLE VS LIST

Key points to remember:

- The literal syntax of tuples is shown by parentheses () whereas the literal syntax of lists is shown by square brackets [] .
- Lists has variable length; tuple has fixed length.
- List has mutable nature; tuple has immutable nature.
- List has more functionality than the tuple. Check using dir(list/tuple)

Commonly used Python builtin functions

- **sum() :** This function is used to get the sum of a list or tuple.
- **min() :** This function is used to get the minimum value in a list or tuple.
- **max():** This function is used to get the maximum value in a list or tuple.

DICTIONARY

- Dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys.
- Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key.
- You can’t use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend().
- It is best to think of a dictionary as a set of key: value pairs, with the requirement that the keys are unique (within one dictionary).
- e.g. d = {‘name’:’abc’, ‘roll number’: 123}

Nested List and Dictionary

Nesting a List in a Dictionary

travel_log = { "France": ["Paris", "Lille", "Dijon"], "Germany": ["Berlin", "Hamburg", "Stuttgart"],}

Nesting Dictionary in a Dictionary

travel_log = { "France": {"cities_visited": ["Paris", "Lille", "Dijon"], "total_visits": 12}, "Germany": {"cities_visited": ["Berlin", "Hamburg", "Stuttgart"], "total_visits": 5},}

AGENDA-WW01D14

- Dictionary
- Dictionary Methods
- Sets
- Operations on sets
- Practice

DICTIONARY

- len()
- datatypes
- dict constructor

Dictionary Methods

1. **clear():** Remove all items from the dictionary.

2. **copy():** Return a shallow copy of the dictionary.

3. **fromkeys(seq[, v]):** Return a new dictionary with keys from seq and value equal to v (defaults to None).

4. **get(key[,d]):** Return the value of key. If key does not exist, return d (defaults to None).

5. **items():** Return a new view of the dictionary's items (key, value) as tuples.

6. **keys():** Return a new view of the dictionary's keys.

Dictionary Methods

7. pop(key[,d]): Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.

8. popitem(): Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty.

9. setdefault(key[,d]): If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None).

10. update([other]): Update the dictionary with the key/value pairs from other, overwriting existing value of keys.

11. values(): Return a new view of the dictionary's values

SET

- A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.
- e.g. set = {"apple", "banana", "cherry"}
- NOTE: Sets are unordered, so you cannot be sure in which order the items will appear.
- You cannot access items in a set by referring to an index, since sets are unordered the items has no index.
- Once a set is created, you cannot change its items, but you can add new items.

add()

- we use the add() method to add an item to a set.
- For example,

```
numbers.add(num)
```

update()

- update() method is used to update the set with items other collection types (lists, tuples, sets, etc).
- For example,

```
companies = {'Lacoste', 'Ralph Lauren’}
tech_companies = ['apple', 'google', 'apple']
companies.update(tech_companies)
print(companies)
# Output: {'google', 'apple', 'Lacoste', 'Ralph Lauren'}
```

discard()

- We use the discard() method to remove the specified element from a set.
- For example,

```
languages = {'Swift', 'Java', 'Python'}
print('Initial Set:',languages)
# remove 'Java' from a set
removedValue = languages.discard('Java')
print('Set after remove():', languages)
```

Operations on sets

- **Union:** Gives all element from both sets without repetition. (using “|”)
- **Intersection:** Gives only common element from both sets. (using “&”)
- **Difference:** Gives element of 1st set which are not present in 2nd set. (using “-”)
- **Symmetric Difference:** Gives all those elements which are not common (using “^”)

- **e.g.** A = {0, 2, 4, 6, 8}, B = {1, 2, 3, 4, 5}
- Union : [0, 1, 2, 3, 4, 5, 6, 8]
- Intersection : [2, 4]
- Difference : [8, 0, 6]
- Symmetric difference : [0, 1, 3, 5, 6, 8]

AGENDA-WW01D15

- About Functions
- Functions Syntax and usages
- Local, Global and Builtin Namespaces
- Argument Passing Techniques

DICTIONARY ITERATION

- We can iterate over a dictionary using the items() method. The items() method returns a view object that contains key-value pairs of the dictionary as tuples.

- Ex:

```
fruits = {'apple': 2, 'banana': 4, 'orange': 1}

for key, value in fruits.items():
    print(key,value)
```

FUNCTIONS

- A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.
- As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.
- Syntax:

```
def function_name(arguments):
        # write all your code here
# outside of function
```

DEFINING A FUNCTION

- You can define functions to provide the required functionality. Here are simple rules to define a function in Python.
- Function blocks begin with the keyword def followed by the function name and parentheses ( ).
- Any input parameters or arguments should be placed within these parentheses.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function.

NAMESPACE

- **Namespace** is a container that holds a set of identifiers (variable names, function names, class names, etc.) and maps them to their corresponding objects.

- **Built-in Namespace:** This namespace contains all the built-in functions and exceptions that come with Python.

- **Global Namespace:** This namespace contains all the identifiers that are defined in the global scope of a program. These identifiers are accessible from anywhere in the program.

- **Local Namespace:** This namespace contains all the identifiers that are defined inside a function. These identifiers are accessible only within the function.

SCOPE OF VARIABLE:

- **Global variables** are the one that are defined and declared outside a function, and we need to use them inside a function.
- **Local variables** are defined inside a function.
- **Ex**

```
a = “this is a global variable”
def local():
        b = “this is a local variable”
        print(a) # we can use a inside/outside function
print(b) # we cannot use b outside the function
```

- Using global keyword, we can make a local variable global.
- **Ex**

```
def func_name():
        global b
        b = “This is a local variable without ‘global b’ statement but now it is global variable ”
#outside the func we can use b
print(b)
```

BREAK

ARGUMENT PASSING TECHNIQUES

1- **Positional Arguments**

- Positional arguments are the most common way to pass arguments to a function.
- They are matched based on their position, meaning the first argument passed is assigned to the first parameter in the function, the second argument to the second parameter, and so on.

- Ex:

```
def add_numbers(x, y):
    return x + y

result = add_numbers(3, 4)
print(result) # Output: 7
```

ARGUMENT PASSING TECHNIQUES

2- **Keyword Arguments**

- Keyword arguments are passed with the name of the parameter as a keyword.
- This allows you to pass arguments in any order and to omit any arguments that have default values.
- Ex:

```
def greet(name, message='Hello'):
    print(f"{message}, {name}!")

greet(name='John') # Output: Hello, John!
greet(message='Hi', name='Kate') # Output: Hi, Kate!
```

ARGUMENT PASSING TECHNIQUES

- ***args :**

The special syntax *args in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-keyworded, variable-length argument list.

- e.g.

```
def myFun(*argv):
        for arg in argv:
                print (arg)
myFun(‘Hello’, ‘Welcome’, ‘to’, ‘Edyoda’)
```

- ****kwargs** :

The special syntax **kwargs in function definitions in python is used to pass a keyworded, variable-length argument list. We use the name kwargs with the double star. The reason is because the double star allows us to pass through keyword arguments

- e.g.

```
def myFun(**kwargs):
        for key, value in kwargs.items():
                print ("%s == %s" %(key, value))
myFun(first ='assessments', mid ='edyoda', last='com')
```

return

- return statement is used to return a value from a function.
- When a return statement is encountered inside a function, the function immediately terminates and returns the specified value to the caller.
- Function returns None by default.
- None is a built-in object that represents the absence of a value or a null value.

```
def add_numbers(a, b):
        result = a + b
        return result
sum = add_numbers(10, 20)
print(sum)  # Output: 30
```

AGENDA-WW01D16

- What are Modules
- What are Packages
- Modules in Detail
- Packages in Detail

MODULES

- A module is a file containing Python definitions and statements that can be imported into other Python files.
- A module can define functions, classes, and variables, and can include runnable code.
- Modules are used to organize code into logical and reusable units, making it easier to manage and maintain large programs.
- By using modules, you can encapsulate functionality and keep it separate from other parts of your code.
- This makes your code more modular and easier to understand and modify.

TYPE OF MODULES

- **Built-in modules:** These are modules that are included with Python itself and provide basic functionality. Examples include the os module for working with the operating system, the math module for mathematical operations, and the datetime module for working with dates and times.

- **Third-party modules:** These are modules that are created by third-party developers and are not part of the Python standard library. They are usually installed using a package manager like pip. Examples include the numpy module for scientific computing, the pandas module for data analysis, and the requests module for working with HTTP requests.

- **Package modules**: These are modules that are part of a package, which is a collection of related modules organized in a directory hierarchy. Each package contains an __init__.py file, which is executed when the package is imported. Examples include the numpy package, which contains modules for numerical computing, and the matplotlib package, which contains modules for data visualization.

MODULES

- Python comes with a large number of built-in modules, such as math, os, and random, which provide commonly used functionality.
- To use a module in Python, you typically need to import it using the import statement. For example, to use the math module, you would use the following code:

```
import math
x = math.sqrt(2)
print(x)
```

PACKAGE

- Package is a collection of modules organized in a directory hierarchy.
- A package is simply a directory containing one or more Python modules, along with a special file called __init__.py.
- The __init__.py file is executed when the package is imported and can be used to perform initialization tasks or define package-level variables and functions.
- Packages are used to organize related modules into a single namespace, making it easier to manage and reuse code.
- By using packages, you can create a hierarchy of modules and sub packages, each with its own namespace.

BREAK

PACKAGE

- Packages are named using the dot notation, with each level of the hierarchy separated by a dot.
- For example, the numpy package is a popular package for scientific computing in Python, and it contains many subpackages and modules:

```
numpy
├── __init__.py
├── core
│   ├── __init__.py
│   ├── numeric.py
│   ├── shape_base.py
│   └── ...
├── fft
│   ├── __init__.py
│   ├── fft.py
│   └── ...
```

PACKAGE

- To use a package in Python, you can import it using the import statement, just like you would with a module. For example, to import the numpy package, you would use the following code:

```
import numpy
x = numpy.array([1, 2, 3])
print(x)
```

AGENDA-WW01D17

Modules in Detail

Packages in Detail

MODULES

- In addition, there are many third-party modules available, which can be installed using package managers such as pip.
- You can also import specific functions or variables from a module using the from keyword. For example, to import just the sqrt function from the math module, you would use the following code

```
from math import sqrt
root of 2
x = sqrt(2)
print(x)
```

BUILT-IN MODULES

os: Provides a way of using operating system dependent functionality like reading or writing to the file system, manipulating paths, etc.

sys: Provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.

datetime: Provides classes for working with dates and times.

math: Provides mathematical functions like sqrt, sin, cos, etc.

random: Provides functions for generating random numbers.

