Introduction to Python for Data Science

- Revision on Python
- Introduction to numpy, pandas, matplotlib, seaborn
- NumPy - Arrays, operations on arrays, important functions of numpy library
- Pandas - DataFrames, data manipulation - handling nulls, filtering values, merge & concat, drop columns & rows, replace values
- Matplotlib - Data Visualization Fundamentals
- Seaborn - Adanced Data visualization

Revision on Python

Ex. WAP to take rate and hours as input from user and print the gross pay based on following conditions -

- if worker is working for 40 hours then the gross pay is product of rate and hours
- if worker is working for more than 40 hours he will get 1.5 times the rate for additional hours

In [3]:
```
hrs = int(input("Enter number of hours - "))
rate = int(input("Enter rate per hour - "))

if hrs <= 40 :
    gross_pay = rate * hrs
else :
    gross_pay = rate * 40 + ((hrs-40) * rate * 1.5)
print(gross_pay)
```

```
4750.0
```

Ex. WAP to calculate BMI of a person for whom weight and height is entered as input from user.

In [5]:
```
weight = float(input("Enter your weight in kgs - "))
height = float(input("Enter your height in kgs - "))

bmi = weight/(height ** 2)
print(bmi)
```

```
22.03856749311295
```

Ex. Extend the BMI program and categories the person's health status

In [7]:
```
weight = float(input("Enter your weight in kgs - "))
height = float(input("Enter your height in kgs - "))

bmi = round(weight/(height ** 2), 2)
print("Your BMI - ", bmi)

if bmi < 18 :
```

```
    print("underweight")
elif bmi < 24 :
    print("Healthy")
elif bmi < 29 :
    print("Overweight")
else :
    print("Obese")
```

```
Your BMI -  22.04
Healthy
```

Ex. WAP to print the sum and product of numbers from 1-10

In [8]:
```
# Basic Approach
total = 0
product = 1
for i in range(1,11) :
    total += i
    product *= i
print(f"Total - {total} and Product - {product}")
```

```
Total - 55 and Product - 3628800
```

In [9]:
```
# Using built-in functions
import math
numbers = range(1, 11)
print(f"Total - {sum(numbers)} and Product - {math.prod(numbers)}")
```

```
Total - 55 and Product - 3628800
```

Ex. WAP to print(store in a new list) squares of numbers in the given list

In [10]:
```
# Basic Approach
numbers = [1, 3, 2, 6, 5, 4]
squares = []

for i in numbers :
    squares.append(i**2)
squares
```

Out[10]:
```
[1, 9, 4, 36, 25, 16]
```

In [11]:
```
# Comprehension approach
[i**2 for i in numbers]
```

Out[11]:
```
[1, 9, 4, 36, 25, 16]
```

Comprehension Syntax -

```
[<expr> for i in <seq> if <cond>]
```

Ex. Create a new list after applying 5% service tax to all the sales values in the given list

In [12]:
```
sales = [500, 300, 900, 800, 1200]
[i*1.05 for i in sales]
```

Out[12]: [525.0, 315.0, 945.0, 840.0, 1260.0]

Examples on Strings

In [13]:
```
strg = "I am in python class"
```

Ex. Print te string in reverse order

In [14]:
```
strg[::-1]
```

Out[14]: 'ssalc nohtyp ni ma I'

Ex. Print following output

```
I Am In Python Class
```

In [16]:
```
strg.title()
```

Out[16]: 'I Am In Python Class'

In [21]:
```
# Example on iterating over words -

for word in strg.split() :
    print(word)
```

```
I
am
in
python
class
```

Ex. WAP to print all the unique vowels in a word entered by user

In [17]:
```
word = input("Enter a word - ").lower()
for ch in word :
    if ch in "aeiou":
        print(ch)
```

```
i
a
o
e
```

In [19]:
```
word = input("Enter a word - ").lower()
for ch in "aeiou" :
    if ch in word:
        print(ch)
```

```
i
```

