Python String:

In Python, strings are immutable, which means we can not change the string after assignment. Python does not support char data type, so 1 single string is referred to as string with length of 1 char.

Python provides varities of String function.

Note: We can perform slicing on Python string, and String are indexable, In Python index starts from 0 to n - 1

All the functions from Python String:

In [1]:
```
# do not understand this line (focus on output only...)
print("All Functions from String: ",[i for i in dir(str) if "__" not in i])
```

```
All Functions from String:  ['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 [2]:
```
a = "U"
```

In [3]:
```
type(a)
```

Out[3]:
```
str
```

In [4]:
```
len(a)
```

Out[4]:
```
1
```

In [5]:
```
a = ""
```

In [6]:
```
type(a)
```

Out[6]:
```
str
```

In [7]:
```
a
```

Out[7]:
```
''
```

In [8]:
```
a = "upGrad"
```

In [9]:
```
1 a
```

Out[9]:
```
'upGrad'
```

Note:

In Python, Index always starts from 0 and ends with n - 1.

In [10]:
```
1 a[0]
```

Out[10]:
```
'u'
```

In [11]:
```
1 a[1]
```

Out[11]:
```
'p'
```

In [12]:
```
1 a[5]
```

Out[12]:
```
'd'
```

In [13]:
```
1 a[6]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[13], line 1
----> 1 a[6]

IndexError: string index out of range
```

Note:

In Python, Indexing can be positive or negative.

Positive Index	0	1	2	3	4	5	6	7	8	9	10	11	
	K	N	O	W	L	E	D	G	E	H	U	T	
	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	Negative Index

st = "KnowledgeHut"

st[5]	e	st[1:5] nowl	here, starts pos : 1 and ends pos : 5, (5 exclusive, 5 - 1 = 4)
st[0]	K	st[1:10:2]	NWEGH start : 1, end: 10 - 1 = 9, stepsize : 2 (by default stepsize is 1)
st[-1]	T		
st[11]	T		

In [15]:
```
1 st = "KNOWLEDGEHUT"
```

In [16]:
```
1 st[0]
```

Out[16]: 'K'

In [17]:
```
1 st[-1]
```

Out[17]: 'T'

In [18]:
```
1 st[1:5]
```

Out[18]: 'NOWL'

In [19]:
```
1 st[1:10:2]
```

Out[19]: 'NWEGH'

In [20]:
```
1 st
```

Out[20]: 'KNOWLEDGEHUT'

In [21]:
```
1 st[-5:-10]
```

Out[21]: ''

In [22]:
```
1 st
```

Out[22]: 'KNOWLEDGEHUT'

In [23]:
```
1 st[5:0]
```

Out[23]: ''

In [24]:
```
1 st[-5:-1]
```

Out[24]: 'GEHU'

In [25]:
```
1 st[-10:-5]
```

Out[25]: 'OWLED'

In [35]:
```
1 # IMMUATABLE
```

In [36]:
```
1 st
```

Out[36]: 'KNOWLEDGEHUT'

In [37]:
```
1 st[-1]
```

Out[37]:
```
'T'
```

In [38]:
```
1 st[-1] = "A"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[38], line 1
----> 1 st[-1] = "A"

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

In [31]:
```
1 string = "python is very EASY lanaGUAGe"
```

In [32]:
```
1 string
```

Out[32]:
```
'python is very EASY lanaGUAGe'
```

capitalize

In [33]:
```
1 print(string.capitalize.__doc__)
```

```
Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower
case.
```

In [34]:
```
1 string.capitalize()
```

Out[34]:
```
'Python is very easy lanaguage'
```

Python Casefold function

Using this function we can match our string as caseless.

In [39]:
```
1 "A" == "a"
```

Out[39]:
```
False
```

In [40]:
```
1 st
```

Out[40]:
```
'KNOWLEDGEHUT'
```

In [41]:
```
st1 = "knowledgeHut"
```

In [42]:
```
st.casefold() == st1.casefold()
```

Out[42]: True

String Find or Index Function

In [43]:
```
quote = "Make it work, make it right, make it fast, make it now"
```

In [44]:
```
quote
```

Out[44]: 'Make it work, make it right, make it fast, make it now'

Find

Using the find function we can find the index of any item in give string.

Find function returns -1 in case of failure, which mean if any item is not the part of it then find function returns - 1.

By default this function returns the lowest index of an item.

