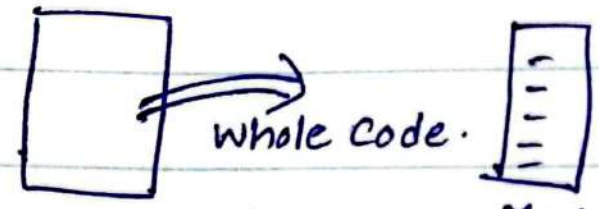
Date: 05/11/2024

Python For Data Science & AI

Difference between Compiler & Interpreter.

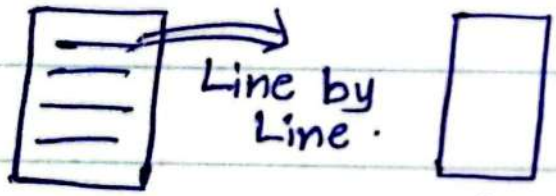
1) Compiler.



Human readable Code.

Machine readable code.

2) Interpreter.



Human readable Code.

Machine readable code.

① Create a variable. (Python is dynamically typed)

Ex :- age = 30

Rules for naming Python Variables

→ name must start with a letter ~~or~~ or the underscore charactor.

Ex :- var1 ✓ , _var1 ✓

→ name cannot start with a number.

Ex :- 2var1 → X

→ name can only contain alpha-numeric characters and underscores.

Ex :- (A-z, 0-9, and _) ✓

→ Avoid using python keywords & reserved words:

Ex :- if, else, while, for X

→ Name should be lowercase

(This is a ethic) (Will not appear any error but this is a coding standared)

Ex :- my_variable_name.

total_volume

number_of_items.

→ Variables should be descriptive.

→ If you have constants, use all uppercase letters with underscores separating words.

Ex :- PI_VALUE = 3.14, MAXIMUM_ATTEMPTS = 3

1) newvariable = 'John" → legal, not good

2) new-variable = "John" → legal

3) -new-variable = "John" → legal

4) newVariable = "John" → legal, not good.

5) NEWVARIABLE = "John" → legal, not good

6) newvariable2 = "John" → legal, not good

7) 2newvariable = "John" → illegal

8) new-variable = "John" → illegal

9) new variable = "John" → illegal

2) Pythen Data Types.

Built in data types:

1) Numeric Types - int, float, complex.
2) String data types - str
3) Sequence types - list, tuple, range.
4) Binary types - bytes, bytearray, memoryview
5) Mapping data type - dict
6) Boolean type - bool
7) Set data types - set, frozenset

int → holds signed integers.
unlimited

float → 15 decimal places.

Check type of a variable

Ex:- a = 200

type(a) ⇒ output
<class 'int'>

Scientific Notation.

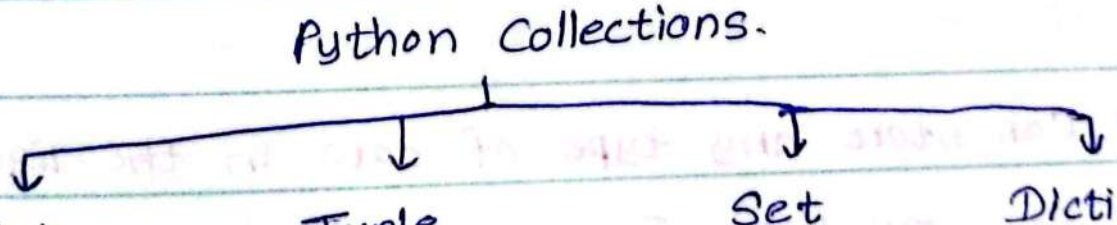
2.5×10^4

→ num_1 = 2.5e4

→ print(num_1, type(num_1))

output ⇒
25000.0
<class 'float'>

Python Collections (Arrays).



List	Tuple	Set	Dictionary.
→ Ordered	→ Ordered	→ unordered	
→ Changeable	→ Unchangeable	→ unchangeable	
→ Allow duplicates	→ Allow duplicates.		

1) Python Lists.

Ex:- my_list = [1, 2, 3, 4, 5]
↑ ↑ ↑ ↑ ↑
0 1 2 3 4 (Indexing)

- Print whole list, print (my_list , type(my_list))

  Output ⇒ [1, 2, 3, 4, 5] <class 'list'>

- Accessing element of the array.,

  print (my_list [2])

  Output ⇒ 3

- Assign a new value (replacing a new value) to an array.

  my_list [1] = 25

  print (my_list) Output ⇒ [1, 25, 3, 4, 5]

* This is because the python Lists ~~can~~ are changeable.

* Can store any type of data in the list

  Ex :- my_list = [1, 10, "cat", 8, True]

* Python lists are ordered.

  When add a new value it adds to the end of the list. That's why the lists are called ordered.

* Python lists allow duplicates.

Ex:- my-list = [1, 10, 10, "cat", 8, True].

print (my-list)

Output ⇒ [1, 10, 10, "cat", 8, True].

- To find the length of a list,

print (len (my-list))

Negative indexing

my-list = [1 , 10 , 10 , "cat" , 8 , True]

-6 -5 -4 -3 -2 -1

print (my-list[-2])

Output ⇒ 8

- Range of indexes.

① my-list = [1, 10, 10, "cat", 8, True]

- print (my-list[1:4])

Output ⇒ [10, 10, cat].

② my-list = [1, 10, 10, "cat", 8, True]

- print (my-list[:3]) Output ⇒ [1, 10, 10]

③ my-list = [1, 10, 10, "cat", 8, True].

- print (my-list[3:]) output ⇒ [cat, 8, True

- Insert a data to a list.

insert

```
my-list = [ 1, 10, 10, "cat", 8, True ].
my-list.insert ( 3, "rabbit" )
print ( my-list )
```

output ⇒ [1, 10, 10, 'rabbit', 'cat', 8, True]

* With 'insert' command we can set data to any position of an a list.

- Append a data to a list.

append

```
my-list = [ 1, 10, 10, "cat", 8, True ].
my-list.append ( "car" )
print ( my-list ).
```

output ⇒ [1, 10, 10, 'cat', 8, True, 'car'].

* With 'append' command, we only can add a data after the last item of the list.

- Add a list to back of the list.

extend

```
my-list = [ 1, 10, 10, "car" ].
new-list = [ "bus", "train" ].
my-list.extend ( new-list )
print ( my-list ).
```

output ⇒ [1, 10, 10, 'car', 'bus', 'train']

- To remove an item from the list.

my_list = [1, 10, 10, "car"].

I want to remove this element.

my_list.pop(1)

print(my_list)

Output ⇒ [1, 10, 'car'].

* We can not delete whole list by 'pop' command

- To remove an item you can use del command also.

my_list = [1, 10, 10, "car"].

Want to delete this.

~~my_list.del(1)~~ my_list.__delitem__(1)

print(my_list) Output ⇒ [1, 10, 'car'].

* We can delete whole list using this 'del' command.

del my_list

~~print(len)~~ print(len(my_list)) ⇒ Error (Because there's no list)

- If we want to clear the items of the list only but keep the list, we can use 'clear' comma

my_list.clear()

print(len(my_list)) ⇒ output ⇒ 0

~~my-list = ["dog~~

- Sort the array in alphabatical order.

① Case Sensitive Sorting.

Capital letters priority > Simple letters priority.

Sort

Ex :- my-list = ["dog", "apple", "Parrot", "banana"]

my-list.sort()

print(my-list)

Output ⇒ ['Parrot', 'apple', 'banana', 'dog']

* By default 'sort' is case sensitive.

② Make the sort insensitive.

Ex :- my-list.sort(key = str.lower)

print(my-list)

Output ⇒ ['apple', 'banana', 'dog', 'Parrot']

* By this way you can make the 'sort' case insensitive. ⇒ sort(key = str.lower)

③ Make the sort decending order.

Ex :- my-list.sort(reverse = True)

print(my-list)

Output ⇒ ['dog', 'banana', 'apple', 'Parrot'].

12 11 2024

(4) How to reverse a list as it is.

Ex :- my_list = ["apple", "dog", "cat"]

my-list.reverse()

print (my-list)

Output ⇒ ['cat', 'dog', 'apple']

The ways to copy a list to another.

- How to copy a array list to an another list.

Ex :- my-list-1 = [20, 15, 10, 5]

my-list-2 = my-list-1.copy()

using copy()

After copying my-list-1 to my-list-2, if we change a value of the my-list-1, it doesn't affect to my-list-2.

- Copy a list to another using list constructor.

Ex :- my-list-1 = [20, 15, 10, 5]

using list constructor list()

my-list-2 = list (my-list-1)

"list" constructor

my-list-1[1] = 40

print (my-list-1, my-list-2)

Output ⇒ [20, 40, 10, 5] [20, 15, 10, 5]

- Joining two list easily.

```
my-list-1 = [1, 2, 3, 4]
my-list-2 = [5, 6, 7, 8]
my-list-3 = my-list-1 + my-list-2
print(my-list-3, len(my-list-3))
```

Output ⇒ [1, 2, 3, 4, 5, 6, 7, 8] 8

2) Python Tuples.

Ex :- my-tuple = (12, "Dog", 10)

- Get tuple length.

len()

```
Ex :- my-tuple = (1, 2, 3)
      print(len(my-tuple))
```

Output ⇒ 3

* If a python tuple has only one element, we should add a comma (,) after the element.

Ex :- my-tuple-1 = ("Dog",)

Otherwise it identifies the type of the tuple "String".

- Negative indexing → Same as lists.

- Range of tuples → Same as lists.

  Ex :- my-tuple [2:5], my-tuple [:4]

  my-tuple [2:]

* Python tuples can not change the value.

Ex :- my-tuple-1 = ("Dog", "Cat", 10)

my-tuple-1 [1] = "Rat"

print (my-tuple-1 [1]).

Error ⇒ TypeError : 'tuple' object does not support item assignment.

How to change a value of a tuple.

Ex :-

my-tuple-1 = ("Dog", "Cat", 10)

my-~~tuple~~list-1 = list (my-tuple-1)

my-list-1 [1]

~~my-tuple-2 [1]~~ = "Rat"

my-tuple-1 = tuple (~~my-tuple-2~~ my-list-1)

print (my-tuple-1, type (my-tuple-1))

Output ⇒ ("Dog", "Rat", 10) <class 'tuple'>.

- Nested Tuples.

Ex:- my-tuple-1 my-list-1 = ((1, 10, 8), ("Dog", "Cat", "Rat"), (True, False))

print (my-tuple-1[0][1], my-tuple-1[1][2], my-tuple-1[2][0])

Output ⇒ 10 Rat True.

- Python Range. Ex:- 2 5 8 11 ... (d = 3)

arithmetic progression - ಅಂಕಗಣಿತ ಶ್ರೇಢಿ.

→ In python, range() is a built-in function used to create an immutable sequence of numbers representing an arithmetic progression.

→ The range() function can be used to generate a range of integers based on a start value, stop value, and step value.

→ It is commonly used in loops and ~~the~~ other scenarios where a sequence of numbers in needed.

range (start, stop, step)

Ex:- s-1 = range (2,15,3)

print (s-1, type (s-1))

Output ⇒ range (2,15,3) <class 'range'>

3) Python Dictionaries { }

Ex:- dict-1 = {

"name" : "Sugar",

"price" : 250.50,

"weight" : "1 kg"

}

print (dict-1, type (dict-1))

Output ⇒ {'name': 'Sugar', 'price': 250.5, 'weight': '1kg'}

<class 'dict'>

* Python dictionaries do not allow duplicate values.

