Data Types

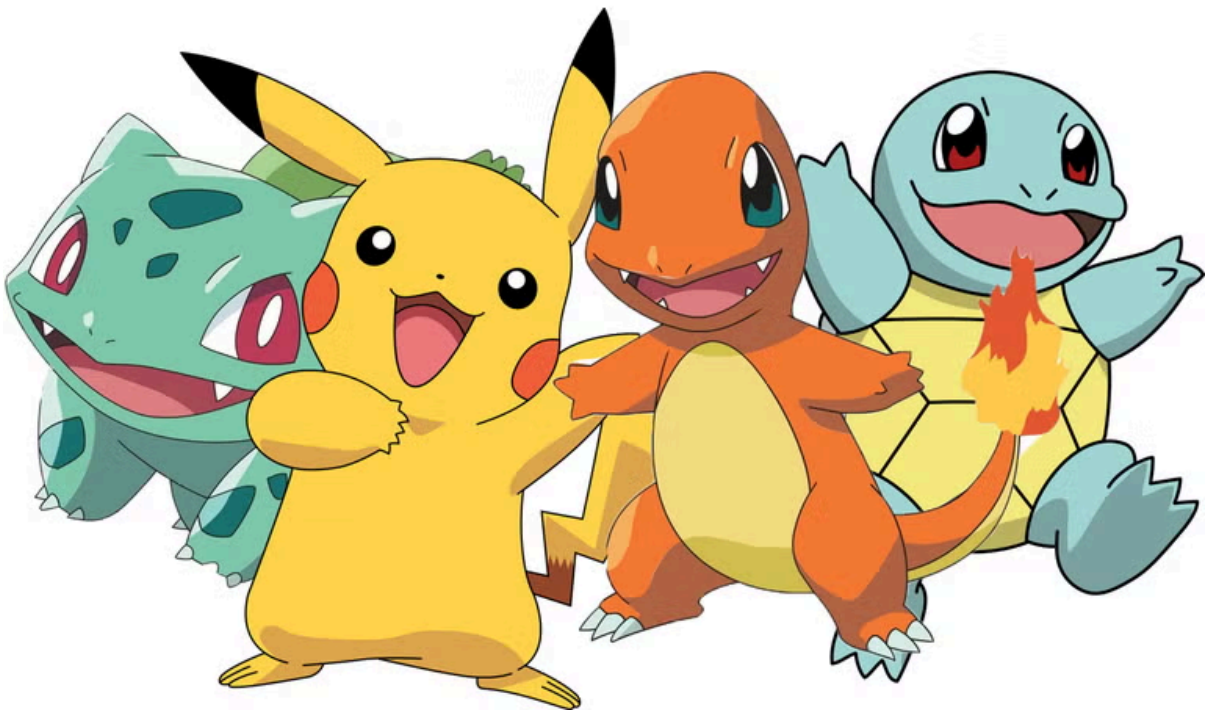
Contents

Note: You can explore the associated workbook for this chapter in the cloud.

There are four essential kinds of Python data with different powers and capabilities:

- Strings (Text)
- Integers (Whole Numbers)
- Floats (Decimal Numbers)
- Booleans (True/False)

They're sort of like starter pack Pokémon!



Skip to main content

Data Types

Take a look at the variables `filepath_of_text` and `number_of_desired_word` in the word count code below.

What differences do you notice between these two variables and their corresponding values?

```
# Import Libraries and Modules

import re
from collections import Counter

# Define Functions

def split_into_words(any_chunk_of_text):
    lowercase_text = any_chunk_of_text.lower()
    split_words = re.split("\W+", lowercase_text)
    return split_words

# Define Filepaths and Assign Variables

filepath_of_text = "../texts/music/Beyonce-Lemonade.txt"
number_of_desired_words = 40

stopwords = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours',
'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers',
 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are',
 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until',
 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here',
 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now', 've', 'll', 'amp']

# Read in File

full_text = open(filepath_of_text, encoding="utf-8").read()

# Manipulate and Analyze File

all_the_words = split_into_words(full_text)
meaningful_words = [word for word in all_the_words if word not in stopwords]
meaningful_words_tally = Counter(meaningful_words)
most_frequent_meaningful_words = meaningful_words_tally.most_common(number_of_desired_words)

# Output Results

most_frequent_meaningful_words
```

You might be wondering...

Why is "../texts/music/Beyonce-Lemonade.txt" colored in red and surrounded by quotation marks while 40 is colored in green and not surrounded by quotation marks? Because these are two different "types" of Python data.

