Week 1: Thursday

Data Types and Variables

Terminology

- Data Mining:
  - Collection, Aggregation, Visualization of Data
- Artificial Intelligence:
  - Programming that causes machines to come to conclusions on their own.
- Machine Learning:
  - training a program to make decisions about data by training it on data. (supervised or unsupervised.)

How do we get our data?

- Data Sources (secondary sources: data that was collected for another purpose)
  - collected data.
  - Third-party databases.
- Operational Databases
- undocumented APIs (Application Programming Interface)

Extract-Transform-Load (ETL)

- Extract: Get the data from the source (copying from operational database).
- Transform: Clean and organize the data.
  - Usually summarizing data. (aggregating)
- Load: Put the data into a database.

Data Warehouse vs Data Lake

Data Warehouse	Data Lake
Structured Data	Unstructured Data
Organized Data	Data is not organized
Non-operational Data	Non-operational Data
Schema-on-write	Schema-on-read

Introduction to Python

- High-level programming language
- Interpreted language.
  - Interpreted to machine language at runtime (i.e., as they run).
  - No need to compile the code (or no build step).
  - Dynamically typed language. (types interpreted at run time).
- Scripting language: run line by line, top to bottom.

Introduction to Python, pt. 2

- Dominates data science and machine learning.
  - Competitors: R & Julia
  - SQL is not a programming language but a query language for databases.
    - You need to know SQL to work with databases.

Let's get started with Colab

- Go to Google Colab
- Sign in with your Google account.
- File -> Open Notebook -> Google Drive -> choose `ipynb` file.
- (Show process on the screen).
- Notice, you now have a Colab Notebooks folder in My Drive.

Mounting Google Drive to pull files

```
from google.colab import drive
drive.mount('/content/drive')
```

Navigating the file system

```
ls # list files
cd # change directory
pwd # print working directory or the folder you are in.
```

- Do this in Colab

```
%ls # list files
%cd # change directory
%pwd # print working directory or the folder you are in.
```

navigating the file system, pt. 2

- When to do this?

```
%cd drive # change to the drive folder
%cd MyDrive # change to the MyDrive folder
%cd 'Colab Notebooks' # change to the Colab Notebooks folder

with open('./data/somefile.csv', 'r') as f: # open file in the data folder
    print(f.read())
```

- When not to do this?
  - when you want to open an .ipynb file

Comments

```
# This is a comment
# Comments are not executed
# They are for the programmer to read
x = 5  # this is a comment, but x is read
print(x)  # this is a comment, but x is printed
# print(x), print is commented out here
```

5

Coments, pt. 2

- When to use them?
  - to expain what the code is doing.
  - help make code readable
- When not to use them?
  - when the code is self-explanatory.
  - when the comment is too verbose.
- Make variables descriptive, so you don't need comments.

Variables & Functions = Programming (plus classes, too)

- Variables are containers that hold data.
- Functions are blocks of code that do something.
  - They take in something, do something?, and return something.
  - They take in arguments and return values.
- We can set a variable as the output of a function.
- This will make sense as we go on.

Variables

- Think if variables as containers that hold data.
- Variables are used to store data values.
- Variables are created when you assign a value to them.
- Variables can be the out

The basic data types in Python:

- **Integers**: Whole numbers, positive or negative, without decimals.
- **Floats**: Real numbers, positive or negative, containing one or more decimals.
- **Strings**: A sequence of characters, enclosed in single or double quotes.
- **Booleans**: True or False values.

Basic Data Types: Code

```
string = "Hello, World!"
print("numeric:", 3 + 3)
print("string:", "something " + "and another thing")
print("Float:", float(3) + float(3))
print("Integer:", int(3.0) + int(3.0))
print("String:", str(3.0) + str(3.0))
```

```
numeric: 6
string: something and another thing
Float: 6.0
Integer: 6
String: 3.03.0
```

Complex Data Types in Python

Lists:

- A collection of items
- ordered and changeable
- Allows duplicate members.

Lists: Code

```
# A list is written with square brackets.
# We can create a list in two ways:
# the inferred way
list1 = ["apple", "banana", "cherry"]
# and the explicit way, using the list() constructor to make a List.
list2 = list(("apple", "banana", "cherry"))

print("list1: ", list1)
print("list2: ", list2)
print(f"turn into a string: {' / '.join(list1)}")
```

```
list1:  ['apple', 'banana', 'cherry']
list2:  ['apple', 'banana', 'cherry']
turn into a string: apple / banana / cherry
```

Tuples:

- A collection of items
- ordered and unchangeable
- Allows duplicate members.