(If there are duplicate values, it replace the first value from the last value:)

Ex :- dict-1 = {

"name" : "Sugar",

"price" : 250.50,

"price" : 120.25,

"weight" : "1 kg"

}

print (dict-1) ⇒ {'name': 'Sugar', 'price': 120.25, 'weight': '1kg'}

len()

- See length of a dictionary.

  print (len (dict-1)) Output ⇒ 3

- How to access a value ~~of~~ in a dictionary.

  print (dict-1 [" weight"]) Output ⇒ 1 kg

- Dictionary items - Data types.

  Strings, Numbers, Booleans, Lists, Tuples, Sets, Other dictionaries, Custom objects

  Ex :- my_dict {

  " name" : " John",

  " grades" : [85, 92, 78],

  " coordinates" : (12.34, 56.78),

  " preferences" : {"color": "blue", "food": "pizza"}

  }

- How to update a value in a dictionary.

  ~~x = my-dict [pr~~

  Ex :- my-dict ["name"] = " peter"

.update({})

- Update method (To update multiple values of a dictionary)

Ex:- dict-1 . update ({"weight": "2 kg", "price": 220.50})

* In the update method, we ~~should~~ must pass a dictionary.

- How to add new value to a dictionary.

Ex:- dict-1 ["color"] = "white"

.pop()

- To remove an item from a dictionary.

Ex :- dict-1 . pop ("color")

.popitem()

- popitem() - To remove last item from a dictionary.

Ex:- dict-1 . popitem ()

del

- We can use del() method to delete an item too.

Ex :- del dict-1 ["weight"]

del dict-1 → This is to destroy the dictionary.

.clear()

- Clear() - This will clear all the items in the dictionary but not the dictionary.

Ex :- dict-1 . clear()

Method 01

.copy() • Copy one dictionary to another dictionary.

```
dict_2 = dict_1.copy()
```

Method 02

dict() • We can copy one dictionary to another by using dictionary constructor.

```
dict_2 = dict(dict_1)
```

dictionary constructor.

.keys() • To abstract all the keys of a dictionary.

Ex :- `dict_1.keys()`

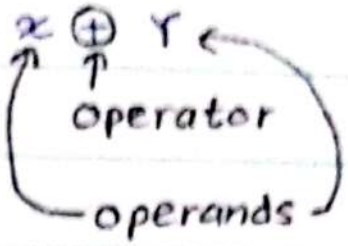
input() Function

(This function use to get user inputs)

Ex :-
```
x = input("Enter a value:")
print(x)
```

Operators of Python.

* Operators vs Operands



Python Operators :-

1) Arithmetic Operators
2) Assignment Operators
3) Comparison Operators
4) Logical Operators
5) Identity Operators
6) Membership Operators
7) Bitwise Operators.

1) Arithmetic Operators.

- + → Addition.
- − → Subtraction.
- * → Multiplication
- / → Division.
- % → Modulus
- ** → Exponentiation.
- // → Floor division (7//2 = 3)

2) Assignment Operators.

• =	(x = 5)	
• +=	(x += 3)	(x = x + 3)
• -=	(x -= 3)	(x = x - 3)
• *=	(x *= 3)	(x = x * 3)
• /=	(x /= 3)	(x = x / 3)
• %=	(x %= 3)	(x = x % 3)
• **=	(x **= 3)	(x = x ** 3)
• //=	(x //= 3)	(x = x // 3)

3) Comparison Operators.

• ==	Equal to	(x == 2)
• !=	Not equal to	(x != 2)
• >	Greater than.	(x > 2)
• <	Less than	(x < 2)
• >=	Greater than or equal to	(x >= 2)
• <=	Less than or equal to.	(x <= 2)

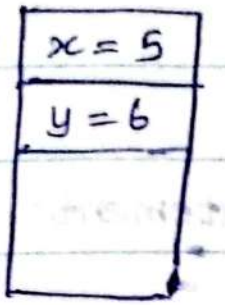
4) Logical Operators

- and → Returns True if both segments are true. (x<5 and x<10)
- or → Returns True if one of the statements is true (x<5 or x<4)
- not → Reverse the result, returns False if the result is true. not (x<5 and x<10)

5) Identity Operators.

- is → Returns True if both variables (x is y) are the same object
- is not → Returns True if both variables (x is not y) are not the same object

* These operators are used to compare the memory locations (identities) of two objects.



x = 5
y = 6

print (x is y)
Output ⇒ False.

print (x is not y)
Output ⇒ True.

6) Membership Operators.

* Used to test whether a value or variable is presented in an object (list, tuple, ...).

Ex :- nums = [10, 12, 8, 5, 4]
print (10 in nums).

- in (x in y)
- not in (x not in y)

7) Bitwise Operators

- & AND (x & y)
- | OR (x | y)
- ^ XOR (x ^ y)
- ~ NOT (~x)
- << Zero fill left shift (x << 2)
- >> Signed right shift (x >> 2)

Control Flow Statements.

1) Conditional Statement

- If condition

Ex:-
```
n = 12
if n % 2 == 0 :
    print ("This is an even number")
else :
    print ("This is an odd number")
```

~~Ex:-~~ ~~a = 10~~
~~if a<0:~~
~~print("~~

Ex :- a = 10
if a<0:
    print("This is a negative number")
elif a>0:
    print("This is a positive number")
else:
    print("This is zero")

19/11/2024

Ternary Operator

Ex :- x = 10
result = "Even" if x % 2 == 0 else "Odd"
print(result)

~~Switch~~

Switch - Case Statement

Ex:-
```
response-code = 203
match response-code:
    case 200:
        print ("OK")
    case 201:
        print ("created")
    case 404:
        print ("404 Not found")
    case 500:
        print ("Internal Server Error")
    case _:
        print ("Something else")
```

Ex:-
```
x = int (input ("Enter a number"))
match x:
    case x if x>1:
        print ("Positive number greater than 1")
    case 1:
        print ("Positive One")
    case 0:
        print ("Zero")
    case -1:
        print ("Negative one")
    case x if x<-1:
        print ("Negative number less than negative one")
    case _:
```

- Matching by the length of an Iterable (Pattern Matching)

Ex:
```
numbers = [4, 3, 7]

match numbers:
    case [x, y]:
        print(x*y)
    case [x, y, z]:
        print(x+y+z)
    case _:
        print("The list does not contain 2 or 3 numbers!")
```

2) Loops.

1) For Loops

Ex:
```
for item in sequence:
    # Code block to execute for each item.
```
(variable)

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

* (we can do this only in lists)

list Comprehension :

[expression for item in iterable if condition]

- expression: value that gets added to the list
- item: Current element from the iterable
- if condition: this is optional (filters elements)

Ex :- my-list = [2, 3, 4]

print([x**2 for x in my-list])

Ex :- Print 1 to 10 squire root values in a list.

print([x**2 for x in range(1, 11, 1)])

Ex :- Output ⇒ [4, 16, 36, 64, 100]

1) print([x**2 for x in range(2, 11, 2)])

2) print([x**2 for x in range(1, 11) if x%2==0])

2) While loops.

Ex:- while condition:
    } body.

Ex :-
```
m = 5
i = 0
while i<m:
    print(i, end=" ")
    i = i+1
print("End")
```

line එක print කරල ඉවර වෙලා අන්තිමට print වෙන්න ඕනෙ නේ.

- Exiting from a loop using 'break'

Ex :- Output ⇒ 2 4 6 8 End.

```
num = int(input("Enter a number: "))
i = 0
while i < num:
    if i == 10:
        break
    else:
        print(i)
        i += 2
print("End")
```

- Exiting from a loop using 'continue'

Ex :- Output ⇒ 3 15 9 7

```
my_list = [3, 8, 15, 10, 9, 7, 14].
for x in my_list:
    if x%1 == 0:
        continue
    else:
        print(x).
```

Functions & Functional Programming.

* Unpacking.

iterable values unpack කරලා වචන වෙනම variable වලට assign කරනවා.

(iterable values කියෙන්නේ list, tuple, dictionary. වගේ)

Ex:- my-list = [10, 12, 8]

x, y, z = my-list

print(x, y, z) ⇒ Output ⇒ 10 12 8.

* tuple also can be done like this. (As lists)

Dictionary Unpacking.

Ex:- person = {

"name" : "John",

"age" : 25

}

{name, age} = person.

* Variable names කලින් වෙන්න ඕන dictionary එකේ key words වලමම.

print(name, age)

Output ⇒ John, 25

Comes a syntax error)

Dictionary Unpacking.

```
person = {
    "name": "John",
    "age": 25,
    "gender": "Male",
}
```

Unpacking :⇒

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

Output ⇒

```
name   John.
age    25
gender Male.
```

- keys විතරක් print කරන්න.

```
for key in person.keys():
    print(key)
```

- values විතරක් print කරන්න.

```
for value in person.values():
    print(value)
```

* Merge Dictionaries.

Ex :-
```
dict-1 = { 'a' : 1 }
dict-2 = { 'b' : 2 }
dict-3 = dict-1 | dict-2
print ( dict-3)
```
Output ⇒ { 'a' : 1 , 'b' : 2 }

* Dictionary එකක key එකකට අදාළව value එක print කිරීම

Ex :-
```
my-dict = {
    'a' : 1,
    'b' : 3,
    'c' : 8
}
```

I ක්‍රමය.

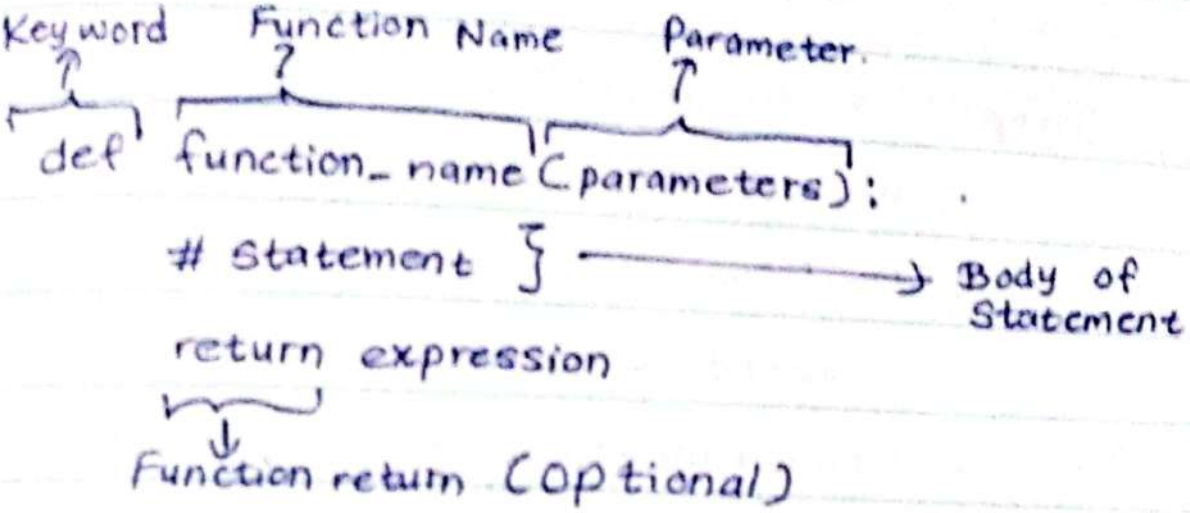
```
print( my-dict['b'])
```
⇒ output ⇒ 3

II ක්‍රමය

```
my-dict.get( key , default-value)
```
Optional (මෙතනට key එක dictionary එකේ නැත්නම් return වෙන්න ඕන value එක)

```
print ( my-dict . get ('d', "Not Found"))
```

What is a function



Ex:- * my_list = [10, 2, 5, 8, 9, 11, 13]

print (my_list [1 : 5 : 2])

start value — end value — step value (ඉදලා ඉඳන් ගමන්.)

