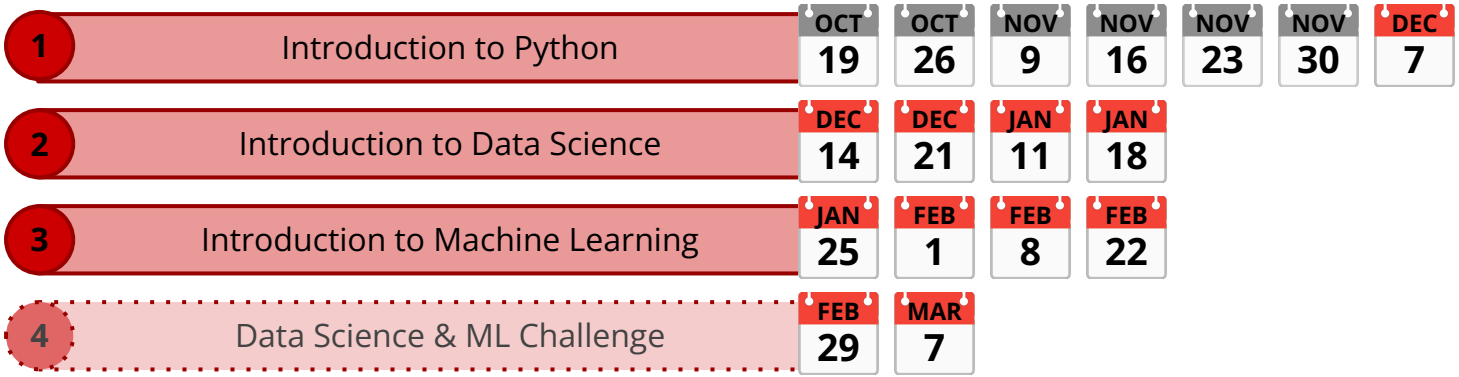
Python for Data Science and Machine Learning

School Year 2023-2024

IST

Course Structure



Jupyter Notebook Setup

In a browser:

```
192.168.10.4:8888
```

Password: **ist**

Recap: Comparisons

- 5 is larger than 3

```
5 > 3
```

- -5 is larger than 9

```
-5 > 9
```

- 2 is the same as 2

```
2 == 2
```

- **not** (negation)

```
not True
```

```
not (5 < 3)
```

- **and** (both must be true)

```
(5 < 6) and (5 < 10)
```

- **or** (either must be true)

```
(5 < 3) or (5 < 10)
```

Recap: If-Statements

You can chain multiple conditions with **`elif`**.

What is the difference between these two snippets of code?

```
x = int(input())

if x < 3:
   print("X is less than 3")
elif x < 10:
   print("X is less than 10")
elif x < 25:
   print("X is less than 25")
```

```
x = int(input())

if x < 3:
   print("X is less than 3")
if x < 10:
   print("X is less than 10")
if x < 25:
   print("X is less than 25")
```

Recap: While-Loops

Allows you to repeat instructions

With an **if-statement**:

```
x = int(input("Insert num < 5: "))

if x >= 5:
   print("ERROR! Wrong number")
   x = int(input("Insert num < 5: "))

print("CORRECT!")
```

With a **while-loop**:

```
x = int(input("Insert num < 5: "))

while x >= 5:
   print("ERROR! Wrong number")
   x = int(input("Insert num < 5: "))

print("CORRECT!")
```

Recap: For-Loops

Repeat a specific amount of times

With a **while-loop**:

```
x = 0

while x < 10:
    print(x)
    x += 1
```

With a **for-loop**:

```
for x in range(10):
    print(x)
```

```
for x in range(2, 10):
    print(x)
```

```
for x in range(2, 10, 3):
    print(x)
```

Recap: Lists

Modifiable containers for data.