In [45]:
```
quote.find("work")
```

Out[45]: 8

In [46]:
```
quote.find("Make")
```

Out[46]: 0

In [47]:
```
quote.find("make")
```

Out[47]: 14

In [48]:
```
quote
```

Out[48]: 'Make it work, make it right, make it fast, make it now'

In [49]:
```
quote.find("Modi")
```

Out[49]: -1

In [50]:
```
quote.find("it")
```

Out[50]: 5

In [51]:
```
1 quote
```

Out[51]: 'Make it work, make it right, make it fast, make it now'

In [53]:
```
1 quote.find("it", 5 + 1)
```

Out[53]: 19

In [55]:
```
1 quote.find("it", quote.find("it") + 1)
```

Out[55]: 19

In [56]:
```
1 quote.find("it", quote.find("it", quote.find("it") + 1) + 1)
```

Out[56]: 34

In [57]:
```
1 quote.find("it", quote.find("it", quote.find("it", quote.find("it") + 1) + 1) + 1
```

Out[57]: 48

Index

Find and Index both are same, where Find returns - 1 in case of failure, where as Index function raise an error when it is failure.

In [58]:
```
1 quote.find("Modi")
```

Out[58]: -1

In [59]:
```
1 quote.index("Modi")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[59], line 1
----> 1 quote.index("Modi")

ValueError: substring not found
```

In [60]:
```
1 quote.index("it", quote.index("it", quote.index("it") + 1) + 1)
```

Out[60]: 34

In [61]:
```
1 quote
```

Out[61]: 'Make it work, make it right, make it fast, make it now'

In [62]:
```
1 quote.index("it")
```

Out[62]: 5

rfind and rindex

Search the data from the right side but counting the index from the left side

In [63]:
```
1 quote.rfind("it")
```

Out[63]: 48

In [64]:
```
1 quote.rindex('it')
```

Out[64]: 48

In [65]:
```
1 quote.rfind("Modi")
```

Out[65]: -1

In [66]:
```
1 quote.rindex("Modi")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[66], line 1
----> 1 quote.rindex("Modi")

ValueError: substring not found
```

very good query by Mr. Karan, but please explore by yourself.

- Split
- rSplit
- Partition
- rPartition

In [67]:
```
1 print("We are learning Python with Abhisheak!!!")
2
3 print("We are learning Python with Abhisheak!!!")
4
5 print("We are learning Python with Abhisheak!!!")
```

```
We are learning Python with Abhisheak!!!
We are learning Python with Abhisheak!!!
We are learning Python with Abhisheak!!!
```

In [68]:
```
1 bucket = "Veg,Bread,Egg,Chocolate,Fruit,Paneer,Soya,Milk"
```

In [69]:
```
1 bucket
```

Out[69]: 'Veg,Bread,Egg,Chocolate,Fruit,Paneer,Soya,Milk'

Split Function

Using this function we can split the data based on any specified char and it wil return an **output in the form of list data structure,** if you are not passing anything to split the data then this function **by default try to split the data based on white space**.

Note: Split function split the data from left to right.

In [70]:
```
1 name = "Ankit Kumar Sharma"
```

In [71]:
```
1 name.split()
```

Out[71]: ['Ankit', 'Kumar', 'Sharma']

In [86]:
```
1 name.split("a")
2
3 # Ankit Kum, r Sh,rm,''
4
```

Out[86]: ['Ankit Kum', 'r Sh', 'rm', '']

In [73]:
```
1 bucket
```

Out[73]: 'Veg,Bread,Egg,Chocolate,Fruit,Paneer,Soya,Milk'

In [74]:
```
1 bucket.split(",")
```

Out[74]: ['Veg', 'Bread', 'Egg', 'Chocolate', 'Fruit', 'Paneer', 'Soya', 'Milk']

In [75]:
```
1 bucket.split(",",3)
```

Out[75]: ['Veg', 'Bread', 'Egg', 'Chocolate,Fruit,Paneer,Soya,Milk']

In [76]:
```
1 bucket.split(",",1)
```

Out[76]: ['Veg', 'Bread,Egg,Chocolate,Fruit,Paneer,Soya,Milk']

In [77]:
```
1 bucket.split(",",5)
```

Out[77]: ['Veg', 'Bread', 'Egg', 'Chocolate', 'Fruit', 'Paneer,Soya,Milk']

In [78]:
```
1 name
```

Out[78]: 'Ankit Kumar Sharma'

In [80]:
```
1 name.split(" ",1)
```

Out[80]: ['Ankit', 'Kumar Sharma']

In [81]:
```
1 email = "myprogrammingisfun@gmail.com"
```

In [82]:
```
1 email.split("@")
```

Out[82]: ['myprogrammingisfun', 'gmail.com']

In [83]:
```
1 print("User Name: ",email.split("@")[0])
```

```
User Name:  myprogrammingisfun
```

In [85]:
```
1 print("Domain Name: ",email.split("@")[-1])
```

```
Domain Name:  gmail.com
```

