WELLCOME

STRINGS PYTHON

EMBARKING ON A JOURNEY INTO DATA SCIENCE

YA MANON

You can have data without information but you cannot have information without data.

-Daniel Keys Maran

STRINGS

In this section we'll review the fundamentals of **strings**, a date type used to store text, and cover functions and methods to manipulate them

TOPICS WE'LL COVER:

- String Basics
- String Arithmetic
- Indexing & Slicing
- String Methods
- F-Strings

GOALS FOR THIS SECTION:

- Create strings from scratch or from other data types
- Learn to manipulate strings using arithmetic, indexing, slicing, and string methods
- Use f-strings to incorporate variables into strings and make them dynamic

STRING BASICS

- String Basics
- String Arithmetic
- Indexing & Slicing
- String Methods
- F-Strings

Strings are sequences of character data commonly used to store text

You can create a string by using quotation marks, or converting from numbers:

Single *quotes*

```
new_string = 'This is a "string".'
print(new_string)
```

```
This is a "string".
```

Double *quotes*

```
newer_string = "Double quotes allow us to use ' inside."
print(newer_string)
```

```
Double quotes allow us to use ' inside.
```

Triple *quotes*

```
# triple quotes
newerer_string = '''Triple quotes allow us to
write multiline strings. We also don't
have to worry about "errors" due to using
single or double quotes. Pretty cool!
'''
print(newerer_string)
```

```
Triple quotes allow us to
write multiline strings. We also don't
have to worry about "errors" due to using
single or double quotes. Pretty cool!
```

Convert *to string*

```
number = 22.5
string = str(number)

string
```

```
'22.5'
```

The quotes indicate this is a string

```
print(string)
```

```
22.5
```

Using print() cleans up the quotes

STRING ARITHMETIC

- String Basics
- String Arithmetic
- Indexing & Slicing
- String Methods
- F-Strings

Some**arithmetic operators** are compatible with strings

You can **concatenate**strings using +

```
'Snow' + 'board'
```

```
'Snowboard'
```

```
subject = 'My'
noun = 'Maven snowboard'
verb = 'cost'
price = 22.55

subject + ' ' + noun + ' ' + verb + ' $' + str(price) +'!'
```

```
'My Maven snowboard cost $22.55!'
```

You can concatenate any number of strings

The addition of ' ' to add spaces, as well as adding punctuation like '!' or '$' is common

You can **repeat**strings using *

```
'Repeat 3 times: ' + ("I will not steal my coworker's pen! ") * 3
```

```
"Repeat 3 times: I will not steal my coworker's pen! I will not steal
my coworker's pen! I will not steal my coworker's pen! "
```

Repeating strings is not very common in practice

STRING INDEXING

Strings, along with sequence data types, can be navigated via their**index**

- Python is **0-indexed**, which means that the first item in a sequence has an index of 0, not 1

```
message = 'I hope it snows tonight!'
```

message

```
0  I
1
2  h
3  o
4  p
…
22 t
23 !
```

```
message[0]
```

```
'I'
```

```
message[1]
```

```
' '
```

```
message[2]
```

```
'h'
```

String[index]

returns a specified character:

- *String[0] returns the first character*
- *String[1] returns the second character*
- *String[2] returns the third character*

Indexing is a critical concept to master for data analysis, and while the 0-index can be confusing at first, with practice it will be second nature

STRING INDEXING

- String Basics
- String Arithmetic
- Indexing & Slicing
- String Methods
- F-Strings

Strings, along with sequence data types, can be navigated via their **index**

- Python is **0-indexed**, which means that the first item in a sequence has an index of 0, not 1

```
message = 'I hope it snows tonight!'
```

```
message[-1]
```

```
'!'
```

String[index] *returns a specified character:*

- *String[-1] returns the last character*

message

```
-24   I
-23
...
-5    i
-4  g
-3  h
-2  t
-1  !
```

*You can also access individual characters in a text string by specifying their **negative** index*

Negative indices begin at the end of the object and work backwards

Since there is no negative 0, this starts at -1

Indexing a value **greater than the length of the string or object** results in an **IndexError**

```
message[52]
```

```
IndexError: string index out of range
```

