Introduction to Strings

- Strings are used in Python to record text information, such as names. It could either be a word, a phrase, a sentence, a paragraph or an entire encyclopedia. Strings in Python are actually a *sequence*, which basically means Python keeps track of every element in the string as a sequence. For example, Python understands the string "joker' to be a sequence of letters in a specific order. This means we will be able to use indexing to grab particular letters (like the first letter, or the last letter).

- This idea of a sequence is an important one in Python and we will touch upon it later on in the future.

Creating a String

- To create a string in Python you need to use either single quotes or double quotes. For example:

```
# Single word
my_first_string= 'algebra'
my_first_string
```

```
'algebra'
```

```
# Entire phrase
phrase = 'Statistics sits at the heart of machine learning'
print(phrase)
```

```
Statistics sits at the heart of machine learning
```

```
# Statement to get the type of the variable
type(phrase)
```

```
str
```

```
# We can also use double quote
my_string = "String built with double quotes"
print(my_string)    # Use the print command
```

```
String built with double quotes
```

```
# Be careful with quotes!
sentence= 'I'm using single quotes, but this will create an error'
print(sentence)
```

→▾   I'm using single quotes, but this will create an error

- The reason for the error above is because the single quote in `I'm` stopped the string. You can use combinations of double and single quotes to get the complete statement.

```
sentence= "I'm using single quotes, but this will create an error"
print(sentence)
```

→▾   I'm using single quotes, but this will create an error

```
hashtag = "#"
print(hashtag)
```

→▾   #

```
type(hashtag)
```

→▾   str

⌄   How to print strings

- Using Jupyter notebook with just a string in a cell will automatically output strings, but the correct way to display strings in your output is by using a print function.

```
# We can simply declare a string
'Deep Learning'
```

→▾   'Deep Learning'

```
# Note that we can't output multiple strings this way
'Linear Algebra'
'Calculus'
```

→▾   'Calculus'

⌄   We can use the `print()` statement to print a string.

```
# print('Linear Algebra')
# print('Calculus')
```

```
print('Use \n to print a new line')
print('\n')
print('See what \n I mean?')
```

⇥▾ Use
      to print a new line

      See what
       I mean?

⌄ Playing with strings

- We can also use a function called `len()` to check the length of a string!

```
algo = 'regres sion '
len(algo)
```

⇥▾ 12

Python's built-in len() function counts all of the characters in the string, including spaces and punctuation.

⌄ String Indexing

- We know strings are a sequence, which means Python can use indexes to call parts of the sequence.

- A string index refers to the location of an element present in a string.

- The indexing begins from 0 in Python.

- The first element is assigned an index 0, the second element is assigned an index of 1 and so on and so forth.

- In Python, we use brackets `[]` after an object to call its index.

```
# Assign string as a string
string  = 'Principal Component Analysis!'


# Print the object
print(string)
```

⇥▾ Principal Component Analysis!

- Let's start indexing!

```python
# Show first element (in this case a letter)
print(string[5])
```

```
i
```

```python
print(string[15])
```

```
n
```

```python
len(string)
```

```
29
```

```python
# Grab the element at the index -1, which is the LAST element
print(string[28])
```

```
!
```

```python
print(string[-2])
```

```
s
```

⌄ String Slicing

- We can use a `:` to perform *slicing* which grabs everything up to a designated point.
- The starting index is specified on the left of the `:` and the ending index is specified on the right of the `:`.
- Remember the element located at the right index is not included.

```python
# Grab everything past the first term all the way to the length of s which is len(s)
print(string)
print(string[:13])
```

```
Principal Component Analysis!
Principal Com
```

```python
string[13]
```

```
'p'
```

```
string[12]
```

➔▾   'm'

```
# Grab everything starting from index 10 till index 18
print(string[10:])
```

➔▾   Component Analysis!

- If you do not specify the ending index, then all elements are extracted which comes after the starting index including the element at that starting index. The operation knows only to stop when it has run through the entire string.

```
print(string[3:5])
```

➔▾   nc

- If you do not specify the starting index, then all elements are extracted which comes befores the ending index excluding the element at the specified ending index. The operation knows only to stop when it has extracted all elements before the element at the ending index.

```
print(string[2:4])
```

➔▾   in

- If you do not specify the starting and the ending index, it will extract all elements of the string.

```
#Everything
print(string[:])
print(string)
```

➔▾   Principal Component Analysis!
     Principal Component Analysis!

- We can also use negative indexing to go backwards.

```
# Last letter (one index behind 0 so it loops back around)
string[-29]
```

➔▾   'P'

- We can also extract the last four elements. Remember we can use the index -4 to extract the FOURTH LAST element

```
string[-1:-4]
```

➡️ ''

```
string[-4:]
```

➡️ 'sis!'

- We can also use index and slice notation to grab elements of a sequence by a specified step size (the default is 1). For instance we can use two colons in a row and then a number specifying the frequency to grab elements. For example:

```
# Grab everything, but go in steps size of 1
print(string[::1])
```

➡️ Principal Component Analysis!

```
print(string[::3])
```

➡️ Pnp mntnys

```
# Grab everything, but go in step sizes of 5
print(string[5:15:5])
```

➡️ iC

```
string[::1]
```

➡️ 'Principal Component Analysis!'

```
# We can use this to print a string backwards
string[::-1]
```

➡️ '!sisylanA tnenopmoC lapicnirP'

```
# We can use this to print a string backwards with steps
string[2:4:-1]
```

➡️ ''

```
string[4:2:-1]
```

```
'cn'
```

```
s = 'foobar'
s[0::-3]
```

```
'f'
```

∨ String Properties

- It's important to note that strings have an important property known as *immutability*.