BUILT-IN MODULES

json: Provides functions for working with JSON (JavaScript Object Notation) data.

csv: Provides functions for working with CSV (Comma Separated Values) data.

re: Provides support for regular expressions (a powerful way of matching patterns in text).

time: Provides functions for working with time and dates.

urllib: Provides functions for working with URLs and HTTP.

collections: Provides a set of tools for working with collections of objects, like lists, tuples, and dictionaries.

MODULE ATTRIBUTES

Python module has its attributes that describes it. Attributes perform some tasks or contain some information about the module

AGENDA-WW01D19

Text File Operations

File handling in detail

OS MODULE

- os.getcwd()
- os.mkdir()
- os.chdir()
- os.remove()
- os.rmdir()

FILE HANDLING

There are 5 Major operations that can be performed on a file are:

- Creation of File
- Opening an existing file
- Reading data from a file
- Writing data in a file
- Closing a file

- Python too support file handling and allow users to handle files i.e., to read and write files, along with many other file handling, to operate on files.
- **Syntax :** open(filename, mode)

FILE OPERATIONS

- File Handling
  - **Mode for opening a file**:
  - “r”, for reading.
  - “w”, for writing
  - “a”, for appending
  - “r+”, for reading and writing

- **Create File:**

```
file = open(‘filename’, ‘w’)
file.write(data)
file.close()
```

FILE OPERATIONS

- **Read File:**

```
file = open(‘fielname’, ‘r’)
print(file.read())
file.close()
```

- **Append File:**

```
file = open(‘fielname’, ‘a’)
file.write(data)
file.close()
```

BREAK

FILE OPERATIONS

- **Using with() function:**
- **code:**

```
with open(filename) as file:
        data = file.read()
        print(data)
```

- The advantage is that the file is properly closed after its suite finishes, even if an exception is raised at some point.
- important function: readline(), write() etc.

- for more built-in functions visit here: https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

FILE OPERATIONS

- **readline():** It is used to read a single line from a file. When readline() is called repeatedly, it will read the file line by line until it reaches the end of the file.

```
with open('myfile.txt', 'r') as f:
    line = f.readline()
    while line:
        print(line)
        line = f.readline()
```

- **writeline()** is not a built-in function in Python. Instead, you can use the write() function to write a string to a file, and then add a newline character (\n) to the end of the string to create a new line.

```
with open('myfile.txt', 'w') as f:
    f.write('Line 1\n')
    f.write('Line 2\n')
    f.write('Line 3\n’)
```

FILE OPERATIONS

- **readlines()** : To read all the lines of a file in a list

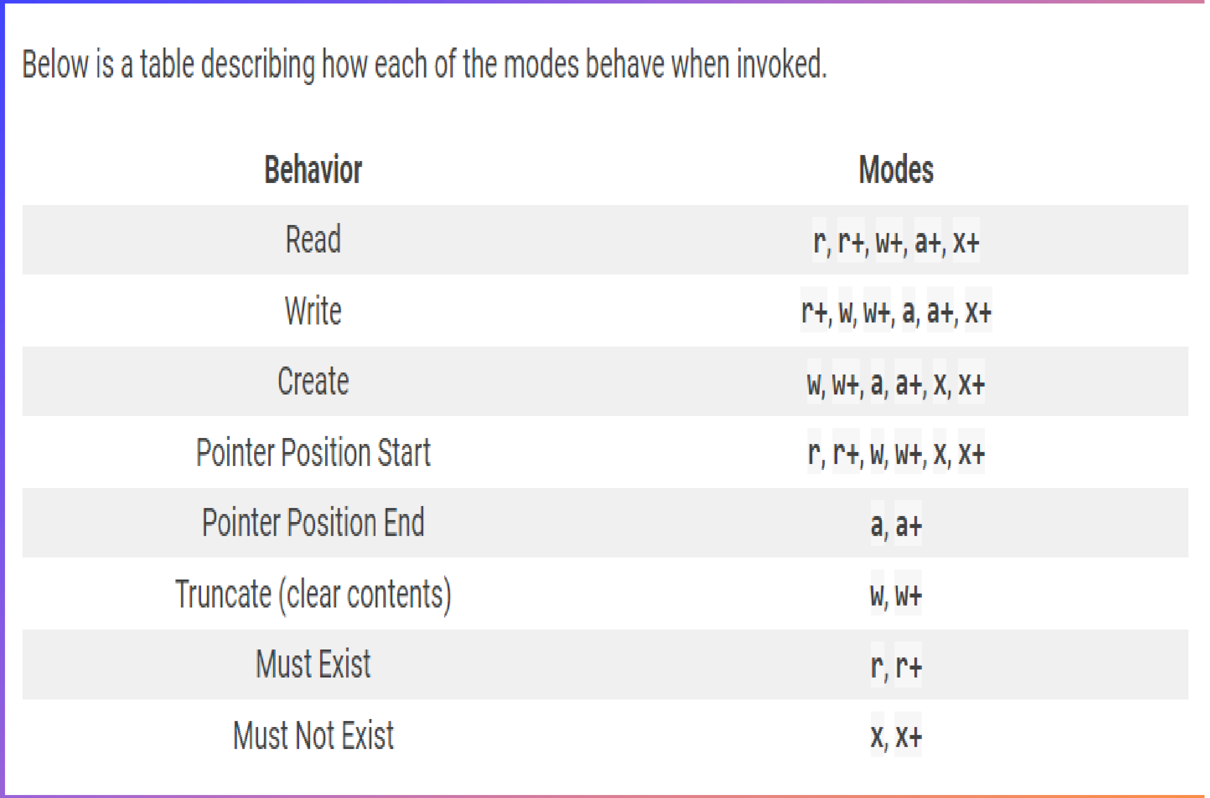
```
with open('myfile.txt', 'w') as f:
        f.readlines(lines)
```

- **writelines() :** To write multiple lines to a file. It takes an iterable as an argument and writes each element of the iterable to the file.

```
lines = ['Line 1\n', 'Line 2\n', 'Line 3\n']
with open('myfile.txt', 'w') as f:
        f.writelines(lines)
```

AGENDA-WW01D20

- Text File Operations
- File handling in detail
- Regex Introduction



Below is a table describing how each of the modes behave when invoked.

Behavior	Modes
Read	r, r+, w+, a+, x+
Write	r+, w, w+, a, a+, x+
Create	w, w+, a, a+, x, x+
Pointer Position Start	r, r+, w, w+, x, x+
Pointer Position End	a, a+
Truncate (clear contents)	w, w+
Must Exist	r, r+
Must Not Exist	x, x+

FILE OPERATIONS

- ‘x’ mode: Exclusive creation that fails if the file already exists.
- '+’ is used to indicate that the file should be opened for both reading and writing.
- ‘b’ :Binary mode. Use this mode for non-textual files

FILE OPERATIONS

- **seek()** – This function is used to change the position of the file pointer in a file. The syntax for the seek() function is:
  - file.seek(offset, whence) , whence means “from where”

0: The 0 value is used for setting the whence argument at the beginning of the file.

1: The 1 value is used for setting the whence argument at the current position of the file.

2: The 2 value is used for setting the whence argument at the end of the file.

Note: In text files (those opened without a b in the mode string), only seeks relative to the beginning of the file are allowed (the exception being seeking to the very file end with

seek(0, 2)) and the only valid offset values are those returned from the f.tell(), or zero

- **tell()** – This function is used to return the current position of the file pointer in a file.
  - Current position in the file represented as number of bytes from the beginning of the file when in binary mode and an opaque number when in text mode
  - position = file.tell()

- **flush()** – This function is used to flush the internal buffer of a file.

BREAK

Significance of 'r' before a string

- Adding the letter "r" before a string is used to indicate that it is a "raw string". This means that escape characters (such as "\n" for a newline) will be ignored and the string will be treated as-is.

- string = "This is a\nnew line."
  - **Output:**

```
This is a
new line.
```

- string = r"This is a\nnew line.“
  - Output:

```
This is a\nnew line.
```

REGEX

- A RegEx, or **Regular Expression**, is a sequence of characters that forms a search pattern.
    - Ex: “\d+”, “\s”
- RegEx can be used to check if a string contains the specified search pattern.

