Single line comment

In [6]:
```
letter = 'P'                # A string could be a single character or a bunch of texts
print(letter)               # P
print(len(letter))          # 1
```

```
P
1
```

In [8]:
```
greeting = 'Hello, World!'  # String could be  a single or double quote,"Hello, World!"
print(greeting)             # Hello, World!
print(len(greeting))        # 13
```

```
Hello, World!
13
```

In [10]:
```
sentence = "I hope you are enjoying 30 days of python challenge"
print(sentence)
```

```
I hope you are enjoying 30 days of python challenge
```

Multi-Line String

In [13]:
```
multiline_string = '''I am a teacher and enjoy teaching.
I didn't find anything as rewarding as empowering people.
That is why I created 30 days of python.'''
print(multiline_string)
```

```
I am a teacher and enjoy teaching.
I didn't find anything as rewarding as empowering people.
That is why I created 30 days of python.
```

Another way of doing the same thing

In [16]:
```
multiline_string = """I am a teacher and enjoy teaching.
I didn't find anything as rewarding as empowering people.
That is why I created 30 days of python."""
print(multiline_string)
```

```
I am a teacher and enjoy teaching.
I didn't find anything as rewarding as empowering people.
That is why I created 30 days of python.
```

String Concatenation

In [19]:
```
first_name = 'Asabeneh'
last_name = 'Yetayeh'
space = ' '
full_name = first_name  +  space + last_name
print(full_name) # Asabeneh Yetayeh
```

```
Asabeneh Yetayeh
```

Checking length of a string using len() builtin function

In [22]:
```
print(len(first_name))  # 8
print(len(last_name))   # 7
print(len(first_name) > len(last_name)) # True
print(len(full_name)) # 15
```

```
8
7
True
16
```

Unpacking characters

In [35]:
```
language = 'Python'
a,b,c,d,e,f = language # unpacking sequence characters into variables
print(a) # P
print(b) # y
print(c) # t
print(d) # h
print(e) # o
print(f) # n
```

```
P
y
t
h
o
n
```

Accessing characters in strings by index

In [38]:
```
language = 'Python'
first_letter = language[0]
print(first_letter) # P
second_letter = language[1]
print(second_letter) # y
last_index = len(language) - 1
last_letter = language[last_index]
print(last_letter) # n
```

```
P
y
n
```

If we want to start from right end we can use negative indexing. -1 is the last index

In [41]:
```
language = 'Python'
last_letter = language[-1]
print(last_letter) # n
second_last = language[-2]
print(second_last) # o
```

```
n
o
```

Slicing

In [52]:
```
language = 'Python'
first_three = language[0:3] # starts at zero index and up to 3 but not include 3
last_three = language[3:6] # [3:6] means 3:(6-1) as 2nd index means n-1. Since Python indexing starts from 0 so
print(first_three) # Pyt
print(last_three) # hon
```

```
Pyt
hon
```

Another way

In [55]:
```
last_three = language[-3:] #(As backward index starts with -1) i.e -3  -2  -1. -3: means Characters BEFORE, beca
print(last_three)   # hon
last_three = language[3:]  #3: means Characters AFTER Slicing
print(last_three)   # hon
```

```
hon
hon
```

Skipping character while splitting Python strings

In [58]:
```
language = 'Python'
pto = language[0:6:2] #
print(pto) # pto
```

```
Pto
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js