Output ⇒ [2, 8]

Parameters or Arguments.

→ Parameter - is the variable listed inside the parentheses in the function definition.

→ Argument - is the value that are sent to the function when it is called.

Argumants of Python

Types of Arguments :

1) Default arguments.
2) Positional arguments.
3) Keyword arguments.
4) Aribitrary Positional arguments.

1) Default arguments.

Ex:-
```
def add_numbers ( a, b=5):
        return a+b

add_numbers ( 6, 8 ) => 14
add_numbers ( 6 ) => 11
```
• Because b is a default argument.

* This will generate an Error

Ex:-
```
def add_numbers ( a, b=5, c)    X
        return a+b+c
```
If not, we have to assign a default value to c also.

* All the arguments to its right must also have default values

(default values තියෙන්න ඕන parentheses එකේ අන්තිමට).

මෙම ලිව්වොත් හරි :-

Ex:-
```
def add_numbers ( a, b, c=5)
        return a+b+c
```

2) Positional arguments. (Normal arguments).

The values are passed to the function based on their position or order. In the function's parameter list.

3) Keyword arguments.

* Does not need to remember the order of parameters.

Ex:- def add_number (a, b):
        return a+b

add_number (a = 5, b = 8)
add_number (b = 8, a = 5)

Here, Order doesn't matter.

Q1) Define a function name calculate_total_cost with the following parameters item_price:

* mandatory, price of the item

quantity : mandatory.

discount : optional ← default = 0 (A percentage applied to the total price)

tax : optional ← default = 0 (A percentage applied for the discounted price).

(Interview Questions)

4) Aribitrary Positional Arguments.

This type of arguments are ~~called~~ used when the programmer does not know the number of arguments to be passed into the function.

Ex:- def aribitrary-positional-arguments(*args):
    print(args, type(args))

aribitrary-positional-arguments(1, 2, 3)
↳ Output ⇒ (1, 2, 3) <class 'tuple'>

Ex:- Write a python function called Summarize_grades that accept a student name ~~and a~~ as mandetory argument and an aribitrary num of grade scores the function should,

1) print the student names.
2) calculate and print the highest grade, lowest and avg from the provided scores.
3) If no grades are provided you print no grades available.

```
def summarize_grades(stu_name, *grades):
    print("Student Name:", stu_name)
    if not grades:
        print("Grades not available")
    else:
        print("Maximum score is:", max(grades))
        print("Lowest score is:", min(grades))
        sum = 0
        for grade in grades:
            sum += grade
        print("Average:", sum/len(grades))
summarize_grades("Kamal", 50, 60, 70)
```

↳ Output ⇒

```
Student Name: Kamal
Maximum Score is: 70
Lowest Score is: 50
Average: 60.0
```

5) Arbitrary Keyword Arguments. (** kwargs)

Ex:- def aribitrary_keyword_args(** kwargs):
    print(kwargs, type(kwargs))

aribitrary_keyword_args(name="John", age=30, city="New York")

↳ Output ⇒ {'name': 'John', 'age': 30, 'city': 'New York'} <class 'dict'>

* It is a dictionary.

* This way we can print keys and values.

def aribitrary_keyword_args(** kwargs):
    ~~print(kwargs, type(k~~
    for key, value in kwargs.items():
        print("key": , key, 'value:', value)

aribitrary_keyword_args(name='John', age=30, city="New York")

↳ Output ⇒

key: name value: John
key: age value: 30
key:

Q1) Employee Management System

* Write a python function called employee_info that accept a required name parameter and an arbitrary number of keyword arguments representing additional details about the employee.

the function should,

i) Print the employee's name iterate through the keyword arguments and print each key value pair in the format "<key> : <value>"

ii) Return a dictionary with all the employee details.

```
def employee_info (name, **kwargs):
    print ("Employee Name: ", name)
    for key, value in kwargs.items():
        print (key, ":", value)
    return {"name": name} | kwargs

employee_info = employee_info (name = "John", age=30,
                               city = "Panadura")

print (employee_info).
```

This is dictionary merging using '|'.
we can do it like this too,

```
return {"name": name, **details}
```

Python Built in Functions

1) abs():

Returns the absolute value

Ex:- abs(-20) → Output → 20

2) map():

Use to apply a given function to every item in an iterable (such as a list, tuple, or other sequence) and return an iterator that contains the results.

- It's a convenient way to perform an operation on each element of a collection without the need for explicit loops.

Ex:- map (your_function, iterable)

map (find_squire , my_list)

↳ The output < class 'map')

* So we should convert it to a list, before do other things.

Ex: my_list = [5, 4, 2]

```
def find_squire(x):
    return x*x
result = map(find_squire, my_list)
print(result, type(result))
result = list(result)
print(result, type(result))
```

Output ⇒ <map object 0x000...> <class 'map'>

[25, 16, 4] <class 'list'>

Q1) Write a function to calculate items of two lists.

```
my_list_1 = [10, 12, 7, 5]
my_list_2 = [9, 5, 4, 1]
def sum_two_numbers(num1, num2):
    return num1 + num2.
result = map(sum_two_numbers, my_list_1, my_list_2)
list = list(result)
print(list)
```

Output ⇒ [19, 17, 11, 6]

Q2) You have a list of integers representing temperatures in celcious.

temperatures = [20, 30, 25, 40, 50]

write a python program using the map function to convert these temperatures into faranhide.

Use the formula Fahrenheit = Celsius $\times \frac{9}{5} + 32$.

```
tempuratures = [20, 30, 25, 40, 50]
def convert_to_farenheit(x):
    return (x*9)/5 + 32
result = map(convert_to_farenheit, temperatures)
temp_list = list(result)
print(temp_list)
```

3) filter():

Filter elements from an iterable (such as a list, tuple, or other sequence) based on a specified condition. It creates a new iterable containing only the elements that satisfy the given condition.

class type. ⟹ <class 'filter'>

* filter (function , iterable)

මේ function එකෙන් return වෙන්න ඕනේ logical value එකක් විතරයි.
Ex :- True / False

මේ iterable එකෙන් output එක විදියට දෙන්නේ, True return කරන items වල values විතරයි.

* මේකෙන් කරන්නේ values, filter කරන එක.

Q) Write a python code to filter out the odd numbers and print even numbers.

```
list_item = [1, 2, 3, 4, 5]
def filter_odd_numbers(x):
    return x % 2 == 0
result = filter( filter_odd_numbers, list_item)
list = list(result)
print (list)
```

Output → [2, 4]

Python Lambda Functions
(Anonymous functions)

Syntax :

```
lambda arguments : expression
```

keyword

Ex:- `lambda x, y : x+y`

මේක reffer කරන්නේ variable එකක් විදිහට.

Ex:-
```
add = lambda x, y : x+y
print( add( 3, 4))
```
⟹ output → 7

How to call without assigning to a variable.

```
result = (lambda x, y : x+y)(5, 3)
```

```
print((lambda x, y : x+y)(5, 3))
```

Write a python program that :

i) Takes a list of tuples, where each tuple contains a name (string) and age (integer).

ii) Use a lambda function to filter out the tuples where the age is ~~greater~~ less than 18

iii) Print the list of tuples after filtering

```
people = [("alice", 22), ("John", 17), ("Andrew", 21),
                                        ("kate", 15)]
result = filter((lambda x: x[1] > 18), people)
list1 = list(result)
print(list1)
```

* Why use lambda functions :-

Ex:-
```
def myfunc(n):
    return lambda a: a * n
mydoubler = myfunc(2)
print(mydoubler)
```
⇒ (Output → 4)

Q) Write a lambda function to decide the maximum value between two numbers.

```
print("Max Value is", (lambda x,y : x if x>y
                                else y)(5,6))
```

Output ⇒ Max Value is 6 //

Python Recursive Functions.

Q) Write a function to get factorial value of a number.

```
def find-factorial(n):
    fact-val = 1
    while (n>0):
        fact-val *= n
        n = n-1
    return fact-val
print( find-factorial(5))
```

Output ⇒ 120.

```
def recursor():
    recursor()
recursor()
```

With using a recursive function:

```
def factorial(x):
    if x==0 or x==1:
        return 1
    else:
        return (x * factorial(x-1))
print(factorial(1000))
```

* By default, the maximum depth of recursion is 1000. If the limit is crossed, it results recursion error.

Advantages of Recursive functions.

- Simplicity and Readability
- Divide and Conquer
- Reduced Code Length

Disadvantages of Recursive Functions.

- Can lead to a stack over flow error.
- Performance Issues
- Higher memory usage
- Complex Debugging.

URL ⇒ Python module index.

'Modules' in Python.

Modules are a fundamental concept in Python that allows you to organize your code into reusable and maintainable components.

- How to import a in built module:

  1) import math.

     print(math.pi)

  2) import math as m.

     print (m.pi)

- Without importing whole module, we can import only method.

  * from my_module import add

    print(add (5,10))

    → if whole module imports like this
    import my-module,
    then we have to access the add method like this,
    print(my-module. add (5,10))

- If we want to import all the methods of a module we can use this,

```
from my-module import add
from my-module import substract
```

my-module
→ add()
→ substract()

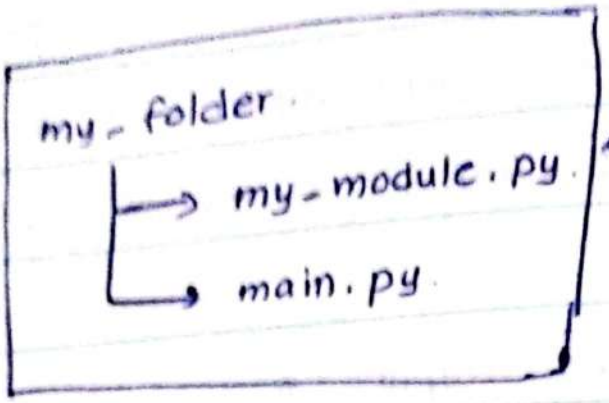
```
from my-module import *
```

if __name__ = "__main__" CONSTRUCT.
(name = main constructor).

- මෙක හැදිච්චි කරන්නේ තනි තනි files test කරගන්න.
- files import කරන්න කලින් ඒ file එකේ තියෙන functions හරියට වැඩ කරනවද කියල බලන්න හොඳ කරයි.

Ex:-
```
if__name__==="__main__":
    print("This code runs when the code is
          executed directly.")
```

Python Package



* Why __init__.py file?

- If we do not put this file into the ~~package~~ directory, it difines as a normal directory, not as a package.

But we implement this file in the directory, python will be difined ~~as~~ the directory as a package

* මේ file එක තමයි package එකක python interpreter වලින්ම interprit කරන්නේ.

my_folder

my_calculator

__init__.py

addition.py

subtraction.py

main.py

my_calculator is a package.