- **RegEx Module**
- Python has a built-in package called re, which can be used to work with Regular Expressions

- **Example:**
    - **re.search()**

      Used for searching a string for a specified pattern. It returns a match object if the pattern is found, otherwise, it returns None.

```
import re
string = "The quick brown fox jumps over the lazy dog.“
pattern = "fox“
result = re.search(pattern, string)
if result:
        print("Pattern found")
else:
        print("Pattern not found")
```

Special sequences -

- ‘\d’ - Matches any digit character
- ‘\s’ - Matches any whitespace character, including spaces, tabs, and newlines

- NOTE: When the + sign is used with a special sequence in a regular expression pattern, it means that the special sequence should be matched one or more times.
- Ex: ‘\d+’ , ‘\s+’

re module inbuilt functions

- **re.match**(): Matches the pattern at the beginning of the string.

```
import re
pattern = r"regular expression pattern"
string = "string to match"
match_object = re.match(pattern, string)
```

- **re.findall():** Returns a list of all non-overlapping matches of the pattern in the string.

```
import re
pattern = r"regular expression pattern"
string = "string to search"
matches = re.findall(pattern, string)
```

re module inbuilt functions

- **re.sub():** Replaces all occurrences of the pattern in the string with a specified replacement string.

```
import re
pattern = r"regular expression pattern"
string = "string to search"
replacement = "replacement string"
new_string = re.sub(pattern, replacement, string)
```

- **re.split():** Splits the string into a list of substrings at each occurrence of the pattern.

```
import re
pattern = r"regular expression pattern"
string = "string to split"
split_list = re.split(pattern, string)
```

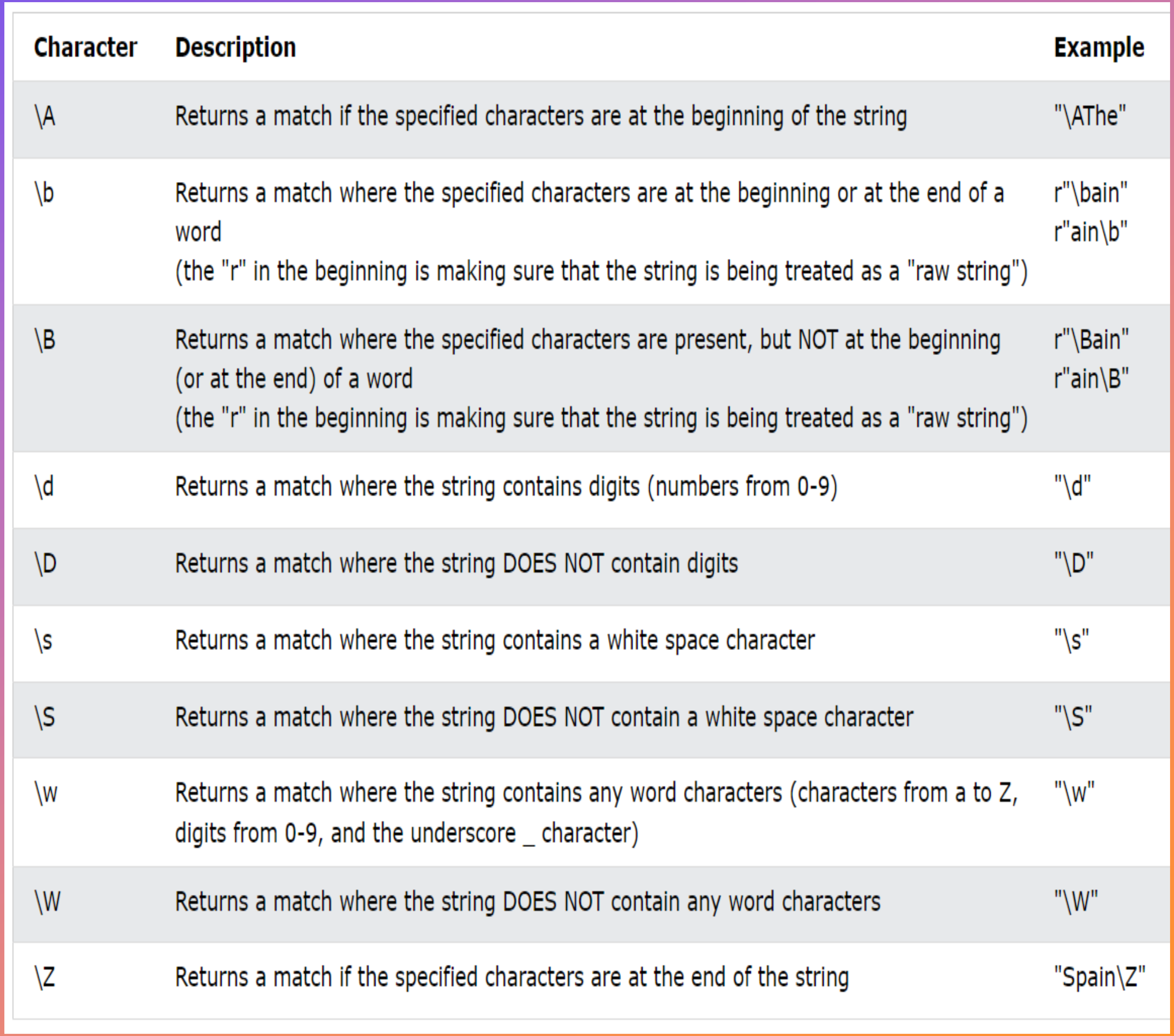
AGENDA-WW01D21

Regex in Detail

Special Sequences

SPECIAL SEQUENCES

Special sequences in Python regular expressions are a set of predefined patterns that match specific characters or character classes.



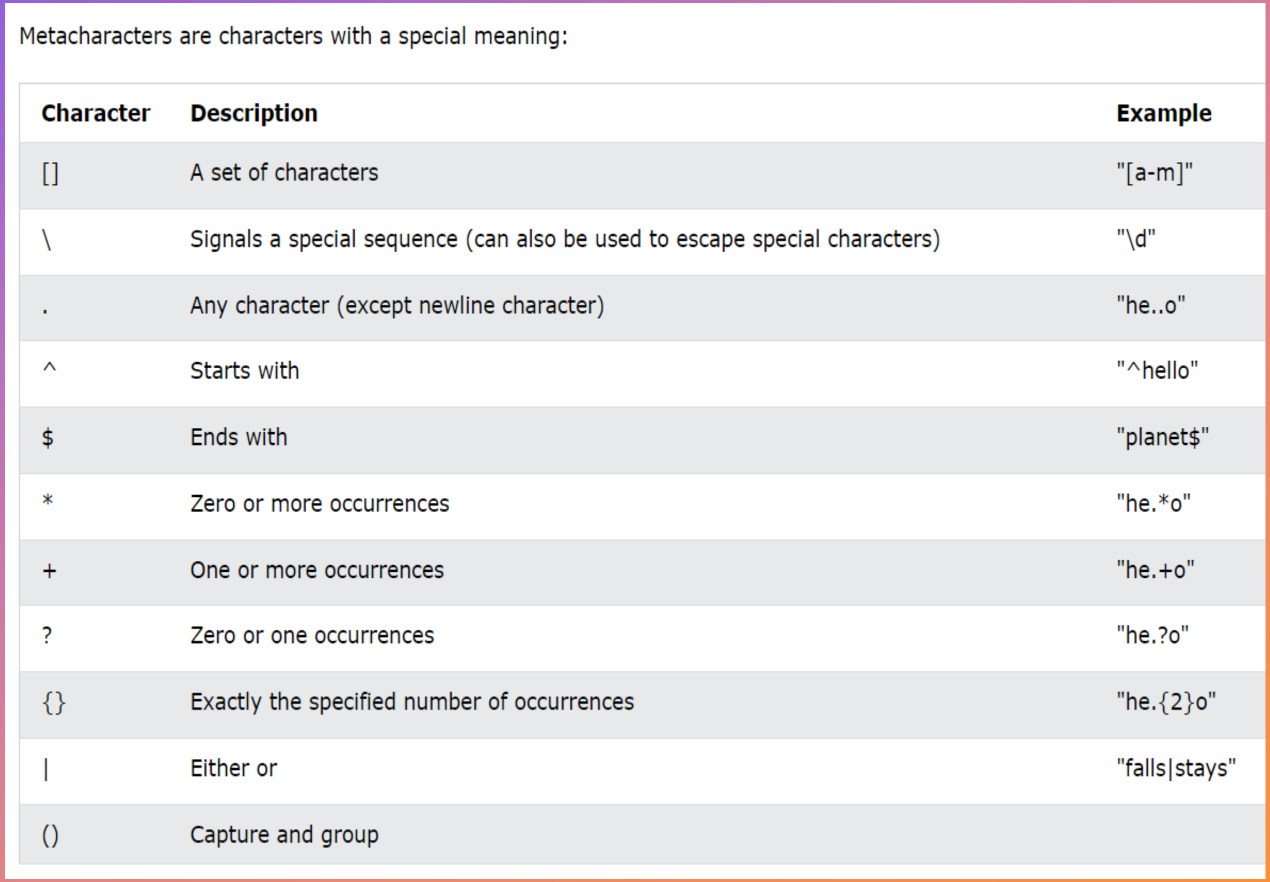
Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\Bain" r"ain\B"
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

