Better programmer tips

1. proper naming should be given to variables

2. Proper spacing

Strings

```
#Double quotations
words = ""
print(type(words))
print(words)
```

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

```
#single quotations
words = ''
print(type(words))
print(words)
```

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

```
#Triple single quotations
words = ''''''
print(type(words))
print(words)
```

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

```
#Slicing
#[Start : End : Step]
#Step is +ve : start from 0 index (Forward direction)
#Step is -ve: start from -1 index (Reverse direction)
words = "Hello world"
print(words[::-2])
```

```
drwolH
```

```
print(words[::2])
```

```
Hlowrd
```

```
print(words[2::])
```

```
llo world
```

```
print(words[-2::])
```

```
ld
```

```
print(words[::-4])
```

```
dwl
```

```
print(words[0:11:2])
```

```
Hlowrd
```

```
#doesn't
print("doesn't")
```

```
doesn't
```

```
#doesn"t
print('doesn"t')
```

```
doesn"t
```

```
print("\\n")
```

```
\n
```

```python
print("\n")
```

```
```

```python
print()
```

```
```

```python
print("\tSai")
```

```
	Sai
```

```python
print("\\tSai")
```

```
\tSai
```

```python
print("\b\b\bSai")
```

```
Sai
```

```python
print("\\b\\b\\bSai")
```

```
\b\b\bSai
```

```python
#Raw String
print("MREC\nDS\nMECH\nMINING")
print(r"MREC\nDS\nMECH\nMINING")
```

```
MREC
DS
MECH
MINING
MREC\nDS\nMECH\nMINING
```

```python
#Concatenation
words = "MREC "
print(2 * words + " DS" + " MECH" + " Mining")
```

```
MREC MREC  DS MECH Mining
```

String inbuilt functions

```
words = "   Hello World  "
print(words)
words = words.strip()
print(words)
```

```
   Hello World  
Hello World
```

Double-click (or enter) to edit

```
print(words.upper())
words = words.upper()
print(words)
```

```
HELLO WORLD
HELLO WORLD
```

```
print(words.lower())
words = words.lower()
print(words)
```

```
hello world
hello world
```

```
words = words.replace("world", "India")
print(words)
```

```
hello India
```

```
words = "sai krishna"
print(words.split(" "))
print(words.split("i"))
print(words.split("s"))
print(words.split("sai"))
```

```
['sai', 'krishna']
['sa', ' kr', 'shna']
['', 'ai kri', 'hna']
['', ' krishna']
```

```
print(len(words))
```

```
11
```

```
print(words.count("i"))
```

```
2
```

```
print(words.index("k"))
```

```
4
```

```
print(words.capitalize())
print(words)
```

```
Sai krishna
sai krishna
```

```
#words = sai krishna
print(words.find("na"))
```

```
9
```

```
print(words.find("krish"))
```

```
4
```

```
#Predict the output
words = "hello"
words[0] = 'i'
print(words)

```

```
#Predict the output
words = "hello"
del words[0]
print(words)

```

Q1. Problem Statement :

You have to write a function that accepts a string of length “length”, the string has some “#”, in it. you have to move all the hashes to the front of the string and return the whole string back and print it.

Example:
Sample Test Case
Input:
Move#Hash#to#Front
Output:
###MoveHashtoFront

```python
#Method - 1
words = input()
hash_count = words.count("#")
print(hash_count)
words = words.replace("#", "")
print(words)
words = hash_count * "#" + words
print(words)
```

```
sai#mrec#ds#mech#mining##
6
saimrecdsmechmining
######saimrecdsmechmining
```

```python
#Method - 2
words = "sai#mrec#ds#mech#mining##"
words_list = words.split("#")
print(words_list)
count = words.count("#")
words = "".join(words_list)
print(words)
words = "#" * count + words
print(words)
```

```
['sai', 'mrec', 'ds', 'mech', 'mining', '', '']
saimrecdsmechmining
######saimrecdsmechmining
```

Tuple

```
#Creating an empty tuple
tuple_items = ()
print(type(tuple_items))
```

```
<class 'tuple'>
```

```
#Creating an empty tuple
tuple_items = tuple()
print(type(tuple_items))
```

```
<class 'tuple'>
```

```
tuple_items = (1)
print(type(tuple_items))
```