(1st way)

addition.py

```
def add(a, b):
    return a+b
```

subtraction.py

```
def substract(a,b):
    return a-b
```

main.py

```
# main.py.
# Import the entire modules.
import my-Calculator.addition
import my-calculator.subtraction.

# Perform some calculations.
result1 = my-calculator.addition.add(5, 3)
result2 = my-calculator.subtraction.substract(10, 4)

# Display the result
print("Addition result:", result1)
print("Subtraction result:", result2).
```

Output ⇒ Addition result : 8
Subtraction result : 6

(2nd way)

main.py:

```
# main.py
# Import the entire modules
from my-calculator import addition.
from my-calculator import subtraction.

# Perform some calculations.
result1 = addition.add(5,3)
result2 = subtraction.subtract(10,4)

# Display the result
print("Addition result:", result1)
print("Subtraction result:", result2)
```

(3rd way) * This way is the best way.

main.py:

```
# main.py.
# Import the
from my-calculator.addition import add
from my-calculator.subtraction import subtract
# Perform some calculations.
result1 = add(5,3)
result2 = subtract(10,4)
```

(4th way) * This is the best way.

In the __init__.py

```
from .addition import add
from .subtraction import substract.
```

↑ මෙතන . එක අනිවාර්යෙන්ම දාන්න ඕනෙ.

main.py

```
# main.py
# Import the entire modules
from my-calculator import add, substract

# Perform some calculations.
result1 = add(5,3)
result2 = substract(10,4).

# Display the result
print("Addition result:", result1)
print("Subtraction result:", result2)
```

Python classes.

* Class Definition.

ex :- class NewCar :

we use Camal Case only when implement a class name.

Capital

class Car :

category = " motor vehicle" ← Class attributes

* How to create a Object (Instance)

ex :- new_car = Car ( )

* How to access an object's attributes.

ex :- print (new_car. category)

* 'self' keyword.

class Car :

category = " motor vehicle"

def display (self):

print (" This is a display function)

• 'self' එන්නේ ඒ method (function) එක අයිති වෙදුම class එකටම කියලා කියන්න.

* __init__ - This is the constructor of a class

Ex:- def __init__(self, name)

↑ Constructor

• Constructor use කරන්නේ instance attributes assign කරන්න.

Ex:- def __init__(self, model, color):

self.model = model

self.color = color

Instance Attributes.

Ex:- class Car:

category = "motor vehicle"

def display(self):

print(self.model)

print(self.color)

def __init__(self, model, color):

self.model = model

self.color = color

* Class attributes can access directly. No need to create an object. But to access instance attributes, need to create an object.

Q) Create a class name BankAccount with the following • An ~~and~~ attribute account_holder to store the name of the account-holder.

~~And attr~~ • An attribute balance to store the account balance, initialized to 0.

Add following methods to the class.

- deposit(amount) → Adds the given amount to the balance.
- withdraw(amount) → Substract given amount to the balance if sufficient fund exist otherwise print an error msg.
- display balance → Print the current balance.

Write a small script to demonstrate the following.

- Create an object of the bank account class.
- Perform a few deposit and withdrawal operations.
- Display the balance after each operation.

```
class BankAccount:
    def __init__(self, name):
        self.account_holder = name
        self.balance = 0
    def deposit(self, amount):
        if amount > 0:
            self.balance += amout
        else:
            print("Amount should be positive.")
    def withdraw(self, amount):
        if amount < 0:
            print("Amount should be positive")
        elif amount > self.balance:
            print("Insufficient fund")
        else:
            self.balance -= amount
    def display(self):
        print("The current balance is," self.balance)
```

Testing:

```
account_1 = BankAccount("John")
account_1.deposit(500)
account_1.display()
account_1.withdraw(100)
account_1.display()
```

Inheritance in Python.

Ex:-

```
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self)
        pass


class Dog(Animal)
    def speak(self):
        print(self.name, "says woof")
class Cat(Animal)
    def speak(self)
        print(self.name, "says meow")

dog = Dog()
```

* මේක create කළොත්, Dog කියන class එකේ constructor එකක් නැති නිසා super class එකේ constructor එක call වෙනවා. එකේ name එකට value එකක් pass කරන්න ඕනෙ නිසා error එකක් print වෙනවා.

dog = Dog("Roxy") → OK.

* If we create a constructor manually in the sub class, it doesn't call the constructor in the super class.

We can create a constructor in Dog class like this:

```
class Dog(Animal):
    def __init__(self, name, color):
        super().__init__(name)
        self.color = color
```

Q) Library management system.

You are tasked with creating a library management system using inheritance. Implement following classes.

1) Base class → LibraryItem

Attributes :

- title (string)
- author (string)
- publication_year (int)

Methods :

- display_info() → print the details of the item.

2) Derived class → Book

it inherits from Library Item:

additional Attributs:

genre (string)

isbn (string)

Override the display_info()

print the details including genre & isbn.

3) Derived class - Magazine.

Inherits from Library Item

additional Attributes:

issue (string)

The issue no. or date of the magazine.

Override display_info()

print the details including the issue.

Task:

1) Create instances of each class (Book, Magazine) with appropriate values for their attributes.

2) Call the display_info method for each instant to test inheritence & method overriding.

Answer:

```
class LibraryItem:
    def __init__(self, title, author, publication_year):
        self.title = title
        self.author = author
        self.publication_year = publication_year
    def display_info(self):
        print("Details:", self.title, self.author,
                                    self.publication_year)

class Book(LibraryItem):
    def __init__(self, title, author, publication_year, genre, isbn):
        super().__init__(title, author, publication_year)
        self.genre = genre
        self.isbn = isbn
    def display_info(self):
        super().display_info()
        print(self.genre, self.isbn)

class Magazine(LibraryItem):
    def __init__(self, title, author, publication_year,
                                    issue):
        super().__init__(title, author, publication_year
        self.issue = issue
    def display_info(self)
        super().display_info()
        print(self.issue)

library_item = LibraryItem("Abc", "Abc", 1995)
book = Book("Abc", "Abc", 1995, "Abc", "Abc")
magazine = Magazine("Abc", "Abc", 1995, "Abc")
library_item.display_info()
book.display_info()
magazine.display_info()
```

Polymorphism in Python.

It allows objects of different classes to be treated as objects of a common superclass, and it enables you to write more flexible and reusable code.

Ex :-

```
class Shape:
    def area(self):
        pass

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self)
        return self.width * self.height


class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius


shapes = [Rectangle(2, 3), Circle(5)]

for shape in shapes:

    print(shape.area())
```

Encapsulation in Python.

* By default all attributes & behaviours are public.

access level → Attributes
access level → Behaviours

access level ⇓ Restrict

* Python වල syntax භාවිතා කරන්නේ. (public, private වගේ keywords නොවේ.)

• Protected

Ex:-
```
def __init__(self, data):
    self._data = data        ← protected

def _speak(self):            ← protected
    print("Hello")
```

• Private

Ex:-
```
def __init__(self, data):
    self.__data = data       ← private

def __speak(self):           ← private
    Print("Hello")
```

Ex:-
```
class MyClass:
    def __init__(self, protected_var, private_var):
        self._protected_var = protected_var
        self.__private_var = private_var
my_obj = MyClass("test1", "test2")
print(my_obj._protected_var)  ⇒ Not Encouraged
print(my_obj.__private_var)  ✗
```

Output :- test1

error

* Protected attributes and methods can be accessed from the outside. but it is not encouraged.
* Private attributes and methods can not be accessed from the outside.

Example :

Getters and setters.

```
class Example:
    def __init__(self):
        self.__data = "Encapculated data"
    def get_value(self):
        print("Value of data", self.__data)
    def set_value(self, new_value)
        self.__data = new_value.
my_example = Example()
my_example.get_value()
my_example.set_value("Hello World")
```

* Protected methods and attributes to be used within the class itself and its subclasses.

Implement Quadatric Equation class.

Write a python program to create a class name, QuadatricEquation that represent a quadatric equation of the form ax^2+bx+c.

The class should have attributes:

- Private attributes --a, --b and --c to store the coreofitient of the quadatric equation.
- Methods.

→ A constructor to initialize the coefitient a, b and c.

→ A private method -- discriminant() that calculates and return the discriminant ($D = b^2-4ac$)

→ A public method find_roots() that use the private discriminant()

Return the root of the quadatric equation
- If $D=0$, one real root
- if $D>0$, two distinct real roots.
- if $D<0$, two complex roots.

```
class QuadraticEquation:
    def __init__(self, a, b, c):
        self.__a = a
        self.__b = b
        self.__c = c

    def __discriminant(self)
        return self.__b**2 - (4 * self.__a *
                              self.__c)

    def find_roots(self)
        D = self.__discriminant()
        if D > 0:
            root_1 = (-self.__b + D**0.5) /
                     (2 * self.__a)
            root_2 = (-self.__b - D**0.5) /
                     (2 * self.__a)
            print("Two roots are", root_1, root_2)
        elif D == 0:
            root_1 = -self.__b / (2 * self.__a)
            print("This has only one root
                  and it is", root_1)
        else:
            print("This has complex roots")

equation_1 = QuadraticEquation(1, -5, 6)
equation_1.find_roots()
```

File Handling.

File Operation

1) Open a file
2) Read or write (perform operation)
3) Close the file.

1) Open file in current directory

Ex :- file_1 = open("myfile.txt")

file path

මේකෙන් create වෙන්නෙ object එකක්.

How to read the content and print.

```
Ex :- file_1 = open("my_file.txt")
      file_content = file_1.read()
      print(file_content).
```

This gives all the content in the my_file.txt file.

file path — file mode.

* file_1 = open("my_file.txt", "r")

By default මේකෙ file mode එක read mode.

Different modes to open a file.

1) r - Open a file for reading. (default) (නැති file name එකක් දැම්මොත් error දෙනවා)

2) w - Open a file for writing.

file එකක් නැති වුණොත් ඒ නමින් අලුත් file එකක් create කරනවා. දැනටම ඒ file එක තියෙනවා නම්, කලින් content එක clear කරලා දැන් content එක write කරනවා.

3) x - Open a file for exclusive creation.

file name එකෙන් අලුත් file එකක් create කරනවා. ඒ file එක තියෙනවා නම් error දෙනවා.

4) a - තියන file එකේ content එක යටට අලුත් file එකේ content එක append වෙනවා.

readline() method.

text file එකේ එක line එකක් විතරක් read කරන්න. (\n (next line character) හමුවෙන තැනට විතරයි මේ method එකෙන් read කරන්නේ.

If the file is in text mode, the readline() returns the line as a string.

If you're working with binary mode, the lines will be returned as bytes objects.

Ex:-
```
file_1 = open("my_file.txt")
file_content = file_1.readlines()
print(file_content, type(file_content))
file_1.close()
```

Output ⇒ ['I love programming.\n', 'I like Data Science.'] <class 'list'>

* readlines() method එකෙන් return වෙන්නෙ list එකක්

With statement for file handling in Python.

Ex:-
```
with open("my_file.txt","r") as file:
    content = file.read()
    print(content).
```

* No need for file.close() as it is automatically handled.

Resource Management : Using with ensures that files are properly closed after operations are complete, preventing issues like file locks, resource leaks, or data corruption.

if we use writelines(), we have to pass a list as parameters of the method.