Skip to main content

Data Type	Explanation	Example
String	Text	`"Beyonce-Lemonade.txt", "lemonade"`
Integer	Whole Numbers	`40`
Float	Decimal Numbers	`40.2`
Boolean	True/False	`False`

Check Data Types

You can check the data type of any value by using the function `type()`.

```
type("lemonade")
```

```
str
```

```
type(filepath_of_text)
```

▶ Show code cell output

```
type(40)
```

```
int
```

```
type(number_of_desired_words)
```

▶ Show code cell output

Strings

A *string* is a Python data type that is treated like text, even if it contains a number. Strings are always enclosed by either single quotation marks `'this is a string'` or double quotation marks `"this is a string"`.

```
'this is a string'
```

Skip to main content

```
"this is also a string, even though it contains a number like 42"
```

```
this is not a string
```

It doesn't matter whether you use single or double quotation marks with strings, as long as you use the same kind on either side of the string.

If you need to include a single or double quotation mark *inside* of a string, then you need to either:

- use the opposite kind of quotation mark inside the string
- or "escape" the quotation mark by using a backslash `\` before it

```
"She exclaimed, 'This is a quotation inside a string!'"
```

```
"She exclaimed, \"This is also a quotation inside a string!\""
```

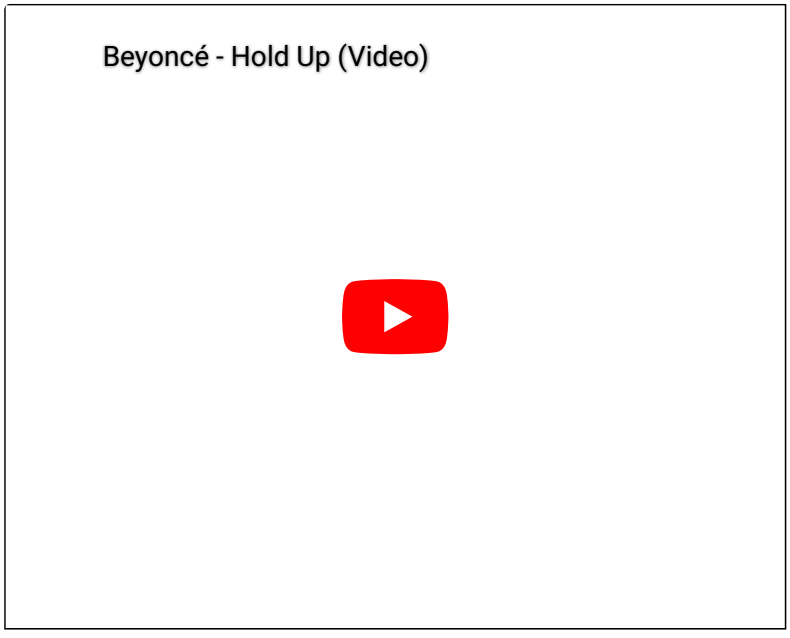
Escape c

A backsl
treat the
characte

String Methods

Each data type has different properties and capabilities. So there are special things that only strings can do, and there are special ways of interacting with strings.

For example, you can *index* and *slice* strings, you can *add* strings together, and you can transform strings to uppercase or lowercase. We're going to learn more about string methods in the next lesson, but here are a few examples using a snippet from Beyoncé's song "Hold Up."




```
lemonade_snippet = "Hold up, they don't love you like I love you"
```

Index

```
lemonade_snippet[0]
```

▶ Show code cell output

Slice

```
lemonade_snippet[0:20]
```

▶ Show code cell output

Add

```
lemonade_snippet + " // Slow down, they don't love you like I love you"
```

▶ Show code cell output

Make uppercase

```
lemonade_snippet.upper()
```

▶ Show code cell output

f-Strings

A special kind of string that we're going to use in this class is called an *f-string*. An f-string, short for formatted string literal, allows you to insert a variable directly into a string. f-strings were introduced with Python version 3.6.

An f-string must begin with an `f` outside the quotation marks. Then, inside the quotation marks, the inserted variable must be placed within curly brackets `{}`.

```
print(f"Beyonce burst out of the building and sang: \n\n'{lemonade_snippet}'")
```

▶ Show code cell output

What do

\n = new

Integers & Floats

An *integer* and a *float* (short for *floating point number*) are two Python data types for representing numbers. Integers represent whole numbers. Floats represent numbers with decimal points. They do not need to be placed in quotation marks.

```
type(40)
```

```
int
```

```
type(40.5)
```

Skip to main content

```
float
```

```
type(40.555555)
```

```
float
```

You can do a large range of mathematical calculations and operations with integers and floats. The table below is taken from Python's documentation about Numeric Types.

Operation	Explanation
`x` + `y`	sum of `x` and `y`
`x` - `y`	difference of `x` and `y`
`x` * `y`	product of `x` and `y`
`x` / `y`	quotient of `x` and `y`
`x` // `y`	floored quotient of `x` and `y`
`x` % `y`	remainder of `x` / `y`
-`x`	`x` negated
+`x`	`x` unchanged
`abs(x)`	absolute value or magnitude of `x`
`int(x)`	`x` converted to integer
`float(x)`	`x` converted to floating point
`pow(x, y)`	`x` to the power `y`
`x` ** `y`	`x` to the power `y`

Multiplication

```
variable1 = 4
variable2 = 2
variable1 * variable2
```

▶ Show code cell output

Exponents

```
variable1 ** variable2
```

▶ Show code cell output

Remainder

```
72 % 10
```

Skip to main content

▶ Show code cell output

Booleans

Booleans are “truth” values. They report on whether things in your Python universe are `True` or `False`. There are the only two options for a boolean: `True` or `False`.