AGENDA-WW01D22

- Regex Sets
- Regex Metacharacters
- Grouping and capturing

METACHARACTERS

Metacharacters are characters with a special meaning

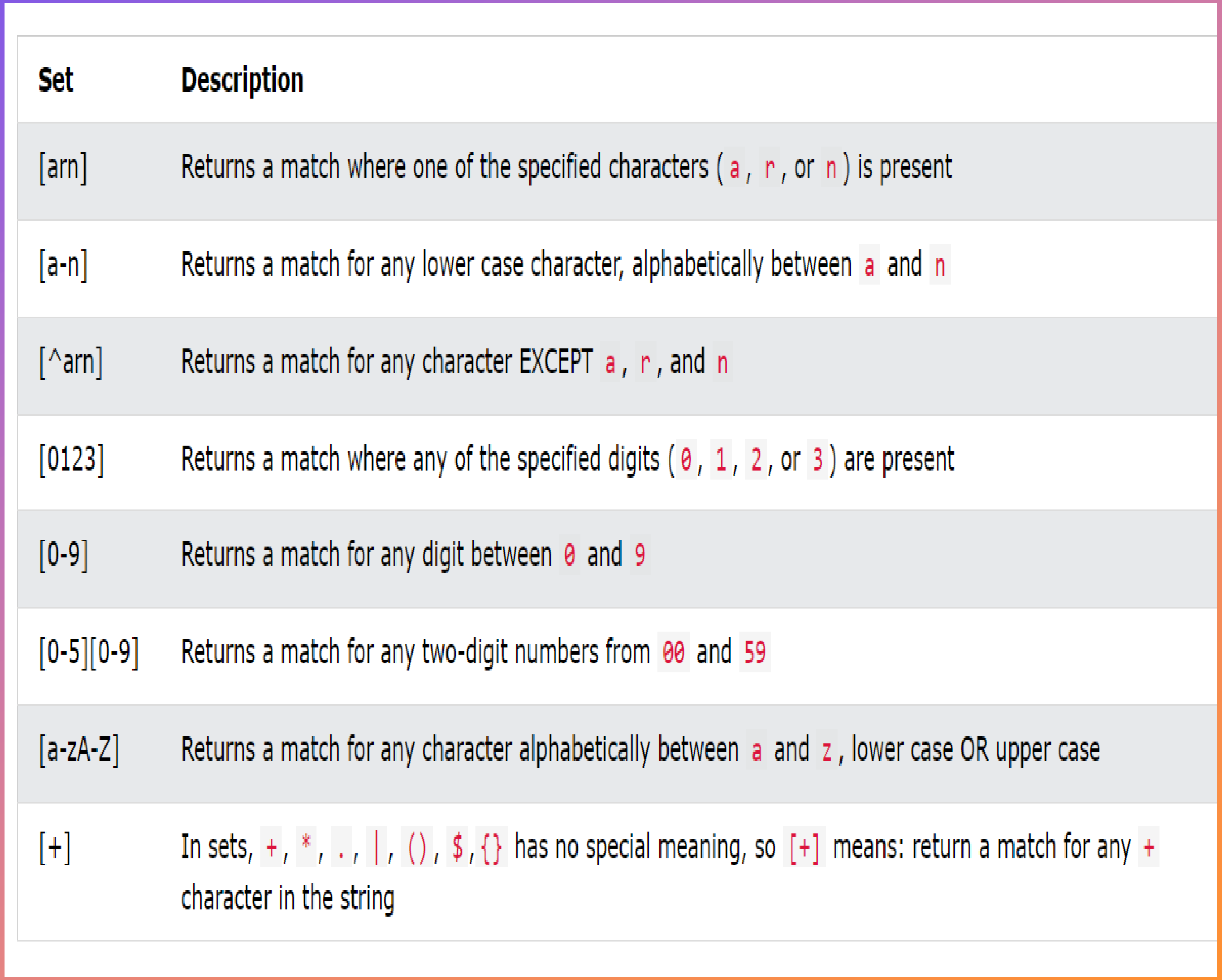


Metacharacters are characters with a special meaning:

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"planet$"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"
?	Zero or one occurrences	"he.?o"
{}	Exactly the specified number of occurrences	"he.{2}o"
|	Either or	"falls|stays"
()	Capture and group	

SETS

A set is a set of characters inside a pair of square brackets [] with a special meaning:



Set	Description
[arn]	Returns a match where one of the specified characters (`a`, `r`, or `n`) is present
[a-n]	Returns a match for any lower case character, alphabetically between `a` and `n`
[^arn]	Returns a match for any character EXCEPT `a`, `r`, and `n`
[0123]	Returns a match where any of the specified digits (`0`, `1`, `2`, or `3`) are present
[0-9]	Returns a match for any digit between `0` and `9`
[0-5][0-9]	Returns a match for any two-digit numbers from `00` and `59`
[a-zA-Z]	Returns a match for any character alphabetically between `a` and `z`, lower case OR upper case
[+]	In sets, `+`, `*`, `.`, `|`, `()`, `$`,`{}` has no special meaning, so `[+]` means: return a match for any `+` character in the string

BREAK

Grouping and capturing

Grouping and capturing in Python regular expressions allow you to create subgroups within a regular expression and capture the matched text in those subgroup

- Using **parentheses** to create subgroups.

```
string = "The price of an apple is $0.75 and the price of a banana is $0.50."
pattern = r"The price of (an|a) (\w+) is \$(\d+\.\d{2})“
```

AGENDA-WW01D23

- map
- filter
- lambda

MAP

- The **map()** function takes a function and applies it to each element of an iterable (e.g., list, tuple, etc.) and returns a new iterable with the results.

- **Syntax:**

```
map(function, iterable)
```

- **For example,** if we want to square each element in a list of numbers, we could use the map() function as follows:

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = map(lambda x: x**2, numbers)
print(list(squared_numbers)) # Output: [1, 4, 9, 16, 25]
```

FILTER

- The **filter()** function takes a function and applies it to each element of an iterable, returning only the elements for which the function returns True.
- **Syntax:**

  ```
  filter(function, iterable)
  ```

- **For example**, if we want to filter out all the even numbers in a list, we could use the filter() function as follows:

  ```
  numbers = [1, 2, 3, 4, 5]
  even_numbers = filter(lambda x: x % 2 == 0, numbers)
  print(list(even_numbers)) # Output: [2, 4]
  ```

LAMBDA

- A **lambda** function is a small anonymous function that can have any number of arguments but can only have one expression.

- Lambda functions are often used as arguments to higher-order functions like map() and filter()

- **Syntax:**

  lambda arguments: expression

- **For example**, we can define a lambda function that squares a number as follows:

```
square = lambda x: x**2
print(square(5))
# Output: 25
```

AGENDA-WW01D24

- lambda
- List comprehension
- Practice

Revision/Doubts

1- Function return statement

2- Matrix Problem

BREAK

LIST COMPREHENSION

- **List comprehension** is a concise way of creating a new list in Python by applying an expression to each item in an existing list or iterable.

- **Syntax**:

  new_list = [expression for item in iterable if condition]

  where:

  - **new_list** is the name of the new list being created
  - **expression** is the operation to be performed on each item in the iterable
  - **item** is the current item being processed in the iterable
  - **iterable** is the original list or other iterable to be processed
  - **if condition** is an optional condition that filters the items in the iterable before processing them

- **Example:**

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = [x for x in numbers if x % 2 == 0]
print(even_numbers) # Output: [2, 4, 6, 8, 10]
```

Difference/Usecase

List comprehension is used to create new lists based on existing lists, lambda functions are used to define small anonymous functions, map is used to apply a function to every element in a list, and filter is used to extract elements from a list based on a condition

AGENDA-WW01D25

Live Project

Implementing a project

Understand the project requirements:

Before you start writing any code, make sure you understand what the project is asking for. Read the project instructions carefully and take note of any specific requirements, such as what functions you need to create, what input/output formats you need to use, and any limitations or constraints.

Plan your approach:

Once you have a clear understanding of the requirements, plan out how you will approach the project. Think about what modules, libraries, and data structures you may need to use, and how you will break the project down into smaller tasks. Consider writing an outline or pseudocode to help you organize your thoughts.

Write the code:

With a plan in place, start writing your code. Begin with the simplest tasks first, and work your way up to more complex ones. Make sure to test your code frequently as you go, to catch any errors early on.

Implementing a project

Debug and troubleshoot:

As you test your code, you will likely encounter errors and bugs. Use the error messages and debugging tools available in Python to identify and fix these issues.

Refactor and optimize:

Once your code is working correctly, look for ways to improve it. This may involve refactoring your code to make it more efficient, or optimizing it to reduce memory usage or improve runtime performance.

Document and test:

Finally, make sure to document your code thoroughly, so that others can understand and use it. Write comments, docstrings, and README files to explain what your code does and how to use it. Test your code thoroughly to ensure it is working as expected and meets the project requirements.

http://blackjack-final.appbrewery.repl.run

FACTS

1. The deck is unlimited in size.
2. There are no jokers.
3. The Jack/Queen/King all count as 10.
4. The Ace can count as 11 or 1.
5. Use the following list as the deck of cards:
6. cards = [11, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10]
7. The cards in the list have equal probability of being drawn.
8. Cards are not removed from the deck as they are drawn.
9. The computer is the dealer.

RULES

1.Deal both user and computer a starting hand of 2 random card values.

2.Detect when computer or user has a blackjack. (Ace + 10 value card).

3.If computer gets blackjack, then the user loses (even if the user also has a blackjack
If the user gets a blackjack, then they win (unless the computer also has a blackjack).

4.Calculate the user's and computer's scores based on their card values.

5.Game ends immediately when user score goes over 21 or if the user or computer gets a blackjack.

6. Ask the user if they want to get another card.

7. Let the computer play.
The computer should keep drawing cards unless their score goes over 16.

8. Compare user and computer scores and see if it's a win, loss, or draw.

9. Print out the player's and computer's final hand and their scores at the end of the game.

AGENDA-WW01D26

- Why OOP
- What is OOP
- Objects and classes
- Attributes and methods

WHY OOP

- Complexity because of Procedural Programming .
- Complexity leads to confusion.
- Code manageability becomes tough
- Parallel coding when multiple members of a team working on same project.
- Ex: A restaurant

WHAT IS OOP

- OOP is a powerful programming model for software development because it makes it easier to write, maintain, and reuse code.
- OOP allows programmers to create their own objects that have methods and attributes.
- In general , OOP allows us to create code that is repeatable and organized.

OBJECTS AND CLASSES

What is **class**?

- A class is a blueprint or a template that defines the attributes and behaviors of a certain type of object.

```
class Car:
    pass
```

What is **Object** ?

- An object is an instance of a class. When we create an object, we are creating a unique instance of that class, with its own set of attributes and behaviors.

```
my_car = Car()
```

- Multiple objects of a class can be created.

ATTRIBUTES

What is **attributes**?

- An attribute is a characteristic of an object.
- In general variables are attributes.

- The syntax for creating an attribute is:

```
self.attribute = something
```

- There is a special method called:

```
__init__()
```

- This method is used to initialize the attributes of an object.

For example:

```
self.color = color
self.speed = 0
```

BREAK

METHODS

What is **methods**?

- A method is the behavior of the object.
- In general functions are methods.
- They are used to perform operations with the attributes of our objects.

Example:

```
def accelerate(self, acceleration):
        self.speed += acceleration
```

AGENDA-WW01D27

__init__ method

self keyword

Practice

Class Attributes Vs Object Attributes

__init__()

- **__init__ method** is a special method that is automatically called when an object of a class is created.
- It is used to initialize the object's attributes or properties.
- The __init__ method is often referred to as the "constructor" method because it is similar to the constructor method in other programming languages.
- However, it is important to note that __init__ is not a true constructor, as the object has already been created before __init__ is called.

self

- The self keyword is used to refer to the instance of a class. It is a reference to the current object that the method or attribute is being called on.

- When you create a class in Python, you can define methods and attributes that are associated with instances of that class.

- When you create an instance of the class, you can then call those methods and access those attributes using the self keyword.

PRACTIC
E

Class attributes vs object attributes

- **Class attributes** are attributes that are defined at the class level and are shared by all instances of the class.
- They are defined outside of any method and are accessed using the class name.

- **Object attributes** are attributes that are defined at the object level and are specific to each instance of the class.
- They are defined inside the class methods and are accessed using the instance name.