STRING SLICING

- String Basics
- String Arithmetic
- Indexing & Slicing
- String Methods
- F-Strings

Slicing returns multiple elements in a string, or sequence, via indexing

```
[start:stop:step size]
```

Index value to start with
(default is 0)

Index value to stop at, not inclusive
(default is all the values after the start)

Amount to increment each subsequent index
(default is 1)

As with indexing, **slicing is a critical concept to master for data analysis**, as it allows you to manipulate a wide variety of data formats

STRING SLICING

String Basics

String Arithmetic

Indexing & Slicing

String Methods

F-Strings

EXAMPLE | *Grabbing components of a text string*

```
message = 'I hope it snows tonight!'
message[0:6:1]
```

```
'I hope'
```

How does this code work?

- **Start: 0**–start with the first element in the string
- **Stop: 6** –stop at the seventh element (index of 6) in the string without including it
- **Step size:1**–return every single element in the range

STRING SLICING

EXAMPLE | *Grabbing components of a text string*

```
message = 'I hope it snows tonight!'
message[:6]
```

```
'I hope'
```

How does this code work?

- **Start: 0**–start with the first element in the string by default
- **Stop: 6** –stop at the seventh element (index of 6) in the string without including it
- **Step size: 1** –return every single element in the range by default

STRING SLICING

String Basics

String Arithmetic

Indexing & Slicing

String Methods

F-Strings

EXAMPLE | *Grabbing components of a text string*

```
message = 'I hope it snows tonight!'
message[2:6]
```

```
'hope'
```

How does this code work?

- **Start:2**–start with the third element (index of 2) in the string
- **Stop:6** –stop at the seventh element (index of 6) in the string without including it
- **Step size:1** –return every single element in the range by default

STRING SLICING

String Basics

String Arithmetic

Indexing & Slicing

String Methods

F-Strings

EXAMPLE | *Grabbing components of a text string*

```
message = 'I hope it snows tonight!'
message[6:]
```

```
' it snows tonight!'
```

How does this code work?

- **Start: 6**–start with the seventh element (index of 6) in the string
- **Stop: 25** –stop at the end of the string
- **Step size: 1** –return every single element in the range by default

STRING SLICING

EXAMPLE | *Grabbing components of a text string*

```
message = 'I hope it snows tonight!'
message[0:6:2]
```

```
'Ihp'
```

How does this code work?

- **Start: 0**–start with the first element in the string
- **Stop: 6** –stop at the seventh element (index of 6) in the string without including it
- **Step size: 2** –return every other element in the range

STRING SLICING

String Basics

String Arithmetic

Indexing & Slicing

String Methods

F-Strings

EXAMPLE | *Grabbing components of a text string*

```
message = 'I hope it snows tonight!'
message[::-1]
```

```
'!thginot swons ti epoh I'
```

How does this code work?

- **Start:-1**–start with the last element in the string by default, due to the negative step size
- **Stop:-25** –stop at the beginning of the string
- **Step size:-1** –return every single element in the range, going backwards

STRING SLICING

String Basics

String Arithmetic

Indexing & Slicing

String Methods

F-Strings

EXAMPLE | *Grabbing components of a text string*

```
message = 'I hope it snows tonight!'
message[-1:-9:-1]
```

```
'!thginot'
```

How does this code work?

- **Start:-1**–start with the last element in the string
- **Stop:-9** –stop at the ninth element from the end of the string without including it
- **Step size:-1** –return every single element in the range, going backwards

STRING SLICING

String Basics

String Arithmetic

Indexing & Slicing

String Methods

F-Strings

EXAMPLE | *Grabbing components of a text string*

```
message = 'I hope it snows tonight!'
message[-9:-1:1]
```

```
' tonight'
```

How does this code work?

- **Start:-9**–start with the ninth element from the end of the string
- **Stop:-1** –stop at last element in the string without including it
- **Step size:1** –return every single element in the range

STRING SLICING

EXAMPLE | *Grabbing components of a text string*

```
message = 'I hope it snows tonight!'
message[-9::]
```

```
' tonight!'
```

How does this code work?