With **variables**:

```
num1 = 42
num2 = 100
num3 = 10

print(num1)
print(num2)
print(num3)
```

With a **list**:

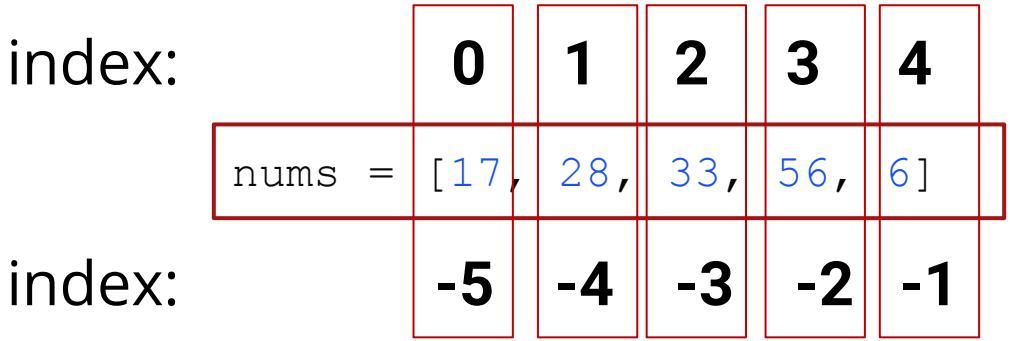
```
nums = [42, 100, 8]

print(nums)
```

Recap: Accessing List Elements

To access list elements you can use the **[index]** operator.

NOTE: List indices start from **0**




```
print(nums[0])
```

```
print(nums[3])
```

```
print(nums[-2])
```

Recap: Modifying Lists

Adding new elements:

1. To insert at the back: **append**
2. To insert in any position: **insert**

```
nums = [42, 100]

nums.append(8)
nums.insert(0, 200)
elem = nums.pop(1)

print(nums)
```

Removing elements:

1. To an element: **pop**

   You may optionally pass an index, default is **-1**.

Recap: Additional List Functions

Additional functions that operate on lists

- Get the length of the list: **len**

```
len([4, 8, 10, 12])
```

```
len([-3])
```

```
len([])
```

- Get the max/min elements in a list: **max** and **min**

```
max([4, 8, -2, 0])
```

```
min([4, 8, -2, 0])
```

- Get the sum of all elements in a list: **sum**

```
sum([4, 8, -2, 0])
```

```
sum([-3])
```

Recap: Iterating Lists

Python provides multiple ways to **iterate over lists**.

The most used methodologies are:

Index-iteration:

```
nums = [10, 20, 30, 40]
for i in range(len(nums)):
    print(nums[i])
```

For-each loop:

```
nums = [10, 20, 30, 40]
for num in nums:
    print(num)
```

The output of the two snippets is identical

Recap Exercise

Complete the **7.0** program.

You are given:

- The list `data` containing integers `[4, 8, 12, 16]`.
- An empty list `doubles`.

Write a program that follows the following steps, what is the output of this program?

1. You a **for-each** loop to add the double of each number from `data` to the `doubles` list.
2. Remove the last element from `doubles` and assign it to a variable `elem`
3. Append the first element of `data` to the end of `doubles`.
4. Insert `elem` as the second element of `doubles`
5. Print the final `doubles` list.

Exercise 7.0 - Solution

```
data = [4, 8, 12, 16]
doubles = []

# 1
for number in data:
    doubles.append(number * 2)
# 2
elem = doubles.pop()
# 3
doubles.append(data[0])
# 4
doubles.insert(1, elem)
# 5
print(doubles)
```

Prints:

```
[8, 32, 16, 24, 4]
```

Recap: Dictionaries

Group data together using keys

With **variables**:

```
num1 = 42
num2 = 100
num3 = 10

print(num1)
print(num2)
print(num3)
```

With a **dict**:

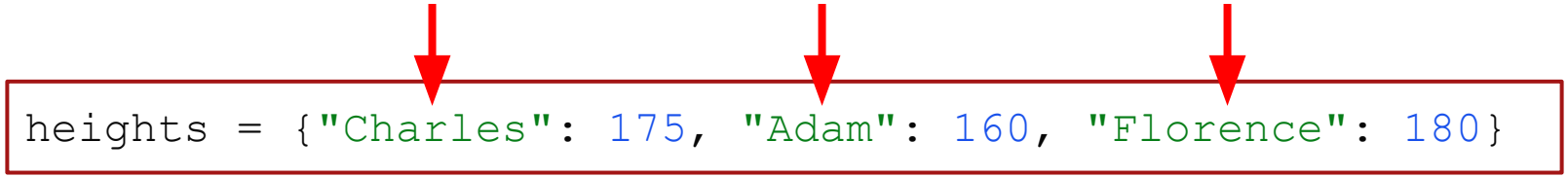
```
nums = {"num1": 42, "num2": 100, "num3": 8}

print(nums)
```

Recap: Accessing Dictionary Elements

To access dictionary elements you can use the **[index]** operator.