For example, let’s assign the variable `beyonce` the value `"Grammy award-winner"`

```
beyonce = "Grammy award-winner"
```

Python Review

Remember the difference between a single equals sign `=` and a double equals sign `==`?

- A single equals sign `=` is used for variable assignment
- A double equals sign `==` is used as the equals operator

We can “test” whether the variable `beyonce` equals `"Grammy award-winner"` by using the equals operator `==`. This will return a boolean.

```
beyonce == "Grammy award-winner"
```

▶ Show code cell output

```
type(beyonce == "Grammy award-winner")
```

```
bool
```

If we evaluate whether `beyonce` instead equals `"Oscar award-winner"`, we will get the boolean answer.

```
beyonce == "Oscar award-winner"
```

▶ Show code cell output

TypeError

If you don’t use the right data “type” for a particular method or function, you will get a `TypeError.`

Let’s look at what happens if we change the data type `number_of_desired_words` to a string `"40"` instead of an integer.

```
import re
from collections import Counter

def split_into_words(any_chunk_of_text):
    lowercase_text = any_chunk_of_text.lower()
    split_words = re.split("\W+", lowercase_text)
    return split_words
```

Skip to main content

```
stopwords = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours',
 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers',
 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are',
 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until',
 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here',
 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now', 've', 'll', 'amp']


full_text = open(filepath_of_text, encoding="utf-8").read()

all_the_words = split_into_words(full_text)
meaningful_words = [word for word in all_the_words if word not in stopwords]
meaningful_words_tally = Counter(meaningful_words)
most_frequent_meaningful_words = meaningful_words_tally.most_common(number_of_desired_words)

most_frequent_meaningful_words
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-a142b58e454a> in <module>
     29 meaningful_words = [word for word in all_the_words if word not in stopwords]
     30 meaningful_words_tally = Counter(meaningful_words)
---> 31 most_frequent_meaningful_words = meaningful_words_tally.most_common(number_of_desired_words)
     32
     33 most_frequent_meaningful_words

~/opt/anaconda3/lib/python3.7/collections/__init__.py in most_common(self, n)
    584         if n is None:
    585             return sorted(self.items(), key=_itemgetter(1), reverse=True)
--> 586         return _heapq.nlargest(n, self.items(), key=_itemgetter(1))
    587
    588     def elements(self):

~/opt/anaconda3/lib/python3.7/heapq.py in nlargest(n, iterable, key)
    544         pass
    545     else:
--> 546         if n >= size:
    547             return sorted(iterable, key=key, reverse=True)[:n]
    548

TypeError: '>=' not supported between instances of 'str' and 'int'
```

Your Turn!

Here's an example of data types in action using some biographical information about me.

```
name = 'Prof. Walsh' #string
age = 1000 #integer
place = 'Chicago' #string
favorite_food = 'tacos' #string
dog_years_age = age * 7.5 #float
student = False #boolean
```

```
print(f'✨This is...{name}!✨')

print(f"""{name} likes {favorite_food} and once lived in {place}.
```

Skip to main content

```
{name} is {age} years old, which is {dog_years_age} in dog years.
The statement '{name} is a student' is {student}.""")
```

▶ Show code cell output

```
print(f"""
name = {type(name)}
age = {type(age)}
place = {type(place)}
favorite_food = {type(favorite_food)}
dog_years_age = {type(dog_years_age)}
student = {type(student)}
""")
```

▶ Show code cell output

Let's do the same thing but with biographical info about you! Ask your partner a few questions and then fill in the variables below accordingly.

```
name = #Your code here
age = #Your code here
home_town = #Your code here
favorite_food = #Your code here
dog_years_age =#Your code here * 7.5
student = False #boolean
```

```
print(f'✨This is...{name}!✨')

print(f"""{name} likes {favorite_food} and once lived in {place}.
{name} is {age} years old, which is {dog_years_age} in dog years.
The statement "{name} is a student" is {student}.""")
```

Add a new variable called `favorite_movie` and update the f-string to include a new sentence about your partner's favorite movie.

```
name =
age =
home_town =
favorite_food =
dog_years_age =
#favorite_movie =
```

```
print(f'✨This is...{name}!✨')

print(f"""{name} likes {favorite_food} and once lived in {place}.
{name} is {age} years old, which is {dog_years_age} in dog years.
The statement "{name} is a student" is {student}.
# YOUR NEW SENTENCE HERE')
```

Previous
Variables

Next
String Methods