AGENDA-WW01D28

getters and setters

•Characteristics of OOPs

Encapsulation

Polymorphism

Class Methods

Getter and setter

- In python there is no strict concept of **private attributes** like in some other programming languages. However, we can use a convention to make an attribute private by adding a double underscore (__) prefix to its name.

- **Getter and setter methods** are used to access and modify class attributes in a controlled way. Getter and setter methods are also known as accessor and mutator methods, respectively.

Getter Method:

- A getter method is used to retrieve the value of a class attribute. In Python, getter methods are implemented using the @**property** decorator.

Setter Method:

- A setter method is used to modify the value of a class attribute. In Python, setter methods are implemented using the @**<attribute_name>.setter decorator**.

CHARACTERISTICS OF OOP

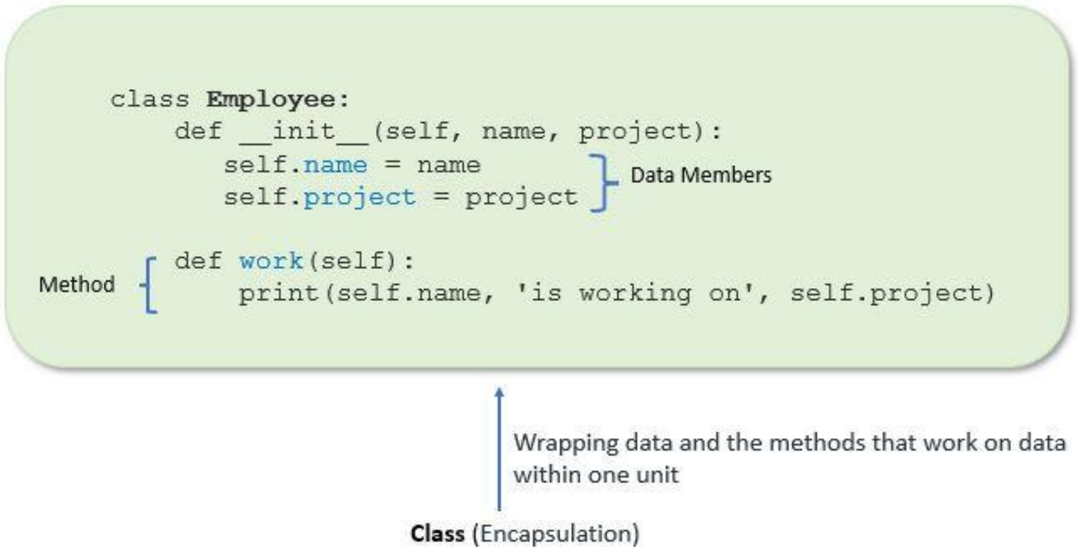
- **Inheritance**: Inheritance is a way to form new classes using classes that have already been defined. The newly formed classes are called derived classes, the classes that we derive from are called base classes.

- **Polymorphism**: Polymorphism refers to the way in which different object classes can share the same method name, and those methods can be called from the same place

- **Modularity**: OOP allows developers to break down complex software systems into smaller, more manageable pieces called objects. This makes it easier to write, maintain, and reuse code.

- **Encapsulation**: OOP encourages developers to hide the internal workings of an object from the outside world. This means that changes to an object's implementation won't affect other parts of the system that use the object.

ENCAPSULATION



```
class Employee:
    def __init__(self, name, project):
        self.name = name
        self.project = project

    def work(self):
        print(self.name, 'is working on', self.project)
```

Data Members
Method
Wrapping data and the methods that work on data within one unit
Class (Encapsulation)

- It that enables the **bundling of data and methods inside a single unit**, preventing access to the internal data and methods from outside the unit, except through the public interface provided by the unit.
- In Python, encapsulation can be achieved by using private and public access modifiers.

- **Advantages of Encapsulation**
  - Security
  - Data Hiding
  - Simplicity

BREAK

POLYMORPHISM

- Polymorphism in Python is the ability of an object to take many forms. In simple words, polymorphism allows us to perform the same action in many different ways

- In polymorphism, a method can process objects differently depending on the class type or data type. Let’s see simple examples to understand it better

Example:

The built-in function **len()** calculates the length of an object depending upon its type. If an object is a string, it returns the count of characters, and If an object is a list, it returns the count of items in a list.

- Polymorphism is mainly used with inheritance.

Class Methods

- **Class methods** are special methods that are bound to the class and not to the instance of the class.
- Class methods are defined using the **@classmethod** decorator.
- **Class methods u**sed to access or modify the class state.
- In method implementation, if we use only class variables, then such type of methods we should declare as a class method.
- The class method has a **cls** parameter which refers to the class.
- Class method is also used as **factory method** to create object.

AGENDA-WW01D29

•Static methods

Dunder or magic methods

•Operator overloading

Practice

static Methods

- A static method is a general utility method that performs a task in isolation.

Inside this method, we **don’t use instance or class variable** because this static method **doesn’t take any parameters like self and cls**.

- A static method is a method that belongs to a class rather than an instance of the class. It can be called without creating an instance of the class and operates on the class itself rather than an instance of the class.

- To create a static method in Python, you can use the @staticmethod decorator.

- **Uses:** A utility method that performs some calculation or formatting could be implemented as a static method.

Letter - f

- Letter 'f' can be used to create a formatted string literal.
- An f-string is a way to embed expressions inside string literals, **using curly braces {}.**
- The syntax for f-strings is to prefix the string literal with the letter 'f', and include expressions inside curly braces {}.
- The expressions are evaluated at runtime and their values are converted to strings and inserted into the string literal at the location of the corresponding curly braces.

Dunder/magic Methods

- Dunder methods (short for "double underscore" methods) in Python are special methods that are surrounded by double underscores on both sides of the method name.
- These are used to define various built-in operations and behaviors for objects in Python.
- These methods are not actually called directly but by Python specific language syntax

- **__init__(self, ...):** The constructor method that gets called when an object is created. It is used to initialize the object's attributes.

- **__str__(self):** A method that returns a string representation of the object when the **str()** function is called on it.

- **__del__(self):** It is used to define the behavior of an object when it is about to be destroyed, or deallocated, from memory. The __del__() method is automatically called by Python's garbage collector when an object is no longer being used or referenced by any other object in the program. Explicitly called using **del**

- **__len__(self):** A method that returns the length of the object when the **len()** function is called on it.

BREAK

Operator overloading

- In Python, we can change the way operators work for **user-defined types**.

- **For example, the + operator** will perform arithmetic addition on two numbers, merge two lists, or concatenate two strings.

- This feature in Python that allows the same operator to have different meaning according to the context is called **operator overloading**.

AGENDA-WW01D30

- Inheritance
- Types of Inheritance
- super(), issubclass()
- Method Overriding
- Practice

INHERITANCE

- The process of inheriting the properties of the **parent class** into a **child class** is called inheritance.
- The existing class is called a base class or parent class and the new class is called a subclass or child class or derived class.
- The main purpose of inheritance is the reusability of code because we can use the existing class to create a new class instead of creating it from scratch.
- In inheritance, the child class acquires all the data members, properties, and functions from the parent class. Also, a child class can also provide its specific implementation to the methods of the parent class.
- **Syntax:**

```
class BaseClass:
        Body of base class
class DerivedClass(BaseClass):
        Body of derived class
```

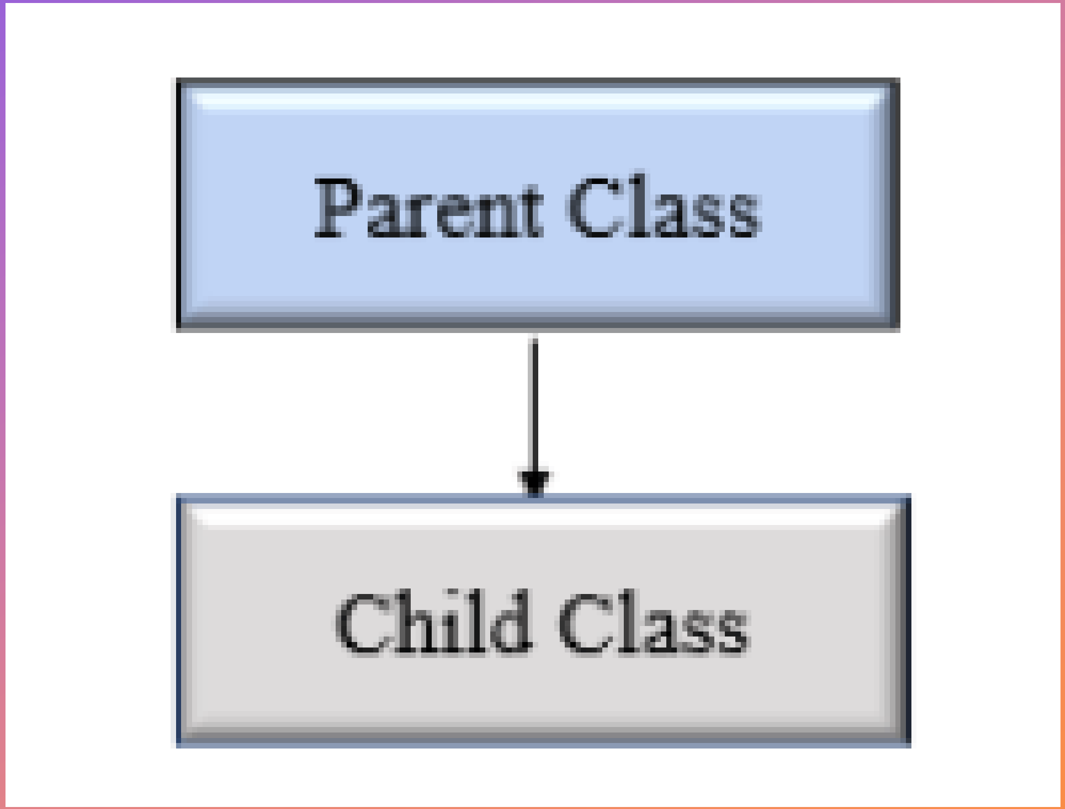
TYPES OF INHERITANCE

- In Python, based upon the number of child and parent classes involved, there are five types of inheritance.