In [24]:
```
import math
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'a
sin', 'asinh', 'atan', 'atan2', 'atanh', 'cbrt', 'ceil', 'comb', 'copysign', 'cos',
'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'exp2', 'expm1', 'fabs', 'fact
orial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'lo
g1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'r
emainder', 'sin', 'sinh', 'sqrt', 'sumprod', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

In [25]:
```
print(dir(str))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq_
_', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__
getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__l
e__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '_
_str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalp
ha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isp
rintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'mak
etrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'str
ip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

In [32]:
```
help(str.replace)
```

```
Help on method_descriptor:

replace(self, old, new, count=-1, /) unbound builtins.str method
    Return a copy with all occurrences of substring old replaced by new.

      count
        Maximum number of occurrences to replace.
        -1 (the default value) means replace all occurrences.

    If the optional argument count is given, only the first count occurrences are
    replaced.
```

Ex. WAP to replace all the vowels in the word with "*"

In [29]:
```
# Basic approach
word = input("Enter a word - ").lower()
for ch in "aeiou" :
    word = word.replace(ch, "*")
print(word)
```

```
s*ng*p*r*
```

In [31]:
```
# use maketrans() and translate() functions
trans_obj = str.maketrans("aeiou", "*****")
word.translate(trans_obj)
```

Out[31]:
```
's*ng*p*r*'
```

Ex. WAP to convert the profit value to int

In [37]:
```
profit = "($1,200)"
trans_obj = str.maketrans("(", "-", "$,)")
profit = profit.translate(trans_obj)
int(profit)
```

Out[37]:
```
-1200
```

In [34]:
```
help(str.maketrans)
```

```
Help on built-in function maketrans:

maketrans(...)
    Return a translation table usable for str.translate().

    If there is only one argument, it must be a dictionary mapping Unicode
    ordinals (integers) or characters to Unicode ordinals, strings or None.
    Character keys will be then converted to ordinals.
    If there are two arguments, they must be strings of equal length, and
    in the resulting dictionary, each character in x will be mapped to the
    character at the same position in y. If there is a third argument, it
    must be a string, whose characters will be mapped to None in the result.
```

Python Containers

Object	Container Object	Sequence Type	Element Type	Enclosed in	Immutability	Duplicates
str()	No	ordered/indexed	characters	"" or ''	Yes	Yes
tuple()	Yes	ordered/indexed	mixed data (heterogeneous)	()	Yes	Yes
list()	Yes	ordered/indexed	mixed data (heterogeneous)	[]	No	Yes
set()	Yes	unordered	heterogeneous (immutable objects)	{}	No	No
dict()	Yes	unordered	Key - immutable Value - any type	{}	No	Key - No Value - Yes

Ex. Calculate Percentage

Write a program to compute the percentages of 10 students from the provided list. Display the results in a tabular format showing each student's ID and percentage. The student IDs should be generated sequentially starting from 101.

In [42]:
```
st = (51, 67, 83)
sum(st)/3
```

Out[42]: 67.0

In [45]:
```
marks = [(51, 67, 83), (41, 93, 36), (50, 31, 87), (94, 46, 52), (80, 61, 69), (72,

percentage = [round(sum(st)/3, 2) for st in marks]
print(percentage)
```

```
[67.0, 56.67, 56.0, 64.0, 70.0, 63.0, 57.33, 52.33, 67.33, 60.33]
```

`enumerate(<list/tuple>, start = 0)` - returns a sequence object of tuples with first value as counter and second value from the list

In [46]:
```
enumerate(percentage)
```

Out[46]: <enumerate at 0x1f44d702570>

In [48]:
```
print(list(enumerate(percentage)))
```

```
[(0, 67.0), (1, 56.67), (2, 56.0), (3, 64.0), (4, 70.0), (5, 63.0), (6, 57.33), (7,
52.33), (8, 67.33), (9, 60.33)]
```

In [49]:
```
print(list(enumerate(percentage, start = 101)))
```

```
[(101, 67.0), (102, 56.67), (103, 56.0), (104, 64.0), (105, 70.0), (106, 63.0), (10
7, 57.33), (108, 52.33), (109, 67.33), (110, 60.33)]
```

In [52]:
```
for i in enumerate(percentage, start = 101) :
    print(i[0], i[1])
```

```
101 67.0
102 56.67
103 56.0
104 64.0
105 70.0
106 63.0
107 57.33
108 52.33
109 67.33
110 60.33
```