Ex:- file.writelines(['Hello word\n', 'This is our python class'])

Q) Implement a simple contact management system

create a program that stores and manage contact in a file name contact.txt

Each contact entry should include name, phone No and email.

Features to implement:

Add a new contact (Append new contact)

View all contacts (Read & display all the contacts)

Exit the program

JSON - JavaScript Object Notation.

* Language independant notation එකක්.
* Platform independant

JSON data is represented as a collection of key-value pairs.

* JSON file එක keys ඔක්කොම්මත් string විය යුතුයි. (Dictionary එක keys වලට number format එකක් වෙනස් වෙන්න පුලුවන්. Dictionary එක single quatation use කරන්න පුලුවන්. නමුත් JSON එක use කරන්න ඕනේ double quatation.

Reading JSON data :

1) import json.

2) #Specify the path to your JSON file.

```
json_file_path = 'example_1.json'
```

3) #Open the JSON file for reading.

```
with open(json_file_path, 'r') as json_file:
    data = json.load(json_file)
```

4) #Now 'data' contains the contents of the JSON file as a python dictionary.

```
print(data, type(data))
```

* The type is 'dictionary'

Writing JSON data

```
1) import json.
2) # Create a python dictionary to represent your data.
   data = {
       "name": "John Doe",
       "age": 30,
       "city": "New York"
   }
3)  json_file_path = 'test.json'
4) with open(json_file_path, 'w') as json_file:
       # serialize the dictionary to a JSON-formatted string.
       json_data = json.dumps(data, indent=4)
       # Write the JSON-formatted string to the file.
       json_file.write(json_data).
5) print(f"JSON data has been written to {json_file_path}")
```

8) You are given a json file name students.json which contains information about students & their grades. Your task is to,

1) Read the json file.
2) Display the name of all students who scored above 75.
3) Add a new student record to the file.
4) Save to updated data back to the JSON file.

```
import json
file_path = 'students.json'
with open(file_path, 'r') as json_file:
    students = json.load(json_file)

print(students, type(students))
for student in students:
    if student["grade"] > 75:
        print(student["name"])
name = input("Enter a name: ")
grade = input("Enter a grade: ")
new_entity = {"name": name, "grade": grade}
students.append(new_entity)
with open(file_path, 'w') as json_file:
    student = json.dumps(students, indent=4)
    json_file.write(student)
```

csv file.

- csv stands for Comma - Separated Values.
- Structured data store කරන්න csv file use කරනවා.
- csv by default open වෙන්නෙ .xcel වලින්.

Reading csv data.

```
import csv
# Specify the path to the csv file you want to read.
csv_file_path = "customers-100.csv"
# Open the csv file for reading.
with open(csv_file_path, 'r', newline='') as csv_file:
    # Create a csv reader object
    csv_reader = csv.reader(csv_file)
    # Iterate through each row in the csv file
    for row in csv_reader:
        # 'row' is a list representing a row of data
        print(row)

print(csv_reader, type(csv_reader))
```

The reference

<class '_csv.reader'>

newline = '' → මේකෙන් වෙන්වෙන line ending වල smoothly handle කරන්න.

(When openning a file, you are telling python to use a universal newline mode.)

Writing csv data.

```
import csv
# Specify the path to the csv file you want to create
csv_file_path = 'sample_output.csv'
# Data to be written to the csv file (a list of dictionaries)
data = [
    {"Name": "John Doe", "Age": 30, "City": "New York"},
    {"Name": "Jane Smith", "Age": 28, "City": "Los Angeles"}
]
# Define the csv filednames (column names)
fieldnames = ["Name", "Age", "City"]
# Open the csv file for writing.
with open(csv_file_path, 'w', newline='') as csv_file:
    # Create a csv writer object
    csv_writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
    # Write the header (column names) to the csv file.
    csv_writer.writeheader()
```

headers වල අඩංගු තොරතුරු

leadcode

```
# Write the data rows to the csv file
for row in data:
    csv_writer.writerow(row)

print(f"csv data has been written to {csv_file_path}")
```

How to read as dictionaries.

```
with open('sample_output.csv', 'r') as file:
    rows = csv.DictReader(file)
    for row in rows:
        print(row, type(row))
```

Q) You are tasked with creating a python program to manage a company's employ records stored in csv files. The program should read the employee details from a csv file, filter the records based on a condition and write the filters record to a new csv file.

1) Input file - employee.csv

Contain the following field,

EmployeeId, Name, Department, Salary

1, John, IT, 60000

2, Jane, HR, 55000

3, Brown, Finance, 70000

high-salary-employees.csv

You will create this file. It should contain record of employees whose salaries are above the field should remains the same.

task :-

1) Read the input file. Use csv.reader to read employees.csv and display all the records on the console.

2) Filter the record identify employees with the Salary > 57000

3) Write the filtered record. Use csv.writer to write filtered record to a new file name high_salary-employees.csv.

Answer :

```
import csv
file_path = 'employee.csv'

with open(file_path, 'r', newline='') as csv_file:
    csv_reader = csv.reader(csv_file)
    for row in csv_reader:
        print(row)

filtered_employees = []
with open(file_path, 'r', newline='') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    for row in csv_reader:
        if int(row["Salary"]) > 57000:
            filtered_employees.append(row)
            print(row)

csv_file_path = 'high-salary-employees.csv'
with open(csv_file_path, 'w', newline='') as csv_file:
    field_names = ["EmployeeId", "Name", "Department",
                   "Salary"]
    writer = csv.DictWriter(csv_file, fieldnames = field_nam
    writer.writeheader()
    writer.writerows(filtered_employees)
```

Nested Function

(A function inside the function)

```
Ex: def outer(x):
        def inner(y):
            return x+y
        return inner

    temp_value = outer(5)
```

```
def inner(y):
    return 5+y
```

```
print( temp_value(7)) ⟶ def inner(7)
                            return 5+7
```

⇒ Output ⇒ 12

Pass Function as Argument

```
Ex:- def add(x,y):
         return x+y
     def calculate( func, x, y):
         return func(x, y)
     result = calculate(add, 4, 6)
     print(result)
```

↘ Output ⇒ 10.

Python Decorators.

Ex:-

```
def ~~pretty~~ make_pretty(func):
    def inner():
        print("I got decorated")
        func()
    return inner

def ordinary():
    print("I am ordinary")

get_decorated = make_pretty(ordinary)
get_decorated()
```

```
~~make~~ def inner():
    print("I got decorated")
    ordinary()
```

Output ⇒

I got decorated

I am ordinary.

@ Symbol with Decorators

```
def make-pretty (func):
    def inner():
        print ("I got decorated")
        func()
    return inner

@make-pretty
def ordinary():
    print ("I am ordinary")

ordinary()
```

මේ annotation එක දැම්මම එතන ඉඳන් function එකෙන් wrapup වෙලා තමයි output එක එන්නේ.

Output ⇒

```
I got decorated
I am ordinary
```

Write a python program with the following requirements.

1) Create a decorator check-positive that,
   i) Checks if the input to a function is a positive number.
   ii) If input is not positive print the message line input must be a positive number
2) Use this decorator on a function, ~~Calculate_squire_ro~~ calculate_squire_root
   - Takes one number as a input
   - Returns the squire root of the input number.

Date 17 12 2024

Python Libraries and Data Processing.

What is a library?

libraries are collections of pre-written code and modules that provide a wide range of functionality to simplify and expedite the development of software.

These libraries contain functions, classes and methods that can be imported and used in your Python programs, reducing the need to reinvent the wheel for common tasks.

Module vs Library.

A module is a file with .py extention.

No:

Date:

Selecting relevent Libraries :

Common open-source

① NumPy Library

- Numpy use කරලයි Arrays process කරන්න.
- Processing time එක list කියන එකේදී 50% speed.
- එකෙන් use කරන්නේ scientific computing වලට.

```
pip install numpy
```

Where is the Numpy Codebase ?

https://github.com/numpy/numpy

Numpy reference

https://numpy.org/doc/stable/reference

How to import numpy library

```
import numpy as np
```

Array Creation

```
a = np.array([2,3,4])
```

Should pass a list

```
print(a)
print(type(a))   # <class 'numpy.ndarray'>
```

Creating Numpy Arrays

```
array1 = np.array([1,2,3,4,5])
array2 = np.array([6,7,8,9,10])
```

Performing Basic Operations

Addition

```
result_addition = array1 + array2
print(result_addition)
```

Output → [7 9 11 13 15]

~~Squire~~ Square Root

```
result_sqrt = np.sqrt(array1)
```

Summation

```
result_sum = np.sum(array1)
```

Mean (Average)

result_mean = np.mean(array2)

Maximum and Minimum

result_max = np.max(array1)

result_min = np.min(array2)

Multidimensional Arrays.

Ex:- array-1 = np.array([[1,2,3],[4,5,6]])

this is a 2D array.

array-2 = np.array([[[1,2,3],[4,5,6]],
[[1,2,3],[4,5,6]]])

this is a 3D array.

. Shape Attribute.

How to check the shape of the array (ఎన్ని వరుసలు ఎన్ని కాలమ్స్)

Ex:- print(array-1.shape)

output ⇒ (2, 3) ⇒ This is a tuple.

rows columns

Output ⇒ [[1 2 3]
[4 5 6]]

```
print (array_2.shape)
```

Output → (2, 2, 3)

- 2 (first) → Outer most element amount
- 2, 3 → Inner elemet dimension.

Output →

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
```

• dtype Attribute

The dtype attribute specifies the data type of the elements stored in the Numpy array, such as int32, float64, or bool

```
array = np.array([1, 2, 3])
print(array.dtype)  → Output : int64.
```

(Array එකේ ~~data~~ elements වල data type එක check කරයි.)

• size Attribute

Returns the number of elements in the array.

```
print(array.size)
```

output : 3

. ndim attribute .

Returns the number of dimensions (axes) in the array.

(2D array)

print (multi_array1.ndim) ⇒ Output ⇒ 2

print (multi_array2.ndim) ⇒ Output ⇒ 3

(3D array)

Numpy Array Indexing .

```
array_2 = np.array([[[1,2,3,4,5],[6,7,8,9,10]]])
print('2nd element on 1st row: ', array_2[0,1])
```

Output ⇒ 2 //

(Q) Create a 2D numpy array which has dimention 4×5, which contains the numbers 1 to 20

Perform the following on this array.

1) Add 10 to every element.
2) Multiply every element by 2.
3) Calculate the squre of each element.

```
import numpy as np
array1 = np.array([[1,2,3,4,5], [6,7,8,9,10],
                   [11,12,13,14,15], [16,17,18,19,20]])

print(array1 + 10)
print(array1 * 2)
print(np.sqrt(array1))
```

Boolean Indexing in Numpy

* Filter elements.

Ex:-
```
import numpy as np.
# Create a Numpy Array.
arr = np.array([10, 20, 30, 40, 50])
# Apply a condition
 (This condition will apply to every element in the
  array.)
condition = arr > 25
print(condition) ⇒ Output [False False True True True]

# Use boolean indexing to filter values.
filtered = arr[condition]
print(filtered) ⇒ Output [30 40 50].
```

```
# We can do above code within a single line.
filtered_array = arr[arr > 25]
print(filtered_array) ⇒ Output [30 40 50].
```

Combine Multiple Conditions.

```
# Create a numpy array.
arr = np.array([10, 20, 30, 40, 50])

# Values greater than 20 and less than 50
filtered_combined = arr[(arr > 20) & (arr < 50)]
print(filtered_combined) ⇒ Output: [30 40].
```

Array Initialization - numpy.zeros()

Syntax : `np.zeros(shape, dtype = float)`

- shape → required
- dtype → Optional (by default)
  - int
  - float
  - complex

Ex:-

```
array_1 = np.zeros(5)
print(array_1) ⇒ output: [0 0 0 0 0]

array_2 = np.zeros((2,3))
print(array_2) ⇒ Output: [[0 0 0],
                          [0 0 0]]
```

Array Initialization - numpy.full()

Syntax : np.full (shape, fill_value)

Ex:- array_1 = np.full (4, 7)

print (array_1) Output ⇒ [7 7 7 7]

array_2 = np.full ((2,3), 8)

print (array_2) Output ⇒ [[8 8 8],
[8 8 8]]

Array Initialization - ~~np~~ numpy.empty()

Syntax : np.empty (shape, dtype = float).

Ex:- array_1 = np.empty ((2,3))

print (array_1)

Output ⇒ [[2.000 3.004 2.4]
[0.00+ 4. 5.]]

Why Use numpy.empty()

1) Performance

(මේකෙදි assign වෙන්නේ cache එකේ තියෙන values. ඒ නිසා speed.)

2) Flexibility.

Q2) Create a 1D numpy array with values from 1 to 20. Use boolean index into extract all even numbers.

Q3) Create a ~~numb~~ 1D numpy array with elements 10, 20, 30, 40, 50. Use boolean index into extrac all elements greater than the mean of the array.

Answers :

Q2)
```
import numpy as np

arr = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
                                    17,18,19,20]])

filtered_array = arr[arr % 2 == 0]

print(filtered_array)
```

Output ⇒ [2 4 6 8 10 12 14 16 18 20].

Q3)
```
import numpy as np

arr1 = np.array([10,20,30,40,50])

mean = np.mean(arr1)

filtered_array = arr1[arr1 > mean]

print(filtered_array)
```

Output ⇒ [40 50]

2) Pandas Library.

How to install :

pip install pandas.

See pandas documentation.

https://pandas.pydata.org/docs/user_guide/index.html

What is DataFrame in Pandas.

- It's a two-dimensional structure with rows and columns.

(Stuctured data store ಮಾಡುವ ಡೇಟಾ framework ಥರ)

- Each column can have different data types.

① Create a DataFrame.

```
import pandas as pd
# Creating a DataFrame from a dictionary.
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [25, 30, 35, 40]}
df = pd.DataFrame(data)
print(df)
```

Output ⇒

	Name	Age
0	Alice	25
1	Bob	30
2	Challe	35
3	David	40

මේ row index වලට හරිය කෙනෙක් නම් දාගන්න පුලුවන්.

② Accessing Cells in a Dataframe.

1) Using loc

Ex :- print (df.loc [1, "Name"])

Output ⇒ Bob.

2) Using iloc

Ex :- print (df.iloc [1, 0])

Output ⇒ Bob

3) locate row.

Ex :- print (df.loc [1])

Output ⇒ Name Bob.

Age 30

Name: 1 , dtype : object.

4) Return multiple rows

```
print(df.loc[[0, 1]])
```

Output →

	Name	Age
0	Alice	25
1	Bob	30

③ Named Indexes

Ex:-

```
data = {
    "calories": [420, 380, 390],
    "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index=["day1", "day2", "day3"])
print(df)
```

Output →

	Calories	duration
day1	420	50
day2	380	40
day3	390	45

refer to the named index:

```
print ( df.loc [" day2" ])
print ( df.loc [["day1", "day2"]])
                    # Return two rows.
```

Output ⇒

```
calories  380
duration   40
Name: day2 , dtype: int64.
```

	calories	duration
day1	420	50
day2	380	40

Get details from the column-name.

```
print ( df ['calories'])
```

Output ⇒

```
day1  420
day2  380
```

Useful Attributes

① DataFrame.shape

Ex:- import pandas as pd

df = pd.DataFrame({'A': [1,2], 'B': [3,4]})

print(df.shape)

Output ⇒ (2, 2)

(2 → rows, 2 → columns.)

② DataFrame.columns.

Ex:- ~~df.~~ print(df.columns).

Output ⇒ Index(['calories', 'duration'], dtype='obje

print(list(df.columns)).

output ⇒ ['calories', 'duration'].

③ DataFrame.size

print(df.size)

output ⇒ 6

④ DataFrame.values

Returns the data as a NumPy array.

```
print ( df.values , type ( df.values ))
```

Output ⇒

```
[[ 400  50]
 [ 380  40]
 [ 390  45]] <class 'numpy.ndarray'>
```

⑤ DataFrame.len()

The len() function is used to return the number of rows in the DataFrame.

```
print ( len ( df ))
```

Output ⇒ 3

⑥ DataFrame.mean()

~~print (df.mean())~~

```
data = {
    'Age' : [25, 30, 35, 40 ],
    'Salary' : [ 50000, 60000, 70000, 80000 ]
df = pd.DataFrame ( data )
print ( df.mean() )
```

Output ⇒

(P1)

Order ID	Product	Category	Quantity	Price per Unit	Region
101	Laptop	Electronics	2	1000	North
102	Smartphone	Electronics	5	800	South
103	Desk chair	Furniture	10	150	East
104	Monitor	Electronics	4	200	West
105	Book Shelf	Furniture	2	300	North

You are a data analysis for a retail company. The company has provided you with the above ~~above table~~ table with sales data. Your task is to analyse the sales data using python and pandas to answer some key business questions.

① Calculate the total revenue for each order.
Add a new column total_revenue to store the total revenue for each order.

② Identify the best selling product.
Find the product with the heighest total sales revenue.

```
import pandas as pd

data = {'Product': ['','','','',''],
        'Category': ['','','','',''],
        'Quantity': [,,,,],
        'PricePerUnit': [,,,,],
        'Region': ['','','','','']}
df= pd.DataFrame( data, index = [101,102,103,104,105])
print(df)
qty_list = df['Quantity']
price_per_unit_list = df['PricePerUnit']
total = list(qty_list * price_per_unit_list)
df['total_revenue'] = total
print(df).

filteredDf = df [df['total_revenue'] == df['total_revenue'].
                                                max()]
print(filteredDf)
```

⑥ DataFrame.mean()

Ex:- data = {
'Age': [25, 30, 35, 40],
'Salary': [50000, 60000, 70000, 80000]
}

df = pd.DataFrame(data)

print(df.mean())

~~⑦ DataF~~ Output ⇒

Age 32.5

Salary 65000.0

dtype: float64

⑦ DataFrame.sum()

print(df.sum())

Output ⇒

Age 130

Salary 260000

dtype: int64

⑧ DataFrame . describe ()

print (df. describe())

std - standard deviation (සම්මත අපගමනය)

Output ⇒ .

	Age	Salary .
count	4.000000	4.000000
mean	32.500000	65000.00000
std	6.454972	12909. 944487
min	25.000000	50000. 000000 .
25%		
50%		
75%		
max	40.000000	80000. 000000

Quarters (චතුර්තක).

* DataFrame. describe() method එකෙන් ~~stas~~ statistical description එකක් ලබාදේ.

Filtering Rows in a DataFrame.

Ex:-
```
data1 = {
    'Name': ['Alice', 'Bob', 'charlie', 'David'],
    'Age': [24, 27, 22, 32],
    'Score': [85, 70, 90, 88]
}
df1 = pd.DataFrame(data1)
filteredDf1 = df1[df1['Age'] > 25]
print(filteredDf1, type(filteredDf1))
```

Output ⇒

	Name	Age	Score
1	Bob	27	70
3	David	32	88

<class 'pandas.core.frame.DataFrame'>

Adding a new Row at the end to the dataframe.

Ex:-
```
data2 = {
    'A': [1, 2],
    'B': [3, 4]
}

df2 = pd.DataFrame(data2)
df2.loc[len(df2)] = [5, 6]
print(df2)
```

Output ⇒

	A	B
0	1	3
1	2	4
2	5	6

Dropping a column or row.

```
df2_drop_row = df2.drop(2, axis = 0)
print(df2_drop_row)
df2_drop_column = df2.drop('B', axis = 1)
print(df2_drop_column)
```

Output ⇒

	A	B
0	1	3
1	2	4

	A
0	1
1	2
2	5

syntax : Dataframe. drop (lables, axis=1, inplace = False)

Set axis = 1 → drop the column.

Set axis = 0 → drop the row

Inplace = True - Affect the original one. (remove original data). (Modify the original array)

Inplace = False - By default this is false. it doesn't affect the original data.

Read csv file.

Ex:-
```
df = pd.read_csv ('sample.csv')
print (df)
```

Read JSON file

Ex:-
```
df = pd.read_json ('sample.json')
print (df)
```

csv file.

- csv stands for Comma - Separated Values.
- structured data store කරන්න csv file use කරනවා.
- csv by default open වෙන්නෙ .xcel වලින්.

Reading csv data.

```
import csv
# Specify the path to the csv file you want to read
csv_file_path = "customers-100.csv"
# Open the csv file for reading.
with open(csv_file_path, 'r', newline='') as csv_file:
    # Create a csv reader object.
    csv_reader = csv.reader(csv_file)
    # Iterate through each row in the csv file
    for row in csv_reader:
        # 'row' is a list representing a row of data
        print(row)

print(csv_reader, type(csv_reader))
```

The reference

<class '_csv.reader'>

newline = '' → මේකෙන් වෙන්නේ line ending එක smoothly handle කරන එක.

(when openning a file, you are telling python to use a universal newline mode.)

Writing csv data.

```
import csv
# Specify the path to the csv file you want to create
csv_file_path = 'sample_output.csv'
# Data to be written to the csv file (a list of
  dictionaries)
data = [
   {"Name": "John Doe", "Age": 30, "City": "New York"
   {"Name": "Jane Smith", "Age": 28, "City": "Los Angeles
]
# Define the csv filednames (column names)
fieldnames = ["Name", "Age", "City"]
# Open the csv file for writing.
with open(csv_file_path, 'w', newline='') as csv_file
     # Create a csv writer object
     csv_writer = csv.DictWriter(csv_file, fieldnames
                                          = fieldnam
     # Write the header (column names) to the csv file.
     csv_writer.writeheader()
```

headers ටික එක පේළියට ලියනවා.

```
# Write the data rows to the CSV file
for row in data:
    csv_writer.writerow(row)

print(f"CSV data has been written to {csv_file_path}")
```

How to read as dictionaries.

```
with open('sample_output.csv', 'r') as file:
    rows = csv.DictReader(file)
    for row in rows:
        print(row, type(row))
```

Q) You are tasked with creating a python program to manage a company's employ records stored in csv files. The program should read the employee details from a csv file, filter the records based on a condition and write the filters record to a new csv file.

1) Input file - employee.csv

Contain the following field,

EmployeeId, Name, Department, Salary

1, John, IT, 60000

2, Jane, HR, 55000

3, Brown, Finance, 70000

high_salary_employees.csv

You will create this file. It should contain record of employees whose salaries are above the field should remains the same.

task :-

1) Read the input file. Use csv.reader to read employees.cs and display all the records on the console.

2) Filter the record identify employees with the Salary > 57000

3) Write the filtered record. Use csv.writer to write filtered record to a new file name high_salary_employees.csv.

Answer :

```
import csv
file_path = 'employee.csv'

with open(file_path, 'r', newline='') as csv_file:
    csv_reader = csv.reader(csv_file)
    for row in csv_reader:
        print(row)

filtered_employees = []
with open(file_path, 'r', newline='') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    for row in csv_reader:
        if int(row["Salary"]) > 57000:
            filtered_employees.append(row)
            print(row)

csv_file_path = 'high-salary-employees.csv'
with open(csv_file_path, 'w', newline='') as csv_file:
    field_names = ["EmployeeId", "Name", "Department",
                   "Salary"]
    writer = csv.DictWriter(csv_file, fieldnames = field_names)
    writer.writeheader()
    writer.writerows(filtered_employees)
```

Nested Function

(A function inside the function)

Ex:
```
def outer(x):
    def inner(y):
        return x+y
    return inner
```

```
temp_value = outer(5)
```

```
def inner(y):
    return 5+y
```

```
print( temp_value(7)) ⟶ def inner(7)
                            return 5+7
```

⇒ Output ⇒ 12

Pass Function as Argument

Ex:-
```
def add(x,y):
    return x+y
def calculate( func, x, y):
    return func(x,y)
result = calculate(add, 4, 6)
print(result)
```

Output ⇒ 10.

Python Decorators.

Ex:-

```
def ~~pretty~~ make_pretty(func):
    def inner():
        print("I got decorated")
        func()
    return inner

def ordinary():
    print("I am ordinary")

get_decorated = make_pretty(ordinary)
get_decorated()
```

```
~~make~~ def inner():
    print("I got decorated")
    ordinary()
```

Output ⇒

I got decorated
I am ordinary.

@Symbol with Decorators

Ex:-
```
def make-pretty (func):
    def inner():
        print("I got decorated")
        func()
    return inner

@make_pretty
def ordinary():
    print("I am ordinary")

ordinary()
```

මේ annotation එක දැම්මම එකට අදාළ function එකෙන් wrapup වෙලා කෙලින් output එක එන්නෙ.

Output ⇒

```
I got decorated
I am ordinary
```

Write a python program with the following requirements.

1) Create a decorator check-positive that,
   i) Checks if the input to a function is a positive number.
   ii) If input is not positive print the message line input must be a positive number
2) Use this decorator on a function, ~~Calculate_squireRt~~ calculate_squire_root
   - Takes one number as a input
   - Returns the squire root of the input number.

17 12 2024

Python Libraries and Data Processing.

What is a library?

libraries are collections of pre-written code and modules that provide a wide range of functionality to simplify and expedite the development of software.

These libraries contain functions, classes and methods that can be imported and used in your Python programs, reducing the need to reinvent the wheel for common tasks.

Module vs Library.

A module is a file with .py extention.

Selecting relevent Libraries :

Common open-source

① NumPy Library

- Numpy use කරන්නේ Arrays process කරන්න.
- Processing time එක list එකට සාපේක්ෂව 50% speed.
- බොහෝම use කරන්නේ scientific computing වලට.

```
pip install numpy
```

Where is the Numpy Codebase?

https://github.com/numpy/numpy

Numpy reference

https://numpy.org/doc/stable/reference

How to import numpy library

```
import numpy as np.
```

Array Creation.

```
a = np.array([2,3,4])
```

should pass a list.

```
print(a)
print(type(a))
```

<class 'numpy.ndarray'>.

Creating Numpy Arrays.

```
array1 = np.array([1,2,3,4,5])
array2 = np.array([6,7,8,9,10])
```

Performing Basic Operations.

Addition

```
result_addition = array1 + array2
print(result_addition)
```

Output ⇒ [7 9 11 13 15]

~~Squire~~ Square Root

```
result_sqrt = np.sqrt(array1)
```

Summation.

```
result_sum = np.sum(array1)
```

```
# Mean (Average)
result_mean = np.mean(array2)

# Maximum and Minimum
result_max = np.max(array1)
result_min = np.min(array2)
```

Multidimensional Arrays.

```
Ex:- array_1 = np.array([[1,2,3],[4,5,6]])
# this is a 2D array.

array_2 = np.array([[[1,2,3],[4,5,6]],
                    [[1,2,3],[4,5,6]]])
# this is a 3D array.
```

• Shape Attribute.

How to check the shape of the array (අංශ ගණන).

```
Ex:- print(array_1.shape)
```

output ⇒ (2, 3) ⇒ This is a tuple.

(2 ↑ rows, 3 ↑ columns.)

```
Output ⇒ [[1 2 3]
          [4 5 6]]
```

Ex: print (array-2. shape)

Output ⇒ (2, 2, 3)

Outer most element amount

Inner elemet dimension.

Output ⇒ [[[1 2 3]
[4 5 6]]

[[1 2 3]
[4 5 6]]]

• dtype Attribute.

The dtype attribute specifies the data type of the elements stored in the Numpy array, such as int32, float64, or bool

array = np.array ([1, 2, 3])

print (array.dtype) ⇒ output : int64.

(Array ~~කොහොම~~ ~~data~~ elements වල data type එක check කරනවා.)

• Size Attribute.

Returns the number of elements in the array.

Print (array.size)

output : 3

• ndim attribute.

Returns the number of dimensions (axes) in the array.

print (multi_array1.ndim) ⇒ Output ⇒ 2 ← (2D array)

print (multi_array2.ndim) ⇒ Output ⇒ 3 ← (3D array)

NumPy Array Indexing.

```
array_2 = np.array([[1,2,3,4,5],[6,7,8,9,10]])
print('2nd element on 1st row:', array_2[0,1])
```

Output ⇒ 2 //

(Q) Create a 2D numpy array which has dimention 4×5, which contains the numbers 1 to 20

Perform the following on this array.

1) Add 10 to every element.
2) Multiply every element by 2.
3) Calculate the squre of each element.

```
import numpy as np
array1 = np.array([[1,2,3,4,5],[6,7,8,9,10],
                   [11,12,13,14,15],[16,17,18,19,20]])
print(array1 + 10)
print(array1 * 2)
print(np.sqrt(array1))
```

Boolean Indexing in Numpy

* Filter elements.

Ex:-
```
import numpy as np.
# Create a Numpy Array.
arr = np.array([10, 20, 30, 40, 50])
# Apply a condition
 (This condition will apply to every element in the
  array.)
 condition = arr > 25
 print(condition) ⇒ Output [False False True True Tru

# Use boolean indexing to filter values
 filtered = arr[condition]
 Print(filtered) ⇒ Output [30 40 50].
# We can do above code within a single line.
 filtered_array = arr[arr > 25]  [30 40 50].
```

Combine Multiple Conditions.

```
# Create a numpy array.
arr = np.array([10, 20, 30, 40, 50])

# Values greater than 20 and less than 50
filtered_combined = arr[(arr > 20) & (arr < 50)]
print(filtered_combined) ⇒ Output: [30 40].
```

Array Initialization - numpy.zeros()

Syntax : `np.zeros(shape, dtype = float)`

- shape → required
- dtype → Optional (by default)
  - int
  - float
  - complex

Ex:-

```
array-1 = np.zeros(5)
print(array-1) ⇒ output: [0 0 0 0 0]

array-2 = np.zeros((2,3))
print(array-2) ⇒ Output: [[0 0 0],
                          [0 0 0]]
```

Array Initialization - numpy.full()

Syntax : np.full (shape, fill_value)

Ex:- array_1 = np.full (4, 7)

print (array_1) Output ⇒ [7 7 7 7]

array_2 = np.full ((2,3), 8)

print (array_2) Output ⇒ [[8 8 8],
[8 8 8]]

Array Initialization - ~~np~~ numpy.empty()

Syntax : np.empty (shape, dtype = float)

Ex:- array_1 = np.empty ((2,3))

print (array_1)

Output ⇒ [[2.000 3.004 2.4]
[0.00+ 4. 5.]]

Why Use numpy.empty()

1) Performance
(මේකෙදී assign වෙන්නේ cache එකේ තියෙන values. ඒ නිසා speed.)
2) Flexibility.

Q2) Create a 1D numpy array with values from 1 to 20. Use boolean index into extract all even numbers.

Q3) Create a ~~numb~~ 1D numpy array with elements 10, 20, 30, 40, 50. Use boolean index into extra all elements greater than the mean of the array.

Answers :

Q2)
```
import numpy as np

arr = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
                17,18,19,20])

filtered_array = arr[arr % 2 == 0]

print(filtered_array)
```

Output ⇒ [2 4 6 8 10 12 14 16 18 20]

Q3)
```
import numpy as np

arr1 = np.array([10,20,30,40,50])

mean = np.mean(arr1)

filtered_array = arr1[arr1 > mean]

print(filtered_array)
```

Output ⇒ [40 50]

2) Pandas Library.

How to install :

pip install pandas.

See pandas documentation.

https://pandas.pydata.org/docs/user_guide/index.html

What is DataFrame in Pandas.

- It's a two-dimensional structure with rows and columns.

(Stuctured data store කරන්න පුළුවන් framework එකක්.)

* Each column can have different data types.

① Create a DataFrame.

```
import pandas as pd
# Creating a DataFrame from a dictionary.
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [25, 30, 35, 40]}
df = pd.DataFrame(data)
print(df)
```

Output ⇒

	Name	Age
0	Alice	25
1	Bob	30
2	Chalie	35
3	David	40

මේ row index වලට පුළුවන් නම් දාන්න පුලුවන්

② Accessing Cells In a Dataframe.

1) Using loc

Ex :- print (df. loc [1, "Name"])

Output ⇒ Bob.

2) Using iloc

Ex :- print (df. iloc [1, 0])

Output ⇒ Bob

3) locate row.

Ex :- print (df. loc [1])

Output ⇒ Name Bob.
Age 30
Name: 1 , dtype : object.

4) Return multiple rows.

```
print(df.loc[[0,1]])
```

Output ⇒

	Name	Age
0	Alice	25
1	Bob	30

⑧ Named Indexes

Ex:-

```
data = {
    "calories": [420, 380, 390],
    "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index=["day1", "day2", "day3"])

print(df)
```

Output ⇒

	Calories	duration
day1	420	50
day2	380	40
day3	390	45

```
# refer to the named index:
print(df.loc["day2"])
print(df.loc[["day1", "day2"]])
                    # Return two rows.
```

Output ⇒

```
calories  380
duration   40
Name: day2, dtype: int64.
```

	calories	duration
day1	420	50
day2	380	40

```
# Get details from the column-name.
print(df['calories'])
```

Output ⇒

```
day1  420
day2  380
```

Super Intelligence

Useful Attributes

① DataFrame.shape

Ex:- import pandas as pd

df = pd.DataFrame({'A': [1,2], 'B': [3,4

print(df.shape)

Output ⇒ (2, 2)

rows Columns.

② DataFrame.columns.

Ex:- ~~df.~~ print(df.columns).

Output ⇒ Index(['calories', 'duration'], dtype='ob

print(list(df.columns)).

output ⇒ ['calories', 'duration'].

③ DataFrame.size

print(df.size)

output ⇒ 6

④ DataFrame.values

Returns the data as a NumPy array.

