Assignment 15 Solutions

1. What are the new features added in Python 3.8 version?

ANS: Some New Features Added in Python 3.8 Version are:

Walrus Operator: This operator is used to assign and return a value in the same expression. This removes the need for initializing the variable upfront. The major benefit of this is saves some lines of code. It is also known as **The Walrus Operator** due to its similarity to the eyes and tusks of a walrus.

yield and **return** statements do not require parentheses to return multiple values.

Reversed works with a dictionary. The built-in-method **reversed()** can be used for accessing the elements in the reverse order of insertion

Dict comprehensions have been modified so that the key is computed first and the value second.

importlib_metadata is a new library added in the Python's standard utility modules, that provides an API for accessing an installed package's metddata, such as its entry points or its top-level name.

f-strings now support **=** to make string interpolation easy. Python 3.8 allows the use of the above-disussed assignment operator and equal sign **(=)** inside the f-string.

In the three-argument form of **pow()**, when the exponent is -1, it calculates the modular multiplicative inverse of the given value

The **csv.DictReader** now returns instances of dictonary instead of a **collections.OrderedDict**.

If you miss a comma in your code such as a = **[(1, 2) (3, 4)]**, instead of throwing **TypeError** , it displays an informative Syntax warning.

In [3]:
```python
# Example of Walrus Operator
if (sum := 10 + 5) > 10: # its always recommended to use paranthesis with walrus operator
    print(sum) #return 15

# Example of yield and return
def hello():
    return 'Hello','Good Morning'
print(hello())
def count():
    for i in range(5):
        yield i,i**2
for ele in count():
    print(ele, end=" ")
print()

# Example of Reversed Support for dict
t_dict = {"Name":"Hrushi Durg","Role":"Data Scientist"}
for ele in reversed(t_dict):
    print(f'{ele}:"{t_dict[ele]}"')

# Example of using = in F-strings
len_string = len("Ineuron Full Stack Data Science")
print(f'The length of string is {len_string = }')

# Example of Infomrative syntax instead of synatx error while missing comma.
r_list = [(1,2) (3,4)]
```

```
15
('Hello', 'Good Morning')
(0, 0) (1, 1) (2, 4) (3, 9) (4, 16)
Role:"Data Scientist"
Name:"Hrushi Durg"
The length of string is len_string = 31
```

```
<>:26: SyntaxWarning: 'tuple' object is not callable; perhaps you missed a comma?
<>:26: SyntaxWarning: 'tuple' object is not callable; perhaps you missed a comma?
C:\Users\hrush\AppData\Local\Temp/ipykernel_17812/855774841.py:26: SyntaxWarning: 'tuple' object is not callable;
perhaps you missed a comma?
  r_list = [(1,2) (3,4)]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_17812/855774841.py in <module>
     24 
     25 # Example of Infomrative syntax instead of synatx error while missing comma.
---> 26 r_list = [(1,2) (3,4)]

TypeError: 'tuple' object is not callable
```

2. What is monkey patching in Python?

ANS: In Python, the term monkey patch refers to making dynamic (or run-time) modifications to a class or module. In Python, we can actually change the behavior of code at run-time.

In [ ]:
```
class A:
    def func(self):
        print("func() is being called")

def monkey_f(self):
    print("monkey_f() is being called")

A.func = monkey_f
some_object = A()
some_object.func()
```

3. What is the difference between a shallow copy and deep copy?

ANS: The Difference between a Shallow Copy and deep copy are as follows:

When an object is copied using **copy()** , it is called **shallow copy** as changes made in copied object will also make corresponding changes in original object, because both the objects will be referencing same address location.

When an object is copied using **deepcopy()** , it is called **deep copy** as changes made in copied object will not make corresponding changes in original object, because both the objects will not be referencing same address location.

In [ ]:
```
from copy import deepcopy, copy
l_one = [1,2,[3,4],5,6]
l_two = deepcopy(l_one)
l_three = l_one
print(f'Original Elements of each List\n{l_one}\n{l_two}\n{l_three}')
l_two[0] = 10
l_three[-1] = 20
print(f'New Elements of each List\n{l_one}\n{l_two}\n{l_three}')
```

4. What is the maximum possible length of an identifier?

ANS: In Python, the highest possible length of an identifier is 79 characters. Python is a high level programming language. It's also a complex form and a collector of waste.

- Python, particularly when combined with identifiers, is case-sensitive.
- When writng or using identifiers in Python, it has a maximum of 79 characters.
- Unlikely, Python gives the identifiers unlimited length.
- However, the layout of PEP-8 prevents the user from breaking the rules and includes a 79- character limit.

5. What is generator comprehension?

ANS: A generator comprehension is a single-line specification for defining a generator in Python.

- It is absolutely essential to learn this syntax in order to write simple and readable code.
- Generator comprehension uses round bracket unlike square in list comprehension.
- The generator yields one item at a time and generates item only when in demand. Whereas, in a list comprehension, Python reserves memory for the whole list. Thus we can say that the generator expressions are memory efficient than the lists.

In [ ]:
```
in_list = [x for x in range(10)] # List Comprehension
print(in_list)
out_gen = (x for x in in_list if x%2 == 0) # Generator Comprehension
print(out_gen) # Returns a Generator Object
for ele in out_gen:
    print(ele, end=" ")
```