In [58]:
```
# unpacking of tuples

tup = (1, 2, 3)
a, b, c = tup
print(f"a - {a} b - {b} c - {c}")
```

```
a - 1 b - 2 c - 3
```

In [53]:
```
for sid, percent in enumerate(percentage, start = 101) :  # unpacking of tuples
    print(sid, percent)
```

```
101 67.0
102 56.67
103 56.0
104 64.0
105 70.0
106 63.0
107 57.33
108 52.33
109 67.33
110 60.33
```

In [62]:
```
print("-"*20)
print(f"SID \t Percentage")
print("-"*20)
for sid, percent in enumerate(percentage, start = 101) :
    print(f"{sid} \t {percent}")
```

```
--------------------
SID      Percentage
--------------------
101      67.0
102      56.67
103      56.0
104      64.0
105      70.0
106      63.0
107      57.33
108      52.33
109      67.33
110      60.33
```

Note -

- Any sequence can be converted to a list or a tuple
- Any sequence of tuples (size 2) can be converted to a dict

In [63]:
```
list(range(1, 11))
```

Out[63]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [64]:
```
tuple(range(1, 11))
```

Out[64]:
```
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

In [67]:
```
print(list(enumerate(percentage, start = 101)))
```

```
[(101, 67.0), (102, 56.67), (103, 56.0), (104, 64.0), (105, 70.0), (106, 63.0), (10
7, 57.33), (108, 52.33), (109, 67.33), (110, 60.33)]
```

In [68]:
```
print(dict(enumerate(percentage, start = 101)))
```

```
{101: 67.0, 102: 56.67, 103: 56.0, 104: 64.0, 105: 70.0, 106: 63.0, 107: 57.33, 108:
52.33, 109: 67.33, 110: 60.33}
```

Ex. WAP to create a dict by combining following list

In [70]:
```
names = ["Jack", "Jane", "Rosie", "George"]
salary = [40000, 50000, 30000, 60000, 25000]
```

`zip(list1, list2,...)` - returns a sequence object of tuples of size n, where n = number of list, combining the elements index-wise

In [72]:
```
list(zip(names, salary))
```

Out[72]:
```
[('Jack', 40000), ('Jane', 50000), ('Rosie', 30000), ('George', 60000)]
```

In [75]:
```
print(dict(enumerate(zip(names, salary), start = 1)))
```

```
{1: ('Jack', 40000), 2: ('Jane', 50000), 3: ('Rosie', 30000), 4: ('George', 60000)}
```

Ex. Validate user Password

Rules -

- The password cannot match the username.
- The password must be between 8 and 16 characters in length.
- The password must include at least one digit, one uppercase letter, one lowercase letter, and one special character.

In [95]:
```
username = "Jane"
password = "Jane@123"

result = []

result.append(username != password)
result.append(len(password) <=16 and len(password)>= 8)
result.append(set(string.ascii_lowercase) & set(password))
result.append(set(string.ascii_uppercase) & set(password))
result.append(set(string.digits) & set(password))
result.append(set("!@#$%&*") & set(password))


if all(result) :
    print("Valid Password")
else:
    print("Invalid Password")
```

Out[95]:
```
True
```

Note - bool() of empty sequence is always FALSE

- **`all()`** - returns True, if all elements in the list are True
- **`any()`** - returns True if any 1 element in the list is True

In [85]:
```
import string
string.ascii_letters
```

Out[85]:
```
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

In [86]:
```
string.ascii_lowercase
```

Out[86]:
```
'abcdefghijklmnopqrstuvwxyz'
```

In [88]:
```
set(string.ascii_lowercase) & set(password)
```

Out[88]:
```
{'a', 'e', 'n'}
```

Function Arguments

- **Function definition**
  - def keyword
  - demo - name of function
  - name and age are parameters
  - function implementation

In [97]:
```
def demo(name, age) :
    print(f"Name - {name} and Age - {age}")
    return ""
```

- **Function call**
  - "jane" and 30 are arguments

In [98]:
```
demo("jane", 30)
```

```
Name - jane and Age - 30
```

Out[98]:
```
''
```

1. Required Positional Argument

In [99]:
```
demo("jane")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[99], line 1
----> 1 demo("jane")

TypeError: demo() missing 1 required positional argument: 'age'
```

