Review: Basics of Python

Expressions:

- 2 + 3
- 2**3

Names and assignment:

- class_number = 123

Numerical representations:

- my_int = 2
- my_float = 3.14159

Let’s quickly practice this in Jupyter!

Strings

Text and Strings

A string value is a snippet of text of any length

- 'a'
- 'word'
- "there can be 2 sentences. Here's the second!"

Strings consisting of numbers can be converted to numbers

- int('12')
- float('1.2')

Any value can be converted to a string

- str(5)

Let's explore this in Jupyter!

Types

Every value has a type

We've seen several types so far:

- int: 2
- Built-in function: abs()
- float: 2.2
- str: 'Red fish, blue fish'

The type function can tell you the type of a value

- type(2)
- type('Red fish')

An expression's type is based on its value, not how it looks

- x = 2
- type(x)

Let's explore this in Jupyter!

Conversions

Strings that contain numbers can be converted to numbers

- int('12')
- float('1.2')
- float('one point two') # Not a good idea!

Any numeric value can be converted to a string

- str(5)

Numbers can be converted to other numeric types

- float(1)
- int(1.2) # DANGER: loses information!

Lists

Lists

Lists are ways to store multiple items

We can create lists using square brackets []

- my_list = [2, 3, 4]

We can also access list items using square brackets []

- my_list[2]

Lists can contain elements of different types

- my_list2 = [5, 6, 'seven']

Let's explore this in Jupyter!


text
MaNiPuLaTiOn

Text manipulation

80% of a Data Scientists time is cleaning data

- Text manipulation is a big part of cleaning data

20% of a Data Scientists time is complaining about cleaning data

Python has many string methods that are useful for manipulating text and cleaning data!

Terminology: functions and methods

Recall: Functions take in values (arguments) and (usually) return another value

- E.g., `abs(-5)` `# takes the absolute value of -5 and returns 5`

Methods are functions that operate on particular pieces of data

- i.e., you can think of methods as a function that are attached to specific type of data

The syntax for methods is: `data_object.method()`

Example:

```
"hello".upper()
"HELLO"   # returns capitalized string
```

String methods: capitalization

Some of the simplest string methods involve changing capitalization

Changing capitalization can be useful when combining (joining) data sets as we will discuss later in the semester.

String methods: capitalization

Python strings have a number of methods to change the capitalization of words including:

- .capitalize(): Converts the first character to upper case
- .lower(): Converts a string into lower case
- .upper(): Converts a string into upper case
- .title(): Converts the first character of each word to upper case
- .swapcase(): Swaps cases, lower case becomes upper case and vice versa

Let's explore this in Jupyter!

String methods: splitting and joining strings

There are several methods that can help us join strings that are contained into a list into a single string, or conversely, parse a single string into a list of strings. These include:

- .split(separator_string): Splits the string at the specified separator, and returns a list
- .splitlines(): Splits the string at line breaks and returns a list
- .join(a_list): Converts the elements of an iterable into a string

Let's explore this in Jupyter!

String methods: finding and replacing substrings

Some methods for locating a substring within a larger string include:

.count(substring): Returns the number of times a specified value occurs in a string

.replace(original_str, replacement_str): Replace a substring with a different string

Also:

.startswith(substring): Returns true if the string starts with the specified value

.endswith(substring) : Returns true if the string ends with the specified value

Let's explore this in Jupyter!

Booleans and comparisons

Comparisons

We can use mathematical operators to compare numbers and strings

- Results return Boolean values True and False

Comparison	Operator	True example	False Example
Less than	<	2 < 3	2 < 2
Greater than	>	3 > 2	3 > 3
Less than or equal	<=	2 <= 2	3 <= 2
Greater or equal	>=	3 >= 3	2 >= 3
Equal	==	3 == 3	3 == 2
Not equal	!=	3 != 2	2 != 2

True is equal to 1

False is equal to 0

True + True + False
is equal to…

2

We can compare strings alphabetically

- a < b

Let's explore this in Jupyter!

String methods: checking string properties

There are also many functions to check properties of strings including:

- .isalnum(): Returns True if all characters in the string are alphanumeric
- .isalpha(): Returns True if all characters in the string are in the alphabet
- .isnumeric(): Returns True if all characters in the string are numeric

- .isspace(): Returns True if all characters in the string are whitespaces

- .islower(): Returns True if all characters in the string are lower case
- .isupper(): Returns True if all characters in the string are upper case
- .istitle(): Returns True if the string follows the rules of a title

Let's explore this in Jupyter!

Additional string methods

String methods: string padding

Often we want to remove extra spaces (called "white space") from the front or end of a string

Conversely, sometimes we want to add extra spaces to make a set of strings the same length

- This is known as "string padding"

Python strings have a number of methods that can pad/trim strings including:

- .strip(): Returns a trimmed version of the string (i.e., with no leading or trailing white space)
  - Also, .rstrip() and .lstrip(): Returns a right/left trim version of the string
- .center(num): Returns a centered string (with equal padding on both sides)
  - Also .ljust(num) and .rjust(num): Returns a right justified version of the string
- .zfill(num): Fills the string with a specified number of 0 values at the beginning

Let's explore this in Jupyter!

String methods: filling in strings with values

There are a number of ways to fill in strings parts of a string with particular values.

Perhaps the most useful is to use "f strings", which have the following syntax such as:

- value_to_fill = "my_value"
- f"my string {value_to_fill} will be filled in"

Let's explore this in Jupyter!

Brief mention: regular expressions

More complex text manipulation can be done using “regular expressions”

```
import re
bool(re.match("m.ss", "mess"))
```