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

```
tuple_items = (1,)
print(type(tuple_items))
```

```
<class 'tuple'>
```

```
tuple_items = 1,2,3,4,5
print(type(tuple_items))
```

```
<class 'tuple'>
```

```
tuple_value = ("MREC")
print(tuple_value)
print(type(tuple_value))
```

```
MREC
<class 'str'>
```

```
#Tuple with single value
tuple_value = ("MREC",)
print(tuple_value)
print(type(tuple_value))
```

```
('MREC',)
<class 'tuple'>
```

```
#Predict the output
tuple_value = (26, 45, "Hello")
tuple_value[1] = 7
print(tuple_value)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-90-c1386904690f> in <cell line: 3>()
      1 #Predict the output
      2 tuple_value = (26, 45, "Hello")
----> 3 tuple_value[1] = 7
      4 print(tuple_value)

TypeError: 'tuple' object does not support item assignment
```

```
#Predict the output
tuple_value = (26, 45, "Hello") * 2
print(tuple_value)
```

```
(26, 45, 'Hello', 26, 45, 'Hello')
```

Inbuilt functions of tuple

```
tuple_values = (9, 5, 89, 4, 8)
print(len(tuple_values))
```

```
5
```

```
print(min(tuple_values))
```

```
4
```

```
print(max(tuple_values))
```

```
89
```

```
print(sum(tuple_values))
```

```
115
```

```
print(sorted(tuple_values))
```

```
[4, 5, 8, 9, 89]
```

```
tuple_example = (0,1)
print(any(tuple_example))
```

```
True
```

```
tuple_example = (4,1)
print(all(tuple_example))
```

```
True
```

```
tuple_example = (0,1,7)
print(all(tuple_example))
```

```
False
```

```
tuple_example = (3, 7, 'p', 'y','z',9.5,'y')
print(tuple_example.index('y'))
```

```
3
```

```
tuple_example = (3, 7, 'p', 'y','z',9.5,'y')
print(tuple_example.count('y'))
```

```
2
```

Range

```
tuple(range(5))
```

```
(0, 1, 2, 3, 4)
```

```
tuple(range(0,5))
```

```
(0, 1, 2, 3, 4)
```

```
tuple(range(1, 11, 2))
```

```
(1, 3, 5, 7, 9)
```

```
tuple(range(11))
```

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
#printing tuple values in the same line
tuple_example = (3, 7, 'p', 'y','z',9.5,'y')
for item in tuple_example:
  print(item, end = " ")
```

```
3 7 p y z 9.5 y
```

Dictionary

{key:value}

```
#Creating an empty dictionary
dict_values = {}
print(dict_values)
print(type(dict_values))
```

```
{}
<class 'dict'>
```

```
#Creating an empty dictionary
dict_values = dict()
print(dict_values)
print(type(dict_values))
```

```
{}
<class 'dict'>
```

```
dict_values = {1 : "Focus", 2 : "Academy", 3 : "for"}
print(dict_values)
```

```
{1: 'Focus', 2: 'Academy', 3: 'for'}
```

```python
dict_values = {1 : (2, 4, 5), "Name" : "Face", 4 : [5, 7]}
print(dict_values)
```

```
{1: (2, 4, 5), 'Name': 'Face', 4: [5, 7]}
```

```python
dict_values = dict([(1, 2), ("Name", "Face"), (4, 5)])
print(dict_values)
```

```
{1: 2, 'Name': 'Face', 4: 5}
```

```python
#Nested dictionary value access
dict_values = {"roll_no" : {"Name" : "sai", "branch" : "Ds"}}
print(dict_values["roll_no"]["Name"])
```

```
sai
```

```python
#Adding elements
dict_values = {}
dict_values[0] = "Apple"
dict_values[1] = "Hard"
dict_values[2] = "Work"
print(dict_values)
```

```
{0: 'Apple', 1: 'Hard', 2: 'Work'}
```

```python
dict_values['new_set'] = 1,5,8
print(dict_values)
```

```
{0: 'Apple', 1: 'Hard', 2: 'Work', 'new_set': (1, 5, 8)}
```