Tuples: Code

```
# A tuple is written with round brackets.
# We can create a tuple in two ways:
# the inferred way
tuple1 = ("apple", "banana", "cherry")
# and the explicit way, using the tuple() constructor to make a tuple.
tuple2 = tuple(("apple", "banana", "cherry"))

print(f"tuple1: {tuple1}")
print(f"tuple2: {tuple2}")
```

```
tuple1: ('apple', 'banana', 'cherry')
tuple2: ('apple', 'banana', 'cherry')
```

Sets:

- A collection of items,
- unordered and unindexed
- No duplicate members

Sets: Code

```
# A set is written with curly brackets.
# We can create a set in two ways:
# the inferred way
set1 = {"apple", "banana", "cherry"}
# and the explicit way, using the set() constructor to make a set.
set2 = set(("apple", "banana", "cherry"))


print(f"set1: {set1}")
print(f"set2: {set2}")
```

```
set1: {'cherry', 'apple', 'banana'}
set2: {'cherry', 'apple', 'banana'}
```

Dictionaries:

- A collection of items
- stored as key:value pairs
- unordered, changeable, and indexed
- No duplicate members
- `{key: value, key1: "value", key2: value}`
- But these can be anything: `{"name": "John", "age": 36, "country": "Norway"}`

Dictionaries: Code

```python
# A dictionary is written with curly brackets, and they have keys and values.
# We can create a dictionary in two ways:
# the inferred way
dict1 = {"brand": "Ford", "model": "Mustang", "year": 1964}
# and the explicit way.
dict2 = dict(brand="Ford", model="Mustang", year=1964)

print(f"dict1: {dict1}")
print(f"dict2: {dict2['brand']}")
```

```
dict1: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
dict2: Ford
```

Data types: Inherent Functions (methods)

What is a method?

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

What is an object/class?

- Everything in Python is an object/class.
- An object is an instance of a class. So a class is a blueprint for creating object.
- An object has properties and methods (functions).
- properties are variables.
- methods are functions.
- so the methods are things we can do with a class/object.

What is an object/class, pt.2?

```
# for demonstration purposes only
import unicodedata


class MyStr:
    def __init__(self, string):
        self.value = string
        self.help = (
            "This is a class that takes a string and returns a lowercase version of it."
        )

    def to_lower(self):
        if not self.value:
            return ""
        new_value = ""
        for i in self.value:
            new_value += (
                unicodedata.normalize("NFKD", i)
                .encode("ASCII", "ignore")
                .decode("ASCII")
                .lower()
            )
        return new_value


a = MyStr("Hello, World!")
print(f"value of a: {a.value}")
print(f"help: {a.help}")
print(f"lowercase value of a: {a.to_lower()}")
```

```
value of a: Hello, World!
help: This is a class that takes a string and returns a lowercase version of it.
lowercase value of a: hello, world!
```

What does this mean for you?

- Everything in Python is an object/class.
- So everything has methods (in theory?)

String methods:

```
a = "Hello, World!"
print(a.upper())
print(a.lower())
print(a.replace("H", "J"))
print(a.split(","))
```

```
HELLO, WORLD!
hello, world!
Jello, World!
['Hello', ' World!']
```

Int and Float methods:

- don't really have methods, but there are function for working with them:
- see below. We will discuss.

List methods:

```
a = ["apple", "banana", "cherry"]
a.append("orange")
a.remove("banana")
print(a)
b = a.pop()
print(f"pop: {b}")
print(f"first index: {a[0]}")
print("last index:", a[-1])
print("between index 1 and 3:", a[0:])
```

```
a.pop()
print(f"after popping: {a}")
```

```
['apple', 'cherry', 'orange']
pop: orange
first index: apple
last index: cherry
between index 1 and 3: ['apple', 'cherry']
after popping: ['apple']
```

Tuple methods:

```
a = ("tuple1", "another_tuple", "last_tuple")
print(a[0])
print(a[-1])
print(a[0:])
print(a.count("tuple1"))
```

```
tuple1
last_tuple
('tuple1', 'another_tuple', 'last_tuple')
1
```

Set methods:

```
a = {"apple", "banana", "cherry"}
a.add("orange")
a.remove("cherry")
a.add("banana")
print(a)
```

```
{'orange', 'apple', 'banana'}
```

Dictionary methods:

```python
a = {"brand": "Ford", "model": "Mustang", "year": 1964}
print(a)
print(a.keys())
print(a.values())
print(a.items())
print(a.get("brand"))
print(a["brand"])
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
dict_keys(['brand', 'model', 'year'])
dict_values(['Ford', 'Mustang', 1964])
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
Ford
Ford
```

Math	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus: remainder of first number divided by second
**	Exponentiation: first number to the power of second
//	Floor Division: quotient of the division, rounded down

Simple Math

```python
a = 5
b = 10
c = a + b
d = a - b
e = a * b
f = a / b
print(f"c: {c}")
print(f"d: {d}")
print(f"e: {e}")
print(f"f: {f}")
```

```
c: 15
d: -5
e: 50
f: 0.5
```

The Weird Ones

```python
a = 5
b = 10
x = 9
y = 21
c = a % b
d = a**b
e = x // a
f = y // a
print(f"c: {c}")
print(f"d: {d}")
print(f"e: {e}")
print(f"f: {f}")
```

```
c: 5
d: 9765625
e: 1
f: 4
```

Data Type Casting