  - Single inheritance
  - Multiple Inheritance
  - Multilevel inheritance
  - Hierarchical Inheritance
  - Hybrid Inheritance

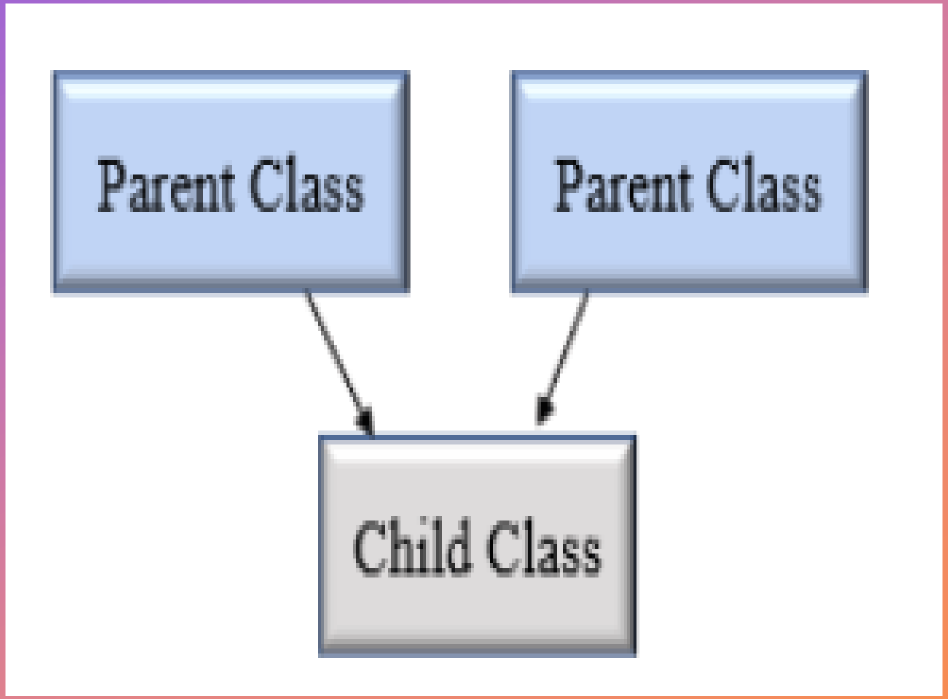
SINGLE INHERITANCE

- In single inheritance, a child class inherits from a single-parent class. Here is one child class and one parent class.



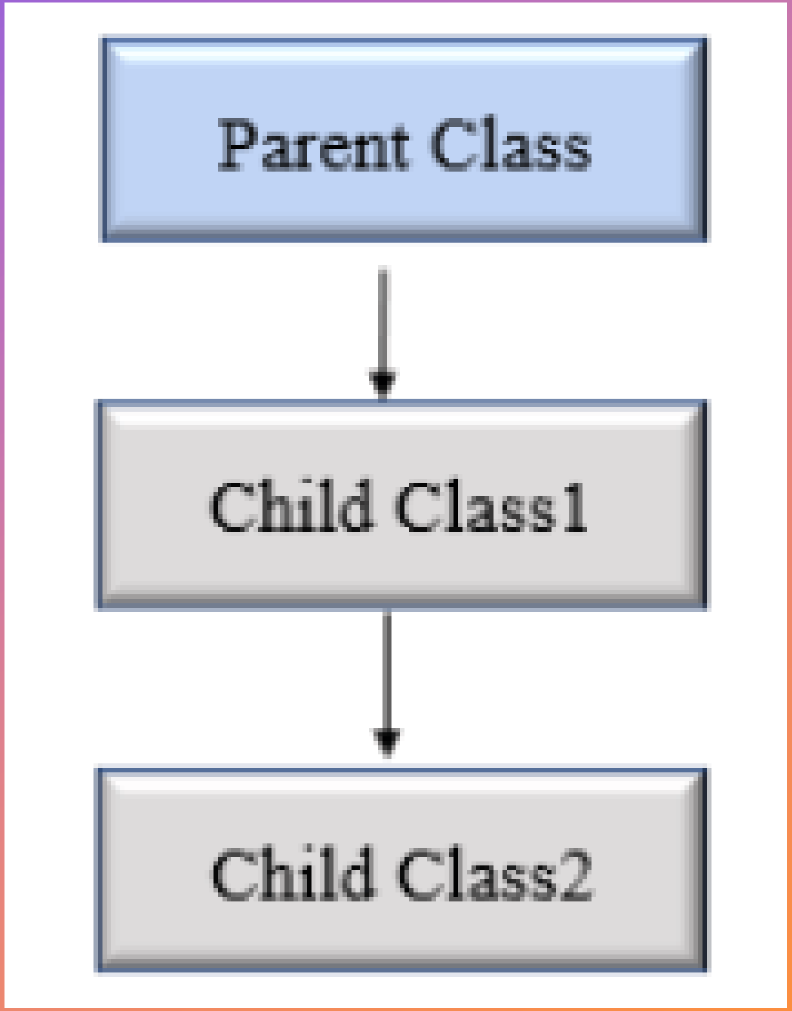
MULTIPLE INHERITANCE

- In multiple inheritance, one child class can inherit from multiple parent classes.
- So here is one child class and multiple parent classes.



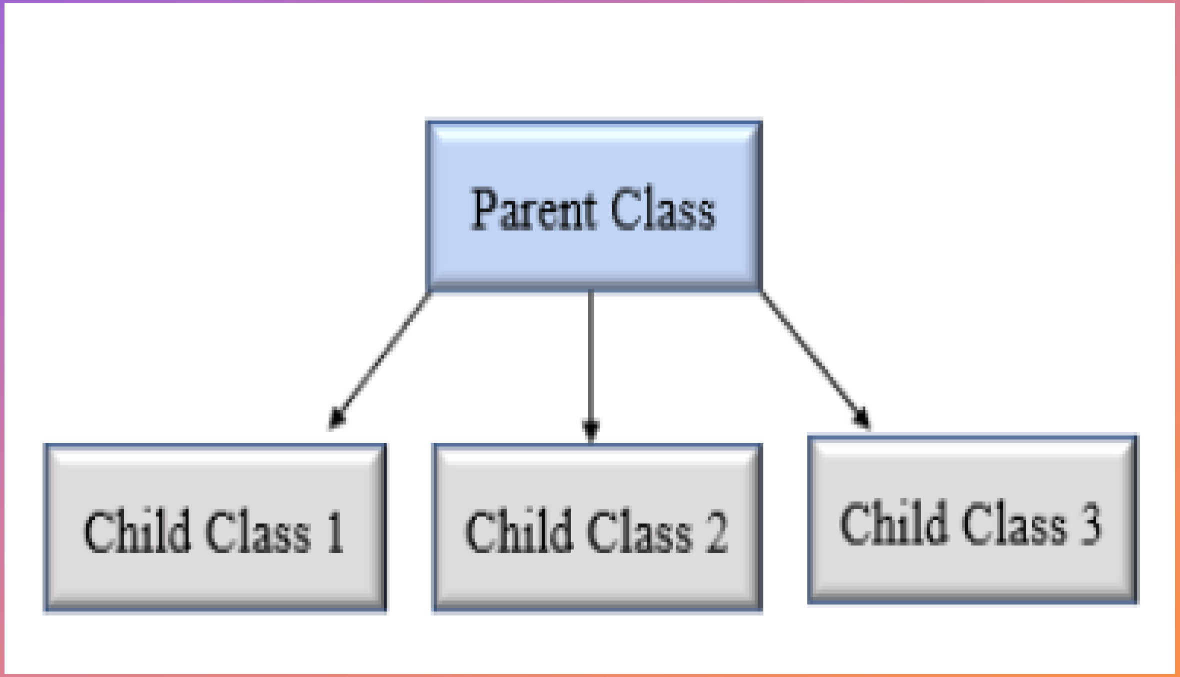
MUTILEVEL INHERITANCE

- In multilevel inheritance, a class inherits from a child class or derived class.
- Suppose three classes A, B, C. A is the superclass, B is the child class of A, C is the child class of B.
- In other words, we can say a chain of classes is called multilevel inheritance.



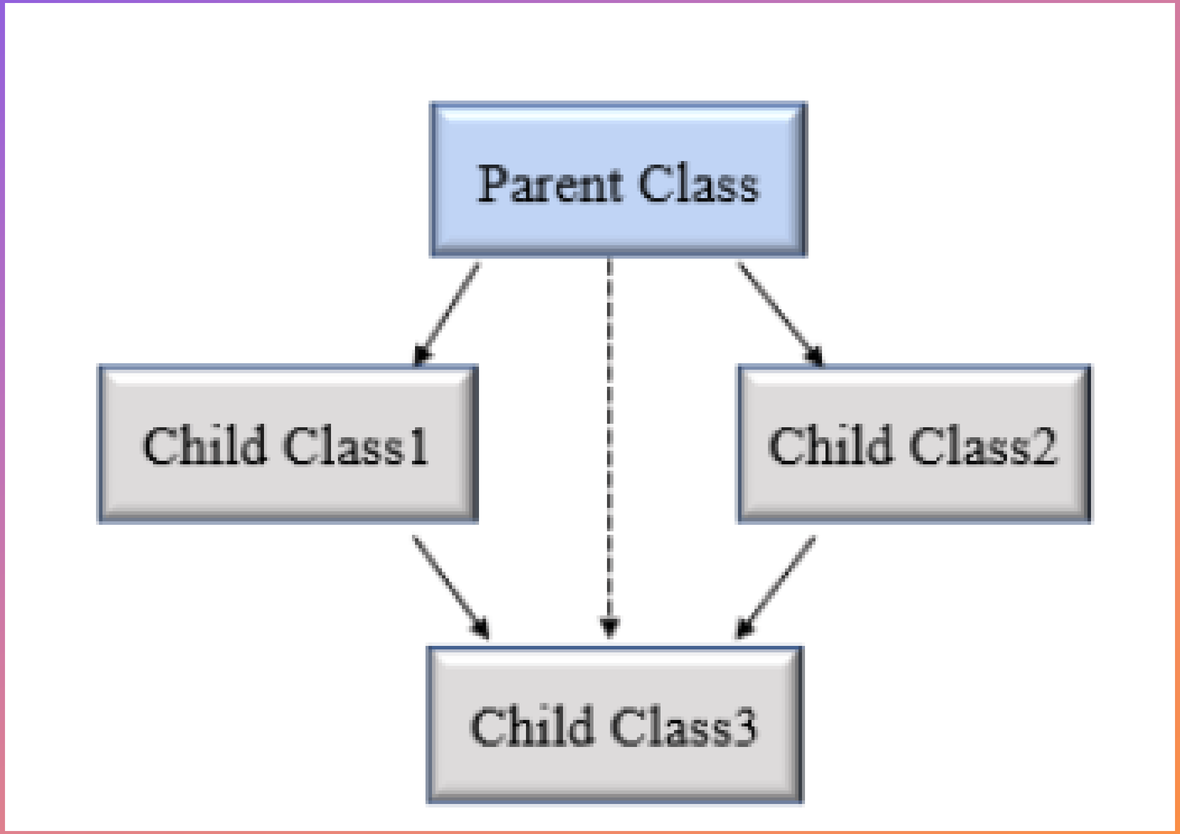
HIERARCHICAL INHERITANCE

- In Hierarchical inheritance, more than one child class is derived from a single parent class.
- In other words, we can say one parent class and multiple child classes.



HYBRID INHERITANCE

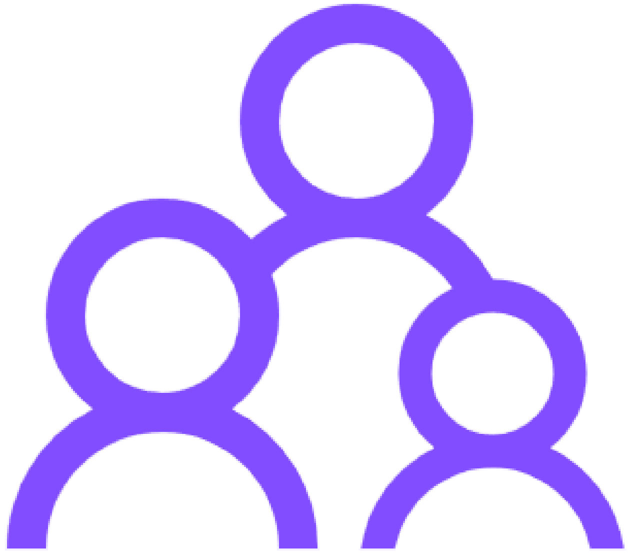
- When inheritance is consists of multiple types or a combination of different inheritance is called hybrid inheritance.



BREAK

super()

- We can refer to parent class by using the super() function.
- The super function returns a temporary object of the parent class that allows us to call a parent class method inside a child class method.



issubclass()

- In Python, we can verify whether a particular class is a subclass of another class.
- For this purpose, we can use Python built-in function issubclass()
- This function returns True if the given class is the subclass of the specified class. Otherwise, it returns False.

METHOD OVERRIDING

- In inheritance, all members available in the parent class are by default available in the child class.
- If the child class does not satisfy with parent class implementation, then the child class is allowed to redefine that method by extending additional functions in the child class. This concept is called method overriding.
- When a child class method has the same name, same parameters, and same return type as a method in its superclass, then the method in the child is said to override the method in the parent class.

AGENDA-WW01D31

- Unit Test
- Debugging

Unit Test

- A unit test is a type of software testing that involves testing individual units or components of a software application in isolation from the rest of the system.
- The goal of unit testing is to verify that each unit of code behaves as expected and produces the correct output for a given input or set of inputs.
- In the context of object-oriented programming, a unit typically refers to a single method or function in a class or module.
- Unit tests are typically written by developers as they write their code, to help ensure that each unit of code behaves as intended and to catch any bugs or errors early in the development process.

TEST FRAMEWORK IN PYTHON

- Python provides several built-in unit testing frameworks that can be used to write and run unit tests for Python code.
    - unittest
    - nose Or nose2
    - Pytest

- The most used unit testing framework in Python is the unittest module, which is part of the Python standard library.

- The unittest framework provides a set of tools and methods for defining and running unit tests in Python, including **test discovery, test fixtures, and assertion methods.**

UNIT TEST FRAMEWORK IN PYTHON

Here is a basic overview of how to use the unittest framework:

- Create a test case class that inherits from **unittest.TestCase**.
- Define test methods in the test case class. **Test methods should start with the word "test".**
- Use assertion methods in the test methods to check that the code being tested is behaving as expected.
- **Use a test runner to discover and run your tests. You can use the unittest.main() function to run all the tests in a module or the unittest.**TextTestRunner class to run tests in a more customizable way.

- Example:

```
import unittest
class MyTestCase(unittest.TestCase):
    def test_addition(self):
        self.assertEqual(2 + 2, 4)

    def test_subtraction(self):
        self.assertEqual(5 - 3, 2)

unittest.main()
```

USEFUL METHODS OF UNITTEST

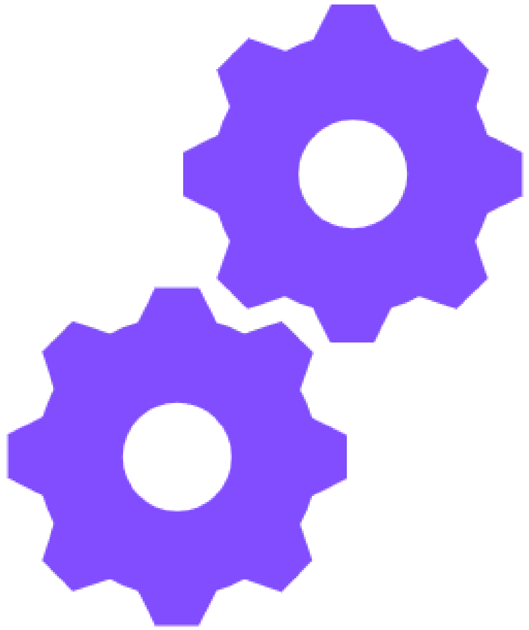
- **assertEqual():** This method checks if two values are equal. If the values are not equal, it will raise an AssertionError with a specified message.
- **assertTrue() and assertFalse():** These methods check if a value is True or False, respectively.
- **assertIn() and assertNotIn():** These methods check if a value is present or absent in a list, tuple, or set.
- **assertRaises():** This method checks if an exception is raised when a particular function is called.
- **setUp() and tearDown():** These methods are called before and after each test method, respectively. They can be used to set up any necessary resources before running the tests and to clean up after the tests are finished

BREAK

Debugging

Debugging is the process of finding and fixing errors or bugs in your code.

- **Print Statements**: The most basic debugging technique is to use print statements in your code to display the values of variables or expressions at different points in your code.
- **pdb:** The Python debugger, or pdb, is a built-in tool that allows you to step through your code line by line and inspect variables and expressions as you go.
- **IDE Debugging:** Many integrated development environments (IDEs) for Python, such as PyCharm or Visual Studio Code, provide advanced debugging features, such as breakpoints, variable inspection, and call stacks.
- **Logging:** The Python logging module allows you to record messages at different levels of severity and output them to a file or console, providing a more systematic and structured approach to debugging.
- **Tracebacks:** When an error or exception occurs in your code, Python generates a traceback that shows the line number and file name of the error, as well as the call stack leading up to the error.



pdb

- pdb (Python Debugger) is a built-in module in Python that allows you to interactively debug your code.
- It can be used to step through **your code, set breakpoints, inspect variables, and more**.
- Here are the basic steps for using pdb to debug your Python code:
- Import the pdb module at the beginning of your script.
- **Set a breakpoint in your code by calling the pdb.set_trace() function** at the point where you want to start debugging.
- Run your script as usual.
- When the pdb.set_trace() line is reached during execution, the debugger will start and you will be able to interactively debug your code.
- Use **the n command** to execute the current line and move to the next line, **the s command to step into a function call**, **the r command to continue execution** until the current function returns, and the **c command to continue execution until the next breakpoint is reached.**
- Use the **p command to print the value of a variable**, and the **q command to quit the debugger and exit the script.**

AGENDA-WW01D32

- Exception Handling
- Generators
- Practice

Exception

- In programming, an exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.
- When an exceptional condition arises, such as an error or unexpected behavior, the program may stop executing, raise an error message, or take some other action to handle the situation.
- In Python, exceptions are represented by objects that inherit from the built-in Exception class.
- When an exceptional condition occurs in Python, an exception object is created and raised.
- This causes the Python interpreter to search for an exception handler, which is a block of code that can handle the exception and continue executing the program.

DIFFERENT TYPES OF EXCEPTION

- **SyntaxError**: This exception is raised when the Python interpreter encounters a syntax error in your code.
- **TypeError**: This exception is raised when you try to perform an operation on an object of the wrong type.
- **ValueError**: This exception is raised when you pass an argument of the correct type but with an inappropriate value.
- **ZeroDivisionError**: This exception is raised when you attempt to divide a number by zero.
- **IndexError**: This exception is raised when you try to access an index that is out of range in a sequence like a list or a string.
- **KeyError**: This exception is raised when you try to access a non-existent key in a dictionary.
- **AttributeError**: This exception is raised when you try to access an attribute of an object that does not exist.
- **FileNotFoundError**: This exception is raised when you try to access a file that does not exist.
- **IOError**: This exception is raised when an input/output operation fails, such as when you try to read from a file that is not open.
- **KeyboardInterrupt**: This exception is raised when the user interrupts the execution of the program by pressing Ctrl+C.

EXCEPTION HANDLING

- Exception handling is a mechanism in Python that allows you to handle errors that occur during the execution of a program.
- Exceptions are raised when an error occurs in a Python program, and the interpreter is unable to continue the execution of the program.
- To handle exceptions in Python, you use a try-except block.
- The try block contains the code that might raise an exception, and the except block contains the code that handles the exception. Here's an example:

```
try:
    # code that might raise an exception
except ExceptionType:
    # code to handle the exception
```

BREAK

Generators

- A generator function is a special type of function that allows you to iterate over a sequence of values without generating the entire sequence in memory at once.

- Instead, the generator function yields a new value each time it is called, allowing you to generate the sequence on the fly.

- A generator function is defined using the yield keyword instead of the return keyword.

- When a generator function is called, it returns a generator object that you can use to iterate over the sequence of values.

- Syntax:

```
def my_generator_function():
    # Generator function body
    yield value1
    yield value2
    ...
    yield valueN
```

next()

- next() function is used to retrieve the next value from a generator.
- When a generator is called with next(), it resumes execution from where it left off and continues until it encounters the next yield statement.
- The value yielded by that yield statement is returned by the next() function.

AGENDA-WW01D33

- CSV File Handling
- JSON File Handling
- XML File handling

CSV FILE HANDLING

- A CSV (Comma Separated Values) format is one of the most simple and common ways to store tabular data. To represent a CSV file, it must be saved with the .csv file extension.

- If you open the above CSV file using a text editor such as notepad, you will see:

```
SN, Name, City
1, Michael, New Jersey
2, Jack, California
```

- While we could use the built-in open() function to work with CSV files in Python, there is a dedicated csv module that makes working with CSV files much easier.

csv module

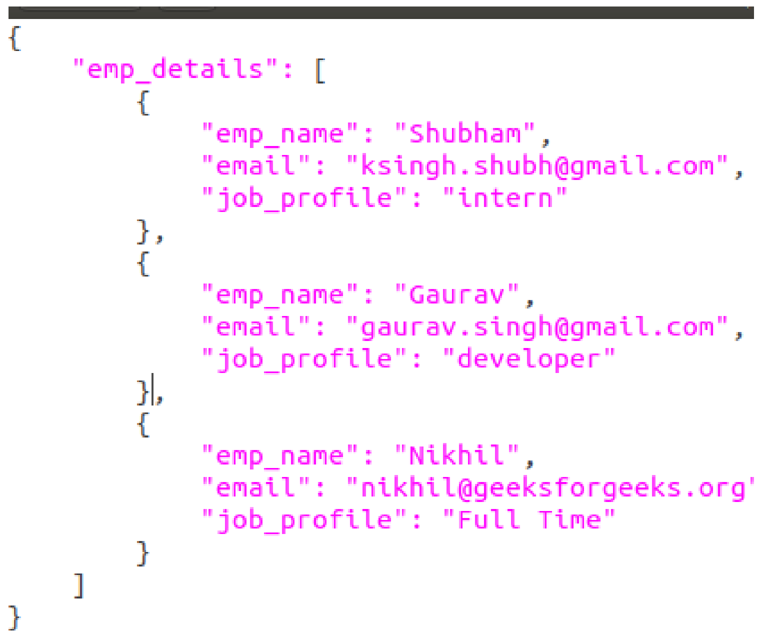
- **csv.writer:** This function returns an object that can be used to write data to a CSV file.
- **csv.reader:** This function returns an object that can be used to read a CSV file line by line, where each line is split into a list of fields.
- **csv.DictReader:** This function returns an object that can be used to read a CSV file line by line, where each line is parsed into a dictionary with the column headers as keys and the field values as values
- **csv.DictWriter:** This function returns an object that can be used to write data to a CSV file, where each row is a dictionary with the column headers as keys and the field values as values.

JSON FILE HANDLING

- The full-form of JSON is JavaScript Object Notation. It means that a script (executable) file which is made of text in a programming language, is used to store and transfer the data.

- Python supports JSON through a built-in package called json. To use this feature, we import the json package in Python script.

- The text in JSON is done through quoted-string which contains the value in key-value mapping within { }.



```
{
    "emp_details": [
        {
            "emp_name": "Shubham",
            "email": "ksingh.shubh@gmail.com",
            "job_profile": "intern"
        },
        {
            "emp_name": "Gaurav",
            "email": "gaurav.singh@gmail.com",
            "job_profile": "developer"
        },
        {
            "emp_name": "Nikhil",
            "email": "nikhil@geeksforgeeks.org'
            "job_profile": "Full Time"
        }
    ]
}
```

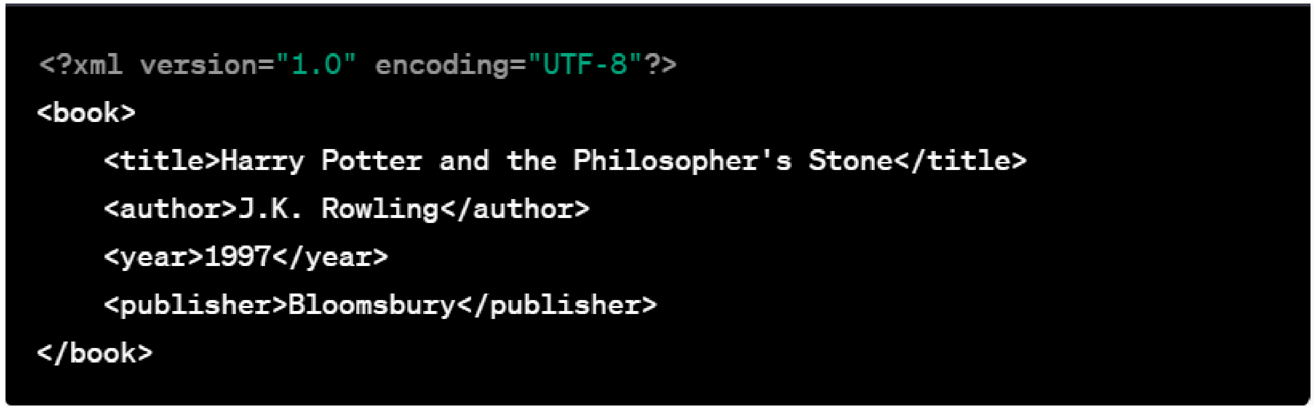
json module

- json.loads: This function converts a JSON string into a Python object.
- json.dumps: This function converts a Python object into a JSON string.
- json.load: This function reads a JSON file and returns a Python object.
- json.dump: This function writes a Python object to a JSON file

BREAK

XML FILE HANDLING

- Python has a built-in module called xml for parsing and handling XML files.
- Import the xml.etree.ElementTree module: The ElementTree module is part of the xml module and provides a simple way to parse and manipulate XML files in Python.
- Load the XML file: Use the ElementTree.parse() method to load the XML file into a tree structure.
- Access the elements of the XML file: You can access the elements of the XML file using the Element object
- Modify the XML file: You can modify the XML file by manipulating the Element objects. For example, to add a new child element to the root element
- Save the modified XML file: Use the ElementTree.write() method to save the modified XML file.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<book>
    <title>Harry Potter and the Philosopher's Stone</title>
    <author>J.K. Rowling</author>
    <year>1997</year>
    <publisher>Bloomsbury</publisher>
</book>
```

AGENDA-WW01D34

- Request Module
- BeautifulSoup
- Data Extraction using fin, find_all

Request Module

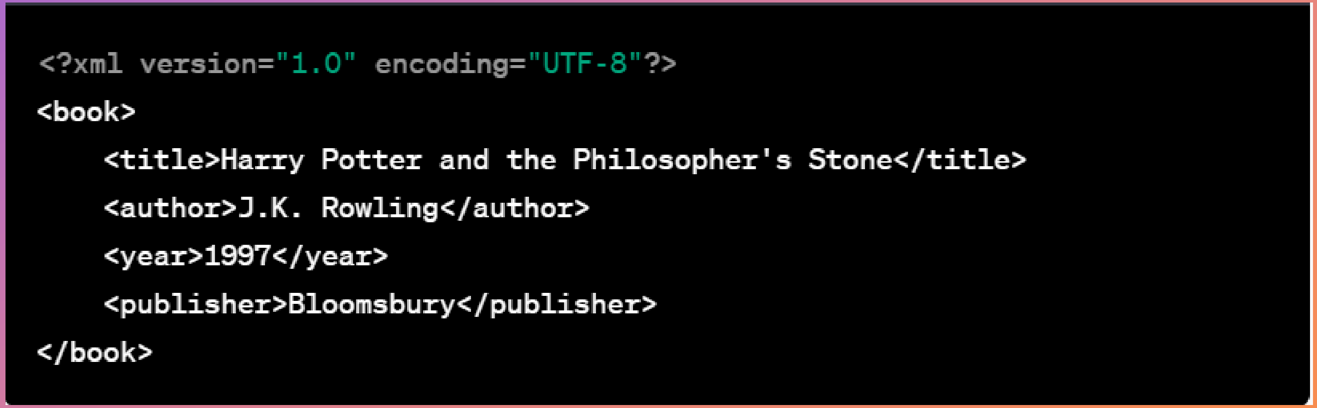
- **Beautiful Soup** is a Python library that is used for web scraping purposes to pull the data out of HTML and XML files. It creates a parse tree from page source code that can be used to extract data in a hierarchical and more readable manner.

Beautiful Soup

- **Beautiful Soup** is a Python library that is used for web scraping purposes to pull the data out of HTML and XML files. It creates a parse tree from page source code that can be used to extract data in a hierarchical and more readable manner.

DATA EXTRACTION USING FIN, FIND_ALL

- Python has a built-in module called xml for parsing and handling XML files.
- Import the xml.etree.ElementTree module: The ElementTree module is part of the xml module and provides a simple way to parse and manipulate XML files in Python.
- Load the XML file: Use the ElementTree.parse() method to load the XML file into a tree structure.
- Access the elements of the XML file: You can access the elements of the XML file using the Element object
- Modify the XML file: You can modify the XML file by manipulating the Element objects. For example, to add a new child element to the root element
- Save the modified XML file: Use the ElementTree.write() method to save the modified XML file.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<book>
    <title>Harry Potter and the Philosopher's Stone</title>
    <author>J.K. Rowling</author>
    <year>1997</year>
    <publisher>Bloomsbury</publisher>
</book>
```

Summary

With PowerPoint, you can create presentations and share your work with others, wherever they are. Type the text you want here to get started. You can also add images, art, and videos on this template. Save to OneDrive and access your presentations from your computer, tablet, or phone.

THANK YOU

Presenter name

Email address

Website