# Introduction to Urban Data Science

# Lecture 2 Basics of Python

Wenzheng Li
Hazel (Yujin) Lee

# Announcement

## TA and instructor office hours

**Instructor**

Office Hours: Monday 2:30 – 4:30pm and Wednesday 11:00am-1:00pm in Sibley Hall 214.
Book a time here

**TA and GTRS**

Yujin Hazel Lee (TA) yl3276@cornell.edu
Office hours: Thursday 5:30-6:30 pm in Sibley Hall 101
Xi Guan (GTRS) xg298@cornell.edu
Office hours: Tuesday 4:30-5:30pm in Sibley Hall 305

## Deadline

In-class exercise - due every Thursday (unless otherwise notified).

# OUTLINE

- Jupyter notebook (continue)
- Python Basics

# Jupyter notebook (continue) 1

# jupyter notebook

For more functions regarding the Jupyter Notebook Markdown, please refer to:
https://www.youtube.com/watch?v=uVLzL5E-YBM

For basic Markdown syntax:

https://www.markdownguide.org/basic-syntax/

https://jupyter.org/try-jupyter/notebooks/?path=notebooks/Intro.ipynb

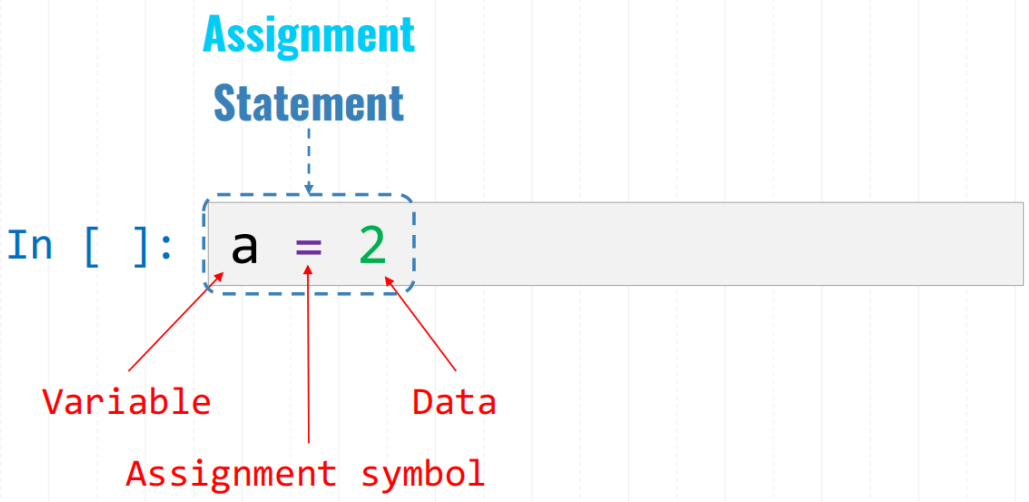
# Python Basics 2

# Basics of Python: Variables

- Create a variable using “=” (e.g., a = 1)
- Variable name:
  - can be any length
  - can consist of letters ( A-Z , a-z ), digits ( 0-9 ), and the underscore character ( _ )
  - CANNOT start with a digit
  - Never name a variable using built-in function names, e.g., print(), len(), list()

9_var=3 or len=5 (Wrong!)

var_9=3 (correct!)



# Basics of Python: Variables

- Numeric: integer (int), decimal number (float)
  - int: e.g., 10
  - float: e.g., 11.80353453

- Text: string (str)
  - string: e.g., "Hello!“

- Boolean: True or False (bool)
  - 3==4 (False)
  - 5 in [1,3,5,7] (True)

- List: collection of numeric, string, or Boolean type data:
  - List: e.g., [1, “hello”, True]
  - Dictionary: e.g., {‘a’: 1, ‘b’: 2, ‘c’:3}
  - Tuple: e.g., (1 , 2, 3). can be understanded as read-only list
  - set: e.g., {1, 2, 3}, an unordered collection of unique elements

# Basics of Python: Variables

***Python counts from 0***

***Python built-in functions/methods***

- **pre-defined functions** that are readily available to use. No need to define these functions; we directly use them.

- Built-in Functions:
  - ***type( )***: check a variable type (integer, string….)
  - ***print( )*****:** print a specified message to the screen
  - ***int( ):*** convert a variable into an integer: int(var1), round down….
  - ***round():*** round to the nearest integer

# Basics of Python: Operators

**Arithmetic operators:**

- +, -, *, /  add, subtract, multiple, and divide
- %: mod, get the remainder of a division. e.g., 3%2=1
- **: power. 10**3, ten to the third power
- //: get the integer part of a quotient. e.g., 7//3=2

**Comparison operators:**

- <, >, <=, >=
- ==: if the values of two operands are equal, "==" returns True, otherwise, it returns False
- !=: if the values of two operands are **NOT** equal, "!=" returns True, otherwise, it returns False

**Logical operators:** evaluate Boolean expressions and determine the logic between conditions

**and (&)**: If both the operands are true then the condition becomes true.
**Or (|)**: If any of the two operands are true then the condition becomes true.
**not**: Get the reversed output

# Basics of Python: List

- Place all the items (elements) inside square brackets [ ], separated by commas.

- Can have any number of items and they may be of different types
  - my_list = [1, "Hello", 3.4]

- two ways to define a list:
  - using [ ]. For example, my_list = [1, "Hello", 3.4]
  - using list(range( )) — two functions: list( ) and range( )

# Basics of Python: List

- **How to select an element or a subset of elements from a list?**

  - **indexing** means selecting an individual element from a list using an **index**.
  - **An index denotes the position of an element in a list.**

  - **slicing** means selecting a subset of elements from a list that is obtained based on the indexes.

