PYTHON FOR DATA SCIENCE

UNIT 08

STRINGS

“STRINGS” ARE CHARACTER DATA

```
string_var = 'Character Data!'
```

String_var is a variable that contains the string ‘Character_Data!’

The characters ‘Character Data!’ are a string.

TECHNICALLY SPEAKING, STRINGS ARE SEQUENCES OF CHARACTERS.

string_var:

C	h	a	r	a	c	t	e	r		D	a	t	a	!

The string ‘Character_Data!’ is actually a sequence of 15 characters that are stored in string_var.

FYI

Strings in Python 3 can use characters from almost any language.

Unicode, formally the **Unicode Standard**, is an information technology standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems. The standard, which is maintained by the Unicode Consortium, defines 144,762 characters covering 159 modern and historic scripts, as well as symbols, emoji, and non-visual control and formatting codes.

Source: https://en.wikipedia.org/wiki/Unicode

CREATING STRINGS

Strings are created with quotes …

… both single and double quotes will work

Single quotes work

```
single_quote = 'This is a valid string.'
print(single_quote)
```

```
This is a valid string.
```

Double quotes work

```
double_quote = "This is also a valid string."
print(double_quote)
```

```
This is also a valid string.
```

- you can use quotes within quotes
- if you want to use single quotes inside a string, then enclose the string inside of double quotes and vice-versa

```
double_inside_single = 'The quote "when life gives you lemons, make lemonade” is credited to Elbert Hubbard.'
print(double_inside_single)
```

```
The quote "when life gives you lemons, make lemonade” is credited to Elbert Hubbard.
```

```
single_inside_double = "It's pure joy coding in Python."
print(single_inside_double)
```

```
It's pure joy coding in Python.
```

SIMPLE OPERATIONS ON STRINGS

- Python has several built-in functions that operate on strings

  Note: not all of these are specific to strings

Basic String Functions

Built-In Function	What the Function Does
len( )	returns the number of characters in the string
print( )	prints the string
str( )	converts other data types to strings

Examples of Basic String Functions

```
print("print() strips out the quotes that enclose the string")
```

```
print() strips out the quotes that enclose the string
```

```
len('len() counts the number of characters in a string')
```

```
49
```

```
# str() converts other data types to strings

num = 35
print(num)
print(type(num))

string_num = str(num)
print(string_num)
print(type(string_num))
```

```
35
<class 'int'>
35
<class 'str'>
```

STRING CONCATENATION

YOU CAN COMBINE TWO STRINGS USING THE + OPERATOR

```
# First string concatenation example

string_1 = "Some LAUNCH classes"
string_2 = " are held at Concurrency."
first_new_string = string_1 + string_2
print(first_new_string)
```

```
Some LAUNCH classes are held at Concurrency.
```

```
# Second string concatenation example

string_1 = "Some LAUNCH classes "
string_2 = "are held at Concurrency."
second_new_string = string_1 + string_2
print(second_new_string)
```

```
Some LAUNCH classes are held at Concurrency.
```

What is the difference between these two examples and what does that mean when you concatenate two strings?

STRING INDEXING

(WORKING WITH PARTS OF STRINGS)

- An index is a way to give a numeric “address” to an element in a sequence (string)
- The characters of a string can be accessed by a numeric index
- Python uses a 0-based index … this means that the index is the offset from the first character
- We can also “slice” strings using indexes … this allows us to get a substring instead of just a single character

Remember: Strings are sequences of characters

string_var = ‘Character_Data!’

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	h	a	r	a	c	t	e	r		D	a	t	a	!

The characters in ‘Character Data!’ are actually elements of a sequence (the string).

EACH CHARACTER HAS AN INDEX (i.e., A POSITION IN THE SEQUENCE)

Python indexes start at zero

The numeric position of each character is called the “index”

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	h	a	r	a	c	t	e	r		D	a	t	a	!

CHARACTERS FROM STRINGS CAN BE RETRIEVED BY USING BRACKETS [ ]

```
string_var = 'Character Data!'
string_var[3]
```

```
'r'
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	h	a	r	a	c	t	e	r		D	a	t	a	!

The code string_var[3] will retrieve the character r.

WHY START AT ZERO?

- An index represents an *offset* from the first character

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	h	a	r	a	c	t	e	r		D	a	t	a	!

The code string_var[3] will retrieve the character r.

So, string_var[3] returned the character 3 positions to the right of the first character.

YOU CAN ALSO USE NEGATIVE INDEXES

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	h	a	r	a	c	t	e	r		D	a	t	a	!
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- The last character of a string can be retrieved with the index -1