- **Start:-9**–start with the ninth element from the end of the string
- **Stop:25** –stop at the end of the string by default
- **Step size:1** –return every single element in the range by default

THE LENGTH FUNCTION

The **len()** function returns the number of elements in an iterable

```
# len returns the length of strings
len('snowboard')
```

When used on a string, it returns the number of characters

```
9
```

```
# len will return the length of any iterable object
len(['beginner', 'enthusiast', 'pro'])
```

When used on a list, it returns the number of elements within it (more on lists later!)

```
3
```

METHODS

String Basics

String Arithmetic

Indexing & Slicing

String Methods

F-Strings

Methods are functions that only apply to a specific data type or object

- We call them by following the object with a period then the method name

```
message = 'I hope it snows tonight!'
```

```
message.replace('snow', 'rain')
```

```
'I hope it rains tonight!'
```

*replace is a **string method** being applied to message, which is a string*

```
number = 256
number.replace(5, 9)
```

```
AttributeError: 'int' object has no attribute 'replace'
```

*If you try to apply a string method to an integer, you will receive an **AttributeError***

STRING METHODS

- String Basics
- String Arithmetic
- Indexing & Slicing
- String Methods
- F-Strings

find	*Returns the index of the first instance of a given string*	**.find(*string*)**
upper	*Converts all characters in a string to uppercase*	**.upper()**
lower	*Converts all characters in a string to lowercase*	**.lower()**
strip	*Removes leading and trailing spaces, or any other character specified, from a string*	**.strip(*optional string*)**
replace	*Substitutes a given string of characters with another*	**.replace(*string to replace*,*replacement*)**
split	*Splits a list based on a given separator (space by default)*	**.split(*optional string*)**
join	*Joins list elements into a single string separated by a delimiter*	***delimiter*.join(*list*)**

For a list of string methods, visit: ***https://docs.python.org/3/library/stdtypes.html#string-methods***

FIND

The **.find()** method searches for a specified value within a string and returns the index of its first instance

```
message = 'I hope it snows tonight!'
```

```
message.find('snow')
```

```
10
```

***'snow'** first appears in the 11th character (index value of 10)*

- **.find(*string*)**

```
message.find('rain')
```

```
-1
```

-1 is returned if the specified value isn't found

UPPER & LOWER

String Basics

String Arithmetic

Indexing & Slicing

String Methods

F-Strings

The **.upper()** and **.lower()** methods return a given string with all its characters in uppercase or lowercase respectively

```
message = 'I hope it snows tonight!'
```

```
print(message.upper())
```

```
I HOPE IT SNOWS TONIGHT!
```

```
print(message.lower())
```

```
i hope it snows tonight!
```

STRIP

String Basics

String Arithmetic

Indexing & Slicing

String Methods

F-Strings

The **.strip()** method removes leading and trailing spaces *(by default)*,or any other specified character, from a string

- **.lstrip()** removes all leading spaces, and **.rstrip()** removes all trailing spaces

```
message = "      I love the space bar          "

message.strip()
```

```
'I love the space bar'
```

This removes leading and trailing spaces by default

- **.strip(*optional string*)**

```
message = '.Remove the punctuation.'

message.strip('.')
```

```
'Remove the punctuation'
```

This removes leading and trailing periods

REPLACE

String Basics

String Arithmetic

Indexing & Slicing

String Methods

F-Strings

The **.replace()** method substitutes a specified string of characters with another

- This is a common method to remove or change punctuation while cleaning text

```
message = 'I hope it snows tonight!'
```

```
message.replace('snow', 'rain')
```

```
'I hope it rains tonight!'
```

*This replaces all instances of **'snow'** with **'rain'***

- **.replace(*string to replace,replacement*)**

```
message.replace(' ', '')
```

```
'Ihopeitsnowstonight!'
```

*This replaces all **spaces** with an **empty string**, which creates one continuous word*

SPLIT & JOIN

The **.split()** method separates a string into a list based on a delimiter *(space by default)*

```
message = 'I hope it snows tonight!'
word_list = message.split(' ')
word_list
```

```
['I', 'hope', 'it', 'snows', 'tonight!']
```

A new, separate element is created each time a space is encountered

- **split(*optional string*)**

The **.join()** method combines individual list elements into a single string, separated by a given delimiter

```
' '.join(word_list)
```

```
'I hope it snows tonight!'
```

This takes the elements in word_list and combines them into a single string, separated by a single space

- ***delimiter*.join(*list*)**

CHAINING METHODS

Methods can be **chained together** to create powerful expressions

```
message = 'I hope it snows TONIGHT!'
message[6::].lower().split()
```

```
['it', 'snows', 'tonight!']
```

message[6::] →'it snows TONIGHT!'