NOTE: You can only access keys that exist



```
heights = {"Charles": 175, "Adam": 160, "Florence": 180}
```

```
print(heights["Adam"])
```

```
print(heights["Florence"])
```

ERROR:

```
print(heights["Dan"])
```

Recap: Modifying Dictionaries

```
data = {"a": 42, "b": 3}
```

1. To insert a new key:

```
data["c"] = 800
```

```
data["d"] = 4.5
```

2. To modify an existing elements you can assign to the key

```
data["a"] = 10
```

```
data["b"] = 3.2
```

3. You can remove elements in a dict with the **del** function.

```
del data["a"]
```

```
del data["c"]
```

Recap: Iterating Dictionaries

Python provides multiple ways to **iterate over dicts**.

The most used methodologies are:

Key-iteration:

```
data = {"a": 4, "f": 1, "z": 8}

for key in data:
    value = data[key]
    print(key, value)
```

For-each loop:

```
data = {"a": 4, "f": 1, "z": 8}

for key, value in data.items():
    print(key, value)
```

The output of the two snippets is identical

Recap Exercise

Complete the **7.1** program.

Write a program given a dictionary named `gifts` with keys as names and gifts as values:

```
gifts = {"Alice": "Teddy Bear", "Bob": "Train Set", "Eve": "Doll"}
```

1. Adds a new entry with `"Charlie"` as the key and `"Book"` as the value.
2. Changes `"Eve"`'s gift to `"Bicycle"`.
3. Removes `"Bob"` from the dictionary.
4. Using a for-each dictionary loop, iterates over `gifts` and print each name and their corresponding gift in the format `"Name: Gift"`.
5. Prints the final output dictionary `gifts`

Exercise 7.1 - Solution

```
# 1
gifts["Charlie"] = "Book"

# 2
gifts["Eve"] = "Bicycle"

# 3
del gifts["Bob"]

# 4
for name, gift in gifts.items():
    print(name + ": " + gift)

# 5
print(gifts)
```

Prints:

```
{
    'Alice': 'Teddy Bear',
    'Eve': 'Bicycle',
    'Charlie': 'Book'
}
```

Recap: Sets

Unordered collections of unique elements

With **variables**:

```
num1 = 42
num2 = 100
num3 = 42

print(num1)
print(num2)

if (num3 != num1) and (num3 != num2):
    print(num3)
```

With a **list**:

```
nums = {42, 100, 42}

print(nums)
```

Recap: Anatomy of a Set

Anatomy of a set:

1. Uses curly brackets **{ }**
2. Elements separated by comma **,**
3. Can take any values (will remove duplicates)

```
nums = {42, 100, 42}
```

```
data = {"A", "C", "D"}
```

Recap: Modifying Sets

Adding new elements:

1. To insert an element: **add**
2. To remove an element: **remove**

```
nums = {42, 100}

nums.add(8)
nums.remove(100)
nums.add(50)

print(nums)
```

Recap: Set Theory

Set theory operations:

```
set1 = {"A", "B", "C"}
set2 = {"B", "C", "D"}
```

1. Union: **set1 | set2** `{"A", "B", "C", "D"}`
2. Intersection: **set1 & set2** `{"B", "C"}`
3. Difference: **set1 - set2** `{"A"}`

Recap Exercise

Complete the **7.2** program.

Given two sets: `set_a` with the numbers `{1, 2, 3}` and `set_b` with `{3, 4, 5}`.

1. Add the number `6` to both `set_a` and to `set_b`
2. Removing the number `1` from `set_a`.
3. Form a new set `u_set` that combines all elements from both `set_a` and `set_b`.
4. Now, create `i_set` that contains only elements common to both `set_a` and `set_b`.
5. Next, make `d_set` that has elements in `set_a` but not in `set_b`.
6. Print `u_set`, `i_set`, and `d_set`.