```
string_var = 'Character Data!'
string_var[-1]
```

```
'!'
```

```
string_var = 'Character Data!'
string_var[-10]
```

```
'c'
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	h	a	r	a	c	t	e	r		D	a	t	a	!
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

STRING SLICING … CREATING SUBSTRINGS

- Python does not have a substring function
- Instead with Python we “slice” strings to create substrings
- You can use bracket notation to slice a string …
  e.g., string_var[2 : 6]

Slice Operator [Start (do include) : Stop (do not include)]

Retrieving Substrings Using “Bracket” Notation

Syntax: your_string[start-index : stop-index]

The numeric index of the first character of the substring

The numeric index of the character where the substring stops … the character in this position is not included

Examples of substrings

```
string_var = 'Character Data!'
string_var[2:6]
```

```
'arac'
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	h	a	r	a	c	t	e	r		D	a	t	a	!
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

If you remove the stop index, the substring will go to the end of the string

```
string_var = 'Character Data!'
string_var[6:]
```

```
'ter Data!'
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	h	a	r	a	c	t	e	r		D	a	t	a	!
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

If you remove the start index, the substring will start at the beginning (i.e., start at 0)

```
string_var = 'Character Data!'
string_var[:4]
```

```
'Char'
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	h	a	r	a	c	t	e	r		D	a	t	a	!
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Some Additional Points On Slicing

- Slicing is very important!!! Learn it well.

- Slicing can get more complicated … we are keeping things relatively simple for now

- Slicing is used in more complicated data structures

  -- Lists
  -- Arrays
  -- DataFrames
  -- NumPy arrays

STRING METHODS

WHAT ARE METHODS?

Python Functions

- A function is a block of organized, reusable code that is used to perform a single, related action.

- Functions provide better modularity for your application and a high degree of code reusing.

Python Methods

- A method is a function which belongs to an object.

- Strings are also objects.
- Each string instance has its own attributes and methods.
- The most important attribute of the string is the collection of characters.
- There are a wide variety of methods for string objects.

FUNCTION VS. METHOD

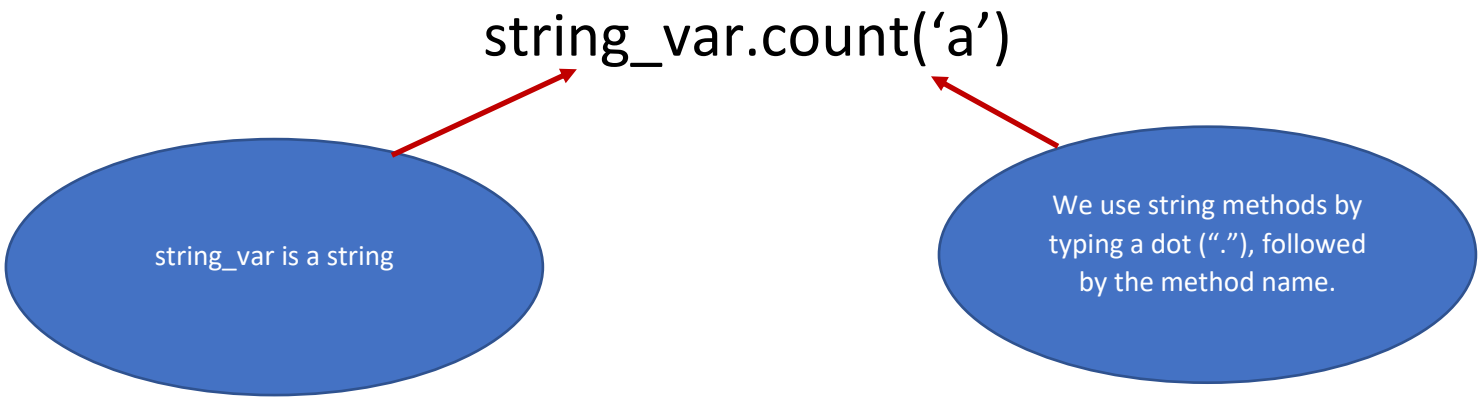
- Methods are associated with classes, but functions are not
- Therefore, methods are specific to classes (and strings are a class)

USING STRING METHODS BY USING “DOT NOTATION”

```
# String method count with argument 'a' ... counts the number of a's in the string

string_var = 'Character Data!'
string_var.count('a')
```

```
4
```



count() is a string method that counts the occurrences of a given character

MOST COMMONLY USED STRING METHODS

(There are many other useful string methods)

Method	What it does
`lower()`	convert characters to lower case
`upper()`	convert characters to upper case
`count()`	count number of occurrences of a sequence of characters
`find()`	find the offset of first occurrence of a sequence of characters
`replace()`	replace a sequence of characters with a new sequence of characters

There will be more to come on Day 2 of Strings!