'it snows TONIGHT!'.lower() →'it snows tonight!

'it snows tonight!'.split() →['it', 'snows', 'tonight!']

How does this code work?

- The string is **sliced**to remove the first two words ('I hope')
- Then it's converted to all lowercase letters with **.lower()**
- And finally divided into a list of separate words with **.split()**

```
len(message[6::].lower().split())
```

```
3
```

You can even wrap this in a len function to return the number of words after 'I hope'

CHAINING METHODS

Methods can be **chained together** to create powerful expressions

- String Basics
- String Arithmetic
- Indexing & Slicing
- String Methods
- F-Strings

```
message = 'I hope it snows tonight!'
```

PRO TIP: Converting text to all upper or lowercase can make many operations, like finding a keyword or joining data much easier

```
message.find('SNOW')
```

```
-1
```

*Find is case-sensitive, so it won't find **'SNOW'** in 'I hope it **snow**s TONIGHT'*

```
message.upper().find('SNOW')
```

```
10
```

By converting the message to uppercase, it finds the match and returns the index

STRING METHOD CHEATSHEET

```
message = 'I hope it snows tonight!'
```

.find(*string*)

Returns the index of the first instance of a given string

```
message.find('snow')
```

```
10
```

.replace(*string to replace,replacement*)

Substitutes a string with another

```
message.replace('snow', 'rain')
```

```
'I hope it rains tonight!'
```

.upper()

Converts all characters in a string to uppercase

```
message.upper()
```

```
'I HOPE IT SNOWS TONIGHT!'
```

.split(*optional string*)

Splits a string into a list based on spaces or a delimiter

```
message.split()
```

```
['I', 'hope', 'it', 'snows', 'tonight!']
```

.lower()

Converts all characters in a string to lowercase

```
message.lower()
```

```
i hope it snows tonight!
```

delimiter.join(*list*)

Combines list elements into a string, separated by a delimiter

```
' '.join(word_list)
```

```
'I hope it snows tonight!'
```

.strip(*optional string*)

Removes leading or trailing spaces or characters

```
message.strip('!')
```

```
'I hope it snows tonight'
```

NOTE: There are many more string methods than covered; to review those, or get additional detail on the methods covered, visit the official Python documentation:

https://docs.python.org/3/library/stdtypes.html#string-methods

F-STRINGS

String Basics

String Arithmetic

Indexing & Slicing

String Methods

F-Strings

You can include variables in strings by using **f-strings**

- This allows you to change the output text based on changing variable values

```
name = 'Chris'
role = 'employee'

print(f"{name} is our favorite {role}, of course!")
```

```
Chris is our favorite employee, of course!
```

Add an **f** before the quotation marks to indicate an f-string, and place the variable names into the string by using curly braces **{}**

```
name = 'Anand'
role = 'customer'

print(f"{name} is our favorite {role}, of course!"
```

```
Anand is our favorite customer, of course!
```

Note that the f-string code doesn't change, but the output does change if the variables get assigned new values

KEY TAKEAWAYS

- Strings are used to store **text** and other sequences of characters
  - *Analysts interact with strings in categorical data, or in large bodies of text that need to be mined for insight*
- Access single characters with **indexing**, and multiple characters with **slicing**
  - *Indexing & slicing are used with many other data types beyond strings, so they are critical to master*
- Learn and leverage **string methods** to facilitate working with strings
  - *You don't need to memorize every method and their syntax on day 1, but it's important to be aware of them*
- Use **f-strings** to incorporate variables into your strings
  - *This will make your text strings dynamic and allow you to avoid writing repetitive code*