Exercise 7.2 - Solution

```
# 1
set_a.add(6)
set_b.add(6)

# 2
set_a.remove(1)

# 3
u_set = set_a | set_b

# 4
i_set = set_a & set_b

# 5
d_set = set_a - set_b
```

Prints:

```
# u_set
{2, 3, 4, 5, 6}

# i_set
{3, 6}

# d_set
{2}
```

Iterating Sets

Python provides one way to **iterate over sets**.

This makes set and list iteration very similar:

For-each loop:

```
nums = {40, 10, 30, 20}
for num in nums:
    print(num)
```

Remember sets are unordered (so no ordering guarantees!)

Exercise

Complete the **7.3** & **7.4** programs.

- **7.3**: Create a set named `numbers` with elements `{1, 2, 3, 4, 5}`.

  Iterate through `numbers` and print each number.

- **7.4**: Using a different list `numbers`, iterate through the list. In each iteration:
  a. If the number is even add it to the set `even`.
  b. If the number if odd add it to the set `odd`.
  c. Print `even` and `odd`. Why are there no duplicates?

Solution 7.3

```
# 1
numbers = {1, 2, 3, 4, 5}

# 2
for number in numbers:
    print(number)
```

Solution 7.4

```
numbers = [1, 2, 3, 2, 5, 2, 4, 1, 7, 6, 2, 0, 3, 15]
even = set()
odd = set()

for number in numbers:
    if number % 2 == 0:
        even.add(number)
    else:
        odd.add(number)

print(even)
print(odd)
```

Data-Structure Membership

You can use the **in** keyword to check if an element is in a given data structure. This applies to **lists**, **sets** and **dictionaries**.

```
data1 = ["a", "b", "c"]
x = "b"

print(x in data1)
```

```
data2 = {"a", "b", "c"}
y = "b"

print(y in data2)
```

```
data3 = {"a": 10, "b": 20}
z = "b"

print(z in data3)
```

Exercise

Complete the **7.5** & **7.6** programs.

- **7.5**: Given a set named `fruits`.
  a. Check if `"apple"` is in the `fruits` set and print the result.
  b. Check if `"orange"` is in the `fruits` set and print the result.

- **7.6**: Write a program that:
  a. Creates a list `colors` with the elements `["red", "blue", "green"]`.
  b. Creates a set `primary_colors` with the elements `{"red", "yellow", "blue"}`.
  c. Checks if the `specific_color` variable is in both `colors` and `primary_colors`
     Print the result.
  d. Checks if each element in `colors` is also in `primary_colors`.
     Print each color and whether it is a primary color.

Solution 7.5

```
fruits = {"apple", "banana", "cherry"}

print("Is apple in fruits?", "apple" in fruits)
print("Is orange in fruits?", "orange" in fruits)
```

Solution 7.6

```python
# 1
colors = ["red", "blue", "green"]

# 2
primary_colors = {"red", "yellow", "blue"}

# 3
specific_color = "green"
print(
    "Is " + specific_color + " in both?",
    specific_color in colors and specific_color in primary_colors,
)

# 4
for color in colors:
    print("Is " + color + " a primary color?", color in primary_colors)
```

Final Exercise

Complete the **7.7** program.

Given the two lists:

```
ist = ["Alice", "Bob", "Charlie", "Diana", "Ethan"]
ism = {"Bob", "Diana", "Fiona", "George", "Hannah"}
```

1. Check if each student in `ist` is also enrolled in ISM. If so, add them to a new set `common`.
2. Create a set `ism_only` which contains only the `ism` students that are not in the `common` set.
3. For each element in `ist`, using a loop, print if they are present in the `ism_only` set.

Solution 7.7

```
ist = ["Alice", "Bob", "Charlie", "Diana", "Ethan"]
ism = {"Bob", "Diana", "Fiona", "George", "Hannah"}
common = set()

# 1
for student in ist:
    if student in ism:
        common.add(student)

# 2
ism_only = ism - common

# 3
for student in ist:
    print(student in ism_only)
```

End of Class

See you all next week!