In [100…
```
demo(30, "jane")
```

```
Name - 30 and Age - jane
```

Out[100…
```
''
```

In [102…
```
lst = [10, 20, 30, 40, 50]
lst.insert(2, 15)
```

```
lst
```

Out[102… [10, 20, 15, 30, 40, 50]

In [103…
```
lst = [10, 20, 30, 40, 50]
lst.insert(15, 2)
lst
```

Out[103… [10, 20, 30, 40, 50, 2]

In [104…
```
lst = [10, 20, 30, 40, 50]
lst.insert( 2, "abcd")
lst
```

Out[104… [10, 20, 'abcd', 30, 40, 50]

In [105…
```
lst = [10, 20, 30, 40, 50]
lst.insert("abcd", 2 )
lst
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[105], line 2
      1 lst = [10, 20, 30, 40, 50]
----> 2 lst.insert("abcd", 2 )
      3 lst

TypeError: 'str' object cannot be interpreted as an integer
```

In [106…
```
help(lst.insert)
```

```
Help on built-in function insert:

insert(index, object, /) method of builtins.list instance
    Insert object before index.
```

2. Default Arguments

In [108…
```
strg = "mississippi"
strg.replace("i", "*")
```

Out[108… 'm*ss*ss*pp*'

In [109…
```
strg = "mississippi"
strg.replace("i", "*", 2)
```

Out[109… 'm*ss*ssippi'

In [110…
```
help(strg.replace)
```

```
Help on built-in function replace:

replace(old, new, count=-1, /) method of builtins.str instance
    Return a copy with all occurrences of substring old replaced by new.

      count
        Maximum number of occurrences to replace.
        -1 (the default value) means replace all occurrences.

    If the optional argument count is given, only the first count occurrences are
    replaced.
```

3. Variable Length Argument

In [112…
```python
def demo(name, *args, age = 30) :
    print(f"Name - {name} and Age - {age}")
    print("args - ", args)
```

In [114…
```python
demo("jane", 50, 60, 70, 80, 90, 25)
```

```
Name - jane and Age - 30
args -  (50, 60, 70, 80, 90, 25)
```

4. Key-word Argument

In [115…
```python
demo("jane", 50, 60, 70, 80, 90, age = 25)
```

```
Name - jane and Age - 25
args -  (50, 60, 70, 80, 90)
```

In [116…
```python
def demo(name, age) :
    print(f"Name - {name} and Age - {age}")
demo(age = 25, name = "jane")
```

```
Name - jane and Age - 25
```

5. Variable Length Key-word Argument

In [117…
```python
def demo(name, *args, age = 30, **kwargs) :
    print(f"Name - {name} and Age - {age}")
    print("args - ", args)
    print("kwargs - ", kwargs)
```

In [125…
```python
demo("jane", 50, 60, 70, 80, 90, age = 25, a = (10, 20, 30), b = 20)
```

```
Name - jane and Age - 25
args -  (50, 60, 70, 80, 90)
kwargs -  {'a': (10, 20, 30), 'b': 20}
```

In [121…
```python
def data_to_file(name, gender, age):
    with open("data.txt", "a") as file :
        file.write(f"{name},{gender},{age}\n")
    print(f"Data for {name} is added to file")
```

In [122…
```
data_to_file("Jane", "F", 30)
```

```
Data for Jane is added to file
```

In [126…
```
employee = [("Rosie", "F", 25),
            ("George", "M", 30),
            ("Jack", "M", 35)]

for emp in employee :
    data_to_file(*emp)
```

```
Data for Rosie is added to file
Data for George is added to file
Data for Jack is added to file
```

- `*` - all the arguments after `*` must be `key-word` only arguments
- `/` - all the arguments before `/` must be `positional-only` arguemnts

In [128…
```
def demo(name, age) :
    print(f"Name - {name} and Age - {age}")

demo("Jane", 25)  # positional
demo(age = 25, name = "jane")  # key-word
demo("Jane", age = 25)  # positional and key-word
```

```
Name - Jane and Age - 25
Name - jane and Age - 25
Name - Jane and Age - 25
```