```
print ( df.values , type ( df.values ))
```

output →

```
[[ 420  50]
 [ 380  40]
 [ 390  45]] <class 'numpy.ndarray'>
```

⑤ DataFrame.len()

The len() function is used to return the number of rows in the DataFrame.

```
print ( len (df))
```

output ⇒ 3

⑥ DataFrame.mean()

~~print (df.mean())~~

```
data = {
    'Age' :[25, 30, 35, 40 ],
    'Salary' : [ 50000, 60000, 70000 , 80000 ]
df = pd. DataFrame ( data )
print ( df.mean() )
```

Output ⇒

Q1)

Order ID	Product	Category	Quantity	Price Per Unit	Region
101	Laptop	Electronics	2	1000	North
102	Smartphone	Electronics	5	800	South
103	Desk chair	Furniture	10	150	East
104	Monitor	Electronics	1	200	West
105	Book Shelf	Furniture	2	300	North

You are a data analysis for a retail company. The company has provided you with the above ~~above table~~ table with sales data. Your task is to analyse the sales data using python and pandas to answer some key business questions.

① Calculate the total revenue for each order.
Add a new column total_revenue to store the total revenue for each order.

② Identify the best selling product.
Find the product with the heighest total sales revenue.

```
import pandas as pd

data = {'Product': ['','','','',''],
        'Category': ['','','','',''],
        'Quantity': [,,,,],
        'PricePerUnit': [,,,,],
        'Region': ['','','','','']}
df= pd.DataFrame( data, index = [101,102,103,104,105]
print(df)
qty_list = df['Quantity']
price_per_unit_list = df['PricePerUnit']
total = list(qty_list * price_per_unit_list)
df['total_revenue'] = total
print(df).

filteredDf = df[df['total_revenue'] == df['total_revenue
                                          max(
print(filteredDf)
```

⑥ DataFrame.mean()

Ex:- data = {

'Age': [25, 30, 35, 40],

'Salary': [50000, 60000, 70000, 80000]

}

df = pd.DataFrame(data)

print(df.mean())

⑦ ~~DataF~~ Output ⇒

Age 32.5

Salary 65000.0

dtype: float64

⑦ DataFrame.sum()

print(df.sum())

Output ⇒

Age 130

Salary 260000

dtype: int64

⑨ DataFrame . describe ( )

print (df. describe ( ))

std - standard deviation (සම්මත අපගමනය)

Output ⇒ .

	Age	Salary .
count	4.000000	4.000000
mean	32.500000	65000.00000
std	6.454972	12909.944487
min	25.000000	50000.000000.
25%		
50%		
75%		
max	40.000000	80000.000000

Quarters (චතුර්තක).

* DataFrame. describe ( ) method එකෙන් ~~stas~~ statistical description එකක් ලබාදේ.

Filtering Rows in a DataFrame.

Ex:-
```
data1 = {
    'Name': ['Alice', 'Bob', 'charlie', 'David'],
    'Age' : [24, 27, 22, 32],
    'Score': [85, 70, 90, 88]
}
df1 = pd.DataFrame(data1)
filteredDf1 = df1[df1['Age'] > 25]
print(filteredDf1, type(filteredDf1))
```

Output ⇒

	Name	Age	Score
1	Bob	27	70
3	David	32	88

<class 'pandas.core.frame.DataFrame'>

Adding a new Row at the end to the dataframe.

Ex:-
```
data2 = {
    'A' : [1, 2],
    'B' : [3, 4]
}
df2 = pd.DataFrame(data2)
df2.loc[len(df2)] = [5, 6]
print(df2)
```

Output ⇒

	A	B
0	1	3
1	2	4
2	5	6

Dropping a column or row.

```
df2_drop_row = df2.drop(2, axis = 0)
print(df2_drop_row)
df2_drop_column = df2.drop('B', axis = 1)
print(df2_drop_column)
```

Output ⇒

	A	B
0	1	3
1	2	4

	A
0	1
1	2
2	5

syntax : Dataframe.drop(lables, axis=1, inplace=False)

Set axis = 1 → drop the column.

Set axis = 0 → drop the row

inplace = True - Affect the original one.
(remove original data).
(Modify the original array)

inplace = False - By default this is false.
it doesn't affect the original data.

Read csv file.

Ex:-
```
df = pd.read_csv('sample.csv')
print(df)
```

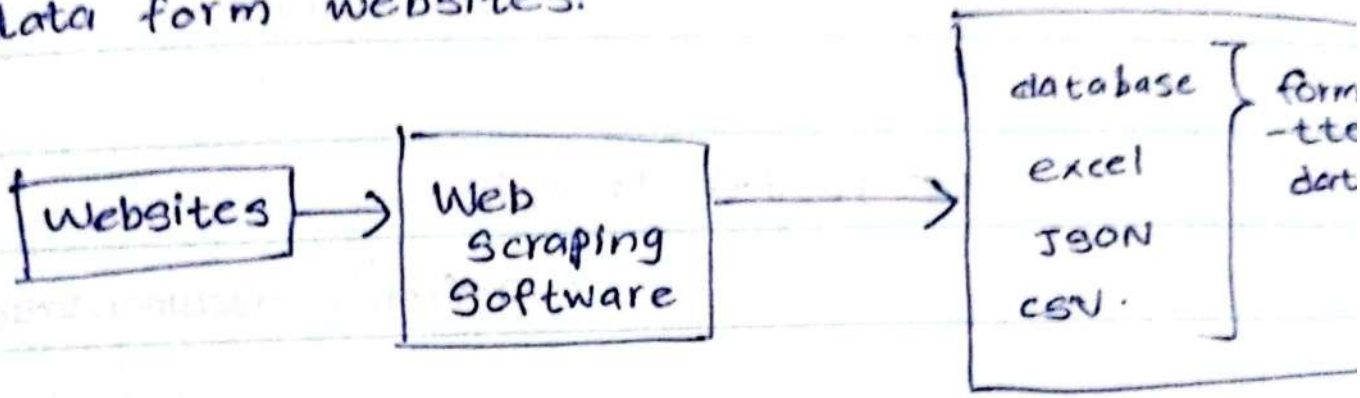
Read JSON file

Ex:-
```
df = pd.read_json('sample.json')
print(df)
```

07 01 2025

Web Scrapping.

Web scrapping is the process of extracting data form websites.



Core libraries for web scraping

- Requests Library -

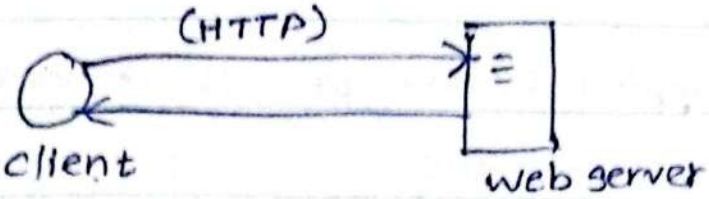
  For making HTTP requests to get web pages.

- Beautiful soup -

  For parsing HTML and XML documents.

Requests Library.

python -m pip install requests.



requests.readthedocs.io → (Official documentation)

Basic Usage: Sending GET Requests.

Ex:-
```
import requests.
response = requests.get('https://api.github.events')
```
- `print(response)`

Output ⇒ < Response [200] >

- `print(response.status_code)`.

output => 200. → මේක int එකක්.

Headers

- `print(response.headers)`

↳ මේකෙන් එන output එක dictionary එකක්.

- You can access individual headers.

```
content_type = response.headers['Content-Type']
print(content_type)
```

- To see the content of the url. (get)

```
print(response.text)
```

- Convert a content to a dictionary using json()

```
if 'application/json' in response.headers.get('Content-T
    data = response.json()
    print(data)
```

Basic Usage: Sending POST Requests.

* Post is used to send data to the server to create or update a resource. It's the method of choice when you're submitting data, like filling out a form or uploading a file.

Ex:-
```
data = {'username': 'user',
        'password': 'pass'}

response = requests.post('https://httpbin.org/post', data)

if response.status_code == 200:
    print('Login Successful')
    print(response.text)
else:
    print('Login failed:', response.status_code)
```

URL Parameters.

* Simple GET with Parameters.

ඕනෙම url එකක් additional informations එවන්න පුළුවන්:-

base url → https://example.com/search?q=python&category=books

fx:-

```
import requests

# Example URL for documentation
url = 'https://example.com/search'

# Parameters you want to send
params = {'q': 'python', 'category': 'books'}

# Making the request
response = requests.get(url, params = params)

# The actual URL sent would be something like:
# https://example.com/search?q=python&categ
                                    =boo

print(response.url)
print(response.text)
```

Basic Authentication

```
import requests
# Example URL for demonstration
url = 'https://httpbin.org/get'
# Credentials
username = 'user'
password = 'pass'
# Sending the GET request with Basic
  Authentication
response = requests.get(url, auth=(username,
                                   password))

# check if the request was successful
if response.status_code == 200:
    print('Authentication successful!')
    print(response.text)
    # Print the content of the response.
else:
    print(f'Authentication failded.
          Status code: {response.status_co
```

Beautiful Soup Library

Beautiful soup is a python library designed for parsing HTML and XML documents.
(HTML එකක් code එක පූරා කොටස් වලට තිබුනා; ඒ කොටස් access කරනවා.)

install ⇒ pip install beautifulsoup4.

Official Documentation :-

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

How to import Beautiful soup Library :-

```
from bs4 import BeautifulSoup.
```

Example :-

```
from bs4 import BeautifulSoup
html_doc = """<html><head><title>The Dormouse's
        story</title></head>
        <body>
        <p class="title"><b>The Dormouse's
                story</b></p>
        .....
"""
soup = BeautifulSoup(html_doc, 'html.parser')
print(soup.title)
print(soup.p) ⇒ පළවෙනි p tag එක ලැබෙනවා
print(soup.a)
p_tag = soup.find('p')
print(p_tag)
p_tags = soup.find_all('p')
print(p_tag) ⇒ හැමෝම p tag ඔක්කොම ලැබෙනවා.
```

Question

(9) Fetch a webpage, check if the request was successful, and extract the main heading text from the page's content. You can assume that the base url is 'http://example.com'.

```
from bs4 import BeautifulSoup
import requests.

response = requests.get('http://example.com')
response_string = str(response.text)
if response.status_code == 200:
    soup = BeautifulSoup(response_string, 'html.parse
    print(soup.title)
else:
    print('Error loading', response.status_code).
    h1_tag = soup.find('h1')
    if h1_tag:
        print("h1 tag:", h1_tag)
    else:
        print("h1 tag not found")
else:
    print('Error loading', response.status_code).
```

API Integration.

Here's a comprehensive example using the Open-Meteo API (https://open-meteo.com) (Only for non commertial use)

Example :-

```
def fetch_weather( latitude, longitude, current_inf
    base_url = 'https://api.open-meteo.com/v1/fore
    param = {
        'latitude': latitude,
        'longitude': longitude,
        'current': current_info
    }
    response = requests.get(base_url, params=
    if response.status_code == 200:
        weather_data = response.json()
        return weather_data.
    else:
        print(f" Request failed with status code:
                                {response.status_code}
        return None.
print( fetch_weather (51.5074, -0.1278,
                    'temperature_2m'))
```

Python Debugging.

1) Syntax Errors. (Ex:- missing colan.)
2) Runtime Errors. (Ex:- Division by zero)
3) Logical Errors. (Ex:- Incorrect calculation logic)

Search → pdb debugger