In [87]:
```
1 lst = ["A","B","C","D"]
```

In [88]:
```
1 lst[0]
```

Out[88]: 'A'

In [89]:
```
1 lst[-1]
```

Out[89]: 'D'

In [90]:
```
1 lst[-4]
```

Out[90]: 'A'

In [91]:
```
1 lst[-1]
```

Out[91]: 'D'

In [94]:
```
1 email = "sugandha.arora@xyz.com"
```

In [96]:
```
1 email.split("@")[0]
```

Out[96]: 'sugandha.arora'

In [97]:
```
email.split("@")[-2]
```

Out[97]: 'sugandha.arora'

In [98]:
```
email.split("@")[1]
```

Out[98]: 'xyz.com'

In [99]:
```
email.split("@")[-1]
```

Out[99]: 'xyz.com'

In [101]:
```
email.split("@")[-2].split(".")
```

Out[101]: ['sugandha', 'arora']

rsplit

In [102]:
```
bucket
```

Out[102]: 'Veg,Bread,Egg,Chocolate,Fruit,Paneer,Soya,Milk'

In [103]:
```
name
```

Out[103]: 'Ankit Kumar Sharma'

In [104]:
```
name.rsplit()
```

Out[104]: ['Ankit', 'Kumar', 'Sharma']

In [105]:
```
name.split()
```

Out[105]: ['Ankit', 'Kumar', 'Sharma']

In [106]:
```
name.split(" ",1)
```

Out[106]: ['Ankit', 'Kumar Sharma']

In [107]:
```
name.rsplit(" ",1)
```

Out[107]: ['Ankit Kumar', 'Sharma']

In [108]:
```
bucket
```

Out[108]: 'Veg,Bread,Egg,Chocolate,Fruit,Paneer,Soya,Milk'

In [109]:
```
1 bucket.rsplit(",",3)
```

Out[109]:
```
['Veg,Bread,Egg,Chocolate,Fruit', 'Paneer', 'Soya', 'Milk']
```

Partition and rPartition

Using Partition function we can partition the data based on specified char, this function always return an outputs in the form of Tuple Data Strucutre (that is represented by ()), and that tuple at least or at most contain only 3 item.

Note: If any specified char is not the part of your data then this function also return an output with 3 items in the form of tuple.

Note: Partition the data based on first occurence.

In [112]:
```
1 print(name.partition.__doc__)
```

```
Partition the string into three parts using the given separator.

This will search for the separator in the string.  If the separator is found,
returns a 3-tuple containing the part before the separator, the separator
itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string
and two empty strings.
```

In [110]:
```
1 name
```

Out[110]:
```
'Ankit Kumar Sharma'
```

In [111]:
```
1 name.partition(" ")
```

Out[111]:
```
('Ankit', ' ', 'Kumar Sharma')
```

In [113]:
```
1 ["A","B","C"] # list -> long bracket
```

Out[113]:
```
['A', 'B', 'C']
```

In [116]:
```
1 ("A","B","C") # tuple -> Parenthesis
```

Out[116]:
```
('A', 'B', 'C')
```

In [117]:
```
1 name
```

Out[117]:
```
'Ankit Kumar Sharma'
```

In [118]:
```
1 name.partition("|")
```

Out[118]:
```
('Ankit Kumar Sharma', '', '')
```

In [119]:
```
1 bucket
```

Out[119]: 'Veg,Bread,Egg,Chocolate,Fruit,Paneer,Soya,Milk'

In [120]:
```
1 bucket.partition(",")
```

Out[120]: ('Veg', ',', 'Bread,Egg,Chocolate,Fruit,Paneer,Soya,Milk')

In [122]:
```
1 name
```

Out[122]: 'Ankit Kumar Sharma'

In [123]:
```
1 name.partition("a")
```

Out[123]: ('Ankit Kum', 'a', 'r Sharma')

In [124]:
```
1 name.rpartition("a")
```

Out[124]: ('Ankit Kumar Sharm', 'a', '')

In [125]:
```
1 name = "Sugandha-arora"
```

In [126]:
```
1 name.rpartition("--")
```

Out[126]: ('Sugandha', '-', 'arora')

In [127]:
```
1 name.rpartition("n")
```

Out[127]: ('Suga', 'n', 'dha-arora')

In [128]:
```
1 a = 5
```

In [129]:
```
1 5 * "*"
```

Out[129]: '*****'

In [ ]:
```
1 email = "kual.kumar12@gmail.com"
```

In [ ]:
```
1 k**********2@g****.com
```

Home Task

In [130]:
```
# email = "abcd@gmail.com"
# a**d@g****.com

# abhishek.s@yahoo.co.in

# a********s@y*******.in

# First char and last char from your username and in between all char should be *
# from domain name, I want first char and everything after last dot, in between a
```