# Basics of Python: List

## Indexing

- To select an element of the list, we use the index operator [ ]. e.g., **my_list[*index*]**
- Always remember **Python index counts from 0**

index 0 1 2 3

[0, 1, 2, 3]

Code:

```
l = [0, 1, 2, 3]
print(l[0])
```

Output:

```
0
```

index 0 1 2 3

[5, 3, 9, 300]

Code:

```
l = [5, 3, 9, 300]
print(l[0])
```

Output:

```
5
```

# Basics of Python: List

## Slicing

- To select a subset of elements, we can use **my_list[*start*:*stop*]**
  - The arguments *start* and *stop* denote the corresponding index. Note that the *start* bound is included in the output. The *stop* bound is one step BEYOND the element you want to select (exclusive).

- Example:

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|

```
lst = [15, "a", 14, 64, 75, 100, 110, 1000]
print(lst[2:5])
```

- Output?

# Basics of Python: List

List-specific methods:

- **my_list.append(*item*)**
  - adds an item to the end of the list
  - *Item (*an argument*)*: the item to be added at the end of the list. The item can be any data type.

- **my_list.index(*element*)**
  - returns the first index of the given element in the list

- **len(*my_list*)**
  - Get the length of a list

# Basics of Python: String

- A sequence of characters, for example, "hello world" and "Way2go".
- Double-quote and single-quote can be used interchangeably (e.g., “a” and ‘a’ are the same).
- Concatenate two strings using **+**

- How to get access to one character or a subset of characters in a string? Using **indexing and slicing**

- String-specific methods:

  - str**.find():** Returns the index of the first occurrence of the substring (if found). If not found, it returns -1.
  - str**.replace(*old_str*,*new_str*)** replace the old substring with the new substring
  - str**.startswith(string)** returns True if a string starts with the specified prefix. If not, it returns False.

# Basics of Python: if statement

The if statement is used when we want to execute a part of code (or make a certain decision) only if a certain condition is satisfied.

In [ ]:

```
a = [0, 1, 2, 3]

if len(a) == 4:
    print('The list has 4 elements.')
else:
    print('The list does have 4 elements.')
```

Expression

Indentation matters

```
The list has 4 elements.
```

# Basics of Python: for statement

**1. if statement:**

```
if CS:
    DM that satisfies the CS
```

- Conditional statement (CS); Decision Making (DM)
- Will execute the DM only if the CS is True; otherwise, DM will not be executed.
- DM starts with an indentation. The first un-indented line marks the end of a DM.

# Basics of Python: for statement

**2. if...else statement**

```
if CS1:
    DM1 that satisfies the CS1
else:
    DM2 that satisfies the else condition
```

- If the CS1 is true, DM1 will be executed only.
- If the CS1 is False, DM2 will be executed.
- Each DM is indicated by an indentation.

# Basics of Python: for statement

### 3. The if...elif...(elif)...else statement

```
if CS1:
    DM1 that satisfies the CS1
elif CS2:
    DM2 that satisfies the CS2
elif CS3:
    DM3 that satisfies the CS3
else:
    DM4 that satisfies the else condition
```

- You can make many "elif CS" depending on how many DMs you have.
- in if...elif...(elif)...else statement,
  - there is only **one** if as the start statement.
  - there can only be **one** "else" block, and it must be at the end.
  - "else" is not mandatory.

# Basics of Python: for-loop

- **A for-loop structure is used for iterating over a sequence (either a list or a string).**

- **Task:** Print each element in list a
- **Use** for-loop

In [ ]:
```
a = [0, 1, 2, 3]
```

# Basics of Python: for-loop

- **Method 1:** Loop over each element in the list

In [ ]:

```
a = [0, 1, 2, 3]

for i in a:
    print(i)
```

Indentation matters

```
0
1
2
3
```

i represents each *element* in list a

# Basics of Python: for-loop

- **Method 2:** Loop over the index of each element using
  - len(): returns the number of items in an object. E.g., len(a) is 4.
  - range(): returns a sequence of numbers, by default, starting from 0 and increments by 1.
    E.g., range(len(a)) is just range(4), and it returns 0,1,2,3 (Note: 4 is not returned)
    E.g., range(1, 4) returns 1,2,3 (Note: starts from 1 and 4 is not returned)

In [ ]:

```
a = [0, 1, 2, 3]

for i in range(len(a)):
    print(a[i])
```

Indentation matters

```
0
1
2
3
```

i represents each ***index*** in list a

# Exercise (another way to sum)

- Sum the numbers 1 through 100 **in a loop** and print the result.
  - For loop

i

1 + 2+ 3+ 4+ … + 100

s

0 +1 + 2+ 3+ 4+ … + 100

s

s

s

# Optional: Basics of Python: For statement (continue and break)

- break statement **terminates the entire loop, skipping any remaining iterations, and jumps to the next code block outside the for-loop.**

- continue statement **skips the rest of the current iteration, and moves directly to the next iteration of the for-loop**.

```
for var in sequence:
    # codes inside for loop
    if  condition:
        break
    # codes inside for loop

# codes outside for loop
```

```
for var in sequence:
    # codes inside for loop
    if  condition:
        continue
    # codes inside for loop

# codes outside for loop
```

# Questions