- This means that once a string is created, the elements within it can not be changed or replaced via item assignment. We will see how we can do such operation using string methods

```
# Can we change our string 'Hello' to 'Cello'? Lets try replacing the first letter H with C
string='Hello'
string[0] = 'C'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-76-f55372fc219f> in <module>()
      1 # Can we change our string 'Hello' to 'Cello'? Lets try replacing the first
letter H with C
      2 string='Hello'
----> 3 string[0] = 'C'

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

- Notice how the error tells us directly what we can't do, that is we can't change the item assignment!

- Something we *can* do is concatenate strings!

```
# Concatenate strings!

string1='abc'
string2='def'
print(string1 + ' ' + string2 )
```

```
abc def
```

```
print(string1 + 4 + string2)
```

```
-------------------------------------------------------------------------
TypeError                                   Traceback (most recent call last)
<ipython-input-79-34b0f8d1c2ec> in <module>()
----> 1 print(string1 + 4 + string2 )

TypeError: can only concatenate str (not "int") to str
```

- To convert an integer into a string, you can use the `str()` function or you can simply write the number in quotes

```
# Concatenate strings!

string1='abc'
string2='def'
num = 4
print(string1 + str(4) + string2)
```

```
abc4def
```

```
str(num)
```

```
'4'
```

```
# Concatenate strings!
string1='abc'
string2='def'
string1 + '4'+ string2
```

```
'abc4def'
```

```
print(string)
```

```
Hello
```

```
# We can reassign string completely though!
string = string + ' concatenate me!'
print(string)
```

```
Hello concatenate me! concatenate me!
```

```
letters = 'wubba'
letters*3
```

```
'wubbawubbawubba'
```

∨ String functions and methods

```
algorithm = 'Neural Networks'

print(algorithm)
```

⇥ Neural Networks

∨ len()

- `len()` function returns the length of the string

```
# Print the length of the string
len(algorithm)
```

⇥ 15

∨ lower()

- `lower()` method converts the string to lowercase

```
# Conver the string to lowercase
algorithm.lower()
```

⇥ 'neural networks'

```
print(algorithm)
```

⇥ Neural Networks

- What about `lower(algorithm)`? Is it not similar to the `len()` function?
- An important point to note here is `len()` is a function and `lower()` is a method for a string object. It will be pretty clear about the exact differences between a function and a method as we move ahead.

```
# Lets try that out
lower(algorithm)
```

```
    ----------------------------------------------------------------------
    NameError                                Traceback (most recent call last)
    <ipython-input-90-4d609230893c> in <module>()
          1 # Lets try that out
    ----> 2 lower(algorithm)

    NameError: name 'lower' is not defined
```

upper()

- `upper()` method converts the string to uppercase

```
# Convert the string to uppercase
algorithm.upper()
```

```
'NEURAL NETWORKS'
```

```
print(algorithm)
```

```
Neural Networks
```

count()

- `count()` method returns the count of a string in the given string. Unlike `lower()` and `upper()` method, the `count()` method takes a string as an argument

```
print(algorithm)
```

```
Neural Networks
```

```
algorithm.count('Ne')
```

```
2
```

```
algorithm.count('eu')
```

```
1
```

```
algorithm.count(' ')
```

```
1
```

```
algorithm.count('Neural')
```

→ 1

```
algorithm.count('Neurla')
```

→ 0

⌄ find()

- `find()` method returns the index of the first occurrence of a string present in a given string. Similar to the `count()` method, the `find()` method takes a string as an argument

```
print(algorithm)
```

→ Neural Networks

```
algorithm.find('e')
```

→ 1

```
algorithm.find('Neural')
```

→ 0

```
algorithm.find('Box')
```

→ -1

- <u>An important point to note here is that if the string which you are looking for, is not contained in the original string, then the find method will return a value of -1</u>

⌄ replace()

- `replace()` method takes two arguments - (i) the string to replace and (ii) the string to replace with, and returns a modified string after the operation

```
print(algorithm)
```

→ Neural Networks

```
algorithm.replace(' ','-')
```

```
'Neural-Networks'
```

```
algorithm.replace('N','L')
```

```
'Leural Letworks'
```

```
print(algorithm)
```

```
Neural Networks
```

- Another important point worth noting here is applying any method on a string does not actually change the original string. For example, when you print out the algorithm string, it still contains the original value 'Neural Networks'
- We need to store the modified string in another variable

```
# Storing the modified string
algorithm_revised  = algorithm.replace('Neural','Artificial Neural')
print(algorithm_revised)
print(algorithm)
```

```
Artificial Neural Networks
Neural Networks
```

˅   Printing strings a bit differently

```
first_name = 'Rahul'
last_name = 'Modi'

full_name = f'Left plus right makes {first_name}  {last_name}'  # Use {} to print the variab
print(full_name)
```

```
Left plus right makes Rahul  Modi
```

```
print(first_name + ' '+last_name)
```

```
Rahul Modi
```

Start coding or generate with AI.

```
first_name = 'Vikash'
middle_name = ''
last_name = 'Srivastava'
```

```
full_name = f'I am none other than {first_name} {middle_name}{last_name}. I am a Data Scient
print(full_name)
```

→▾ I am none other than Vikash Srivastava. I am a Data Scientist

```
print(f'I am none other than {first_name} {middle_name}{last_name}. I am a Data Scientist')
```

→▾ I am none other than Vikash Srivastava. I am a Data Scientist

⌄ Check if a string contains a particular word or character

```
my_string = 'Albert Einstein'
```

```
'Albert' in my_string
```

→▾ True

```
'Alberta' in my_string
```

→▾ False

Start coding or generate with AI.