In [133]:
```
email = input("Enter your email: ")
# here your logic
print(email)
```

```
Enter your email: abc@gmail.com
abc@gmail.com
```

In [132]:
```
a = "abhi"
b = "sheK"
a + b
```

Out[132]: 'abhisheK'

In [134]:
```
name
```

Out[134]: 'Sugandha-arora'

lower, upper, swapcase

In [135]:
```
name.lower()
```

Out[135]: 'sugandha-arora'

In [136]:
```
name.upper()
```

Out[136]: 'SUGANDHA-ARORA'

In [137]:
```
name.swapcase()
```

Out[137]: 'sUGANDHA-ARORA'

Join

Using this function we can join string, list, tuple etc

In [138]:
```
1 name
```

Out[138]: 'Sugandha-arora'

In [139]:
```
1 "|".join(name)
```

Out[139]: 'S|u|g|a|n|d|h|a|-|a|r|o|r|a'

In [140]:
```
1 lst
```

Out[140]: ['A', 'B', 'C', 'D']

In [141]:
```
1 "&".join(lst)
```

Out[141]: 'A&B&C&D'

strip, lstrip, rstrip

Using these we can remove specified chars from left side or right side, or both side.

- strip: removing leading and trailing char from both side
- rstrip: from right side
- lstrip: from left side

By default based on space it will remove.

In [142]:
```
1 name = "  Abhisheak Saraswat   ***"
```

In [143]:
```
1 name
```

Out[143]: ' Abhisheak Saraswat   ***'

In [144]:
```
1 name.lstrip()
```

Out[144]: 'Abhisheak Saraswat   ***'

In [145]:
```
1 name.lstrip(" A")
```

Out[145]: 'bhisheak Saraswat   ***'

In [146]:
```
1 name.rstrip("* ")
```

Out[146]: ' Abhisheak Saraswat'

In [147]:
```
1 name.rstrip("* ").lstrip()
```

Out[147]: 'Abhisheak Saraswat'

In [148]:
```
1 name = "******Abhishek Saraswat****"
```

In [149]:
```
1 name.lstrip("*")
```

Out[149]: 'Abhishek Saraswat****'

In [150]:
```
1 name.rstrip("*")
```

Out[150]: '******Abhishek Saraswat'

In [151]:
```
1 name.strip("*")
```

Out[151]: 'Abhishek Saraswat'

Startswith and EndsWith

In [152]:
```
1 quote
```

Out[152]: 'Make it work, make it right, make it fast, make it now'

In [153]:
```
1 quote.startswith("Make")
```

Out[153]: True

In [154]:
```
1 quote.endswith("now")
```

Out[154]: True

In [155]:
```
1 quote.endswith("Abhishek")
```

Out[155]: False

In [156]:
```
1 quote.endswith("Modi")
```

Out[156]: False

In [157]:
```
1 quote.endswith("w")
```

Out[157]: True

In [158]:
```
1 name = "abhishe2342@#$"
```

In [159]:
```
1 name.isdigit()
```

Out[159]: False

In [160]:
```
1 name.isnumeric()
```

Out[160]: False

In [161]:
```
1 name.isalnum()
```

Out[161]: False

In [162]:
```
1 name = "abhishe234"
```

isalnum:

if your string contains either alpha or number then this function return an output as True

In [163]:
```
1 name.isalnum()
```

Out[163]: True

In [164]:
```
1 name = "Abhishek"
```

In [165]:
```
1 name.isalnum()
```

Out[165]: True

In [166]:
```
1 name = "02849243"
```

In [167]:
```
1 name.isnumeric()
```

Out[167]: True

In [168]:
```
1 name = "Abhishek"
```

In [169]:
```
1 name.isalpha()
```

Out[169]: True

In [174]:
```
1 name = "abhishek123"
2 name.isalpha()
```

Out[174]: False

In [171]:
```
1 name.isalnum()
```

Out[171]: True

In [175]:
```
name = "23589adsjfsa#$"
```

In [176]:
```
name.isalnum()
```

Out[176]: False

In [172]:
```
name = "12.45"
```

In [177]:
```
name.isdecimal?
```

In [178]:
```
name = 'H.12'
```

In [179]:
```
name.isdecimal()
```

Out[179]: False

In [180]:
```
number = "100.00"
```

In [181]:
```
number.isdecimal()
```

Out[181]: False

In [182]:
```
"100".isdecimal()
```

Out[182]: True

In [185]:
```
number = "100."
```

In [186]:
```
number.isdecimal()
```

Out[186]: False

In [187]:
```
a = "100"
```

In [188]:
```
a.isnumeric()
```

Out[188]: True

In [ ]:
```

```