In [129…
```
def demo(name, age, /) :  # name and age are positional-only
    print(f"Name - {name} and Age - {age}")

demo("Jane", 25)  # positional
demo(age = 25, name = "jane")  # key-word   (Error)
demo("Jane", age = 25)  # positional and key-word  (Error)
```

```
Name - Jane and Age - 25
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[129], line 5
      2     print(f"Name - {name} and Age - {age}")
      4 demo("Jane", 25)  # positional
----> 5 demo(age = 25, name = "jane")  # key-word
      6 demo("Jane", age = 25)  # positional and key-word

TypeError: demo() got some positional-only arguments passed as keyword arguments: 'n
ame, age'
```

In [130…
```
def demo(name, /, age) :  # name is positional-only and age is key-word only
    print(f"Name - {name} and Age - {age}")

demo("Jane", 25)  # positional
demo("Jane", age = 25)  # positional and key-word
demo(age = 25, name = "jane")  # key-word   (Error)
```

```
Name - Jane and Age - 25
Name - Jane and Age - 25
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[130], line 6
      4 demo("Jane", 25)  # positional
      5 demo("Jane", age = 25)  # positional and key-word
----> 6 demo(age = 25, name = "jane")  # key-word   (Error)

TypeError: demo() got some positional-only arguments passed as keyword arguments: 'n
ame'
```

In [132…
```
def demo(name, *, age) :  # name is positional-only and age is key-word only
    print(f"Name - {name} and Age - {age}")

# demo("Jane", 25)  # positional (Error)
demo("Jane", age = 25)  # positional and key-word
demo(age = 25, name = "jane")  # key-word
```

```
Name - Jane and Age - 25
Name - jane and Age - 25
```

In [133…
```
def demo(name, /, *, age) :  # name is positional-only and age is key-word only
    print(f"Name - {name} and Age - {age}")

# demo("Jane", 25)  # positional (Error)
demo("Jane", age = 25)  # positional and key-word
demo(age = 25, name = "jane")  # key-word   (Error)
```

```
Name - Jane and Age - 25
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[133], line 6
      4 # demo("Jane", 25)  # positional (Error)
      5 demo("Jane", age = 25)  # positional and key-word
----> 6 demo(age = 25, name = "jane")  # key-word   (Error)

TypeError: demo() got some positional-only arguments passed as keyword arguments: 'n
ame'
```

In [135…
```
help(sorted)
```

```
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

In [136…
```
sorted([1, 3, 2, 4], reverse=True)
```

Out[136…
```
[4, 3, 2, 1]
```

Function Object

In [137…
```
len("Abcd")  # function call
```

Out[137…
```
4
```

In [139…
```
var = len  # function object
type(var)
```

Out[139…
```
builtin_function_or_method
```

In [140…
```
var("Abcd")
```

Out[140…
```
4
```

In [141…
```
lst = ["train", "bike", "flight", "car"]
sorted(lst)
```

Out[141…
```
['bike', 'car', 'flight', 'train']
```

In [142…
```
sorted(lst, key = len)
```

Out[142…
```
['car', 'bike', 'train', 'flight']
```

In [143…
```
sorted(lst, key = lambda strg : strg[-1])
```

Out[143…
```
['bike', 'train', 'car', 'flight']
```

In [144…
```
var = lambda strg : strg[-1]
var
```

Out[144…
```
<function __main__.<lambda>(strg)>
```

In [146…
```
max(lst, key = len)
```

Out[146…
```
'flight'
```

In [147…
```
add = lambda a, b : a + b
add(2, 3)
```

Out[147…
```
5
```

Ex. WAP to sort the percentage dict by the values

In [170…
```
final_dict = dict(sorted(percentages.items(), key = lambda element : element[1], re
print("-"*20)
print(f"SID \t Percentage")
print("-"*20)
for sid, percent in final_dict.items() :
    print(f"{sid} \t {percent}%")
```

```
--------------------
SID      Percentage
--------------------
105      70.0%
109      67.33%
101      67.0%
104      64.0%
106      63.0%
110      60.33%
107      57.33%
102      56.67%
103      56.0%
108      52.33%
```

In [168…

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: