# APS106

UNIVERSITY OF TORONTO

## Tutorial 9 – Week 10

*We'll be starting at the 10 minute mark*

if nothing else, write #cleancode

# Agenda

- Lecture review
  - Aliasing
  - Passing Mutable Objects into a Function
  - Passing Immutable Objects into a Function
  - Default function values
  - File I/O
- Practice questions

# Learning Objectives

After this tutorial, learners should be able to:

- recognize / describe / create  aliases for Python objects
- recognize / describe mutable/immutable objects
- use mutable/immutable objects as arguments for functions
- recognize / describe / create optional function parameters
- open regular files
- open csv files
- operate with regular file objects, e.g., iterate over, close, etc
- operate with csv file objects, e.g., iterate over, close, etc
- use the built-in  context manager “open” (for file management)
- recognize / describe / create dictionaries
- recognize / describe / create sets
- iterate over tuples/sets/dictionaries

# APS106

UNIVERSITY OF TORONTO

## Review of Lecture

### Aliasing

if nothing else, write #cleancode

# Aliasing (Review)

- Two variables are said to be **aliases** when they reference the same object.
- If the object represents a mutable value, changes made through one variable are visible to all aliases,
- Example:

```
>>>  lst1 = [2, 4, 6, 8]
>>>  lst2 = lst1
>>>  lst2[0] = "hi"
```

```
>>>  print(lst1)
     ['hi', 4, 6, 8]
>>>  print(lst2)
     ['hi', 4, 6, 8]
```

- You can  check if two variables are aliases by using the **`is`** operator, which returns **`True`** if both variables reference the same object

```
>>>  lst1 is lst2
     True
```

# Using Mutable Objects as Function Arguments[1]

- If an object passed to a function as argument is **mutable**, modification of the object made within the function may be "visible" outside the function

- **WHY?**

  The function gets a **reference** to the object, not a copy of it, and modifications to the object may be visible to variables outside the scope of the function via the aliasing mechanism

# Using Mutable Objects as Function Arguments[2]

```
def func(my_input):
    my_input *= 2
```

The list object `my_input` references is changed "in place"

```
>>> x = [[1, 2], [2,3]]
>>> print(x)
    [[1, 2], [2,3]]

>>> func(x)
>>> print(x)
    [[1, 2], [3, 4], [1, 2], [3, 4]]
```

When func is called, variables `x` and `my_input` become aliases

The value of variable `x` was affected by the execution of `func()`

# Using Mutable Objects as Function Arguments[3]

```
def func1(my_input):
    my_input = my_input + my_input
```

`my_input` will reference a NEW list object !

```
>>> x = [[1, 2], [2,3]]
>>> print(x)
    [[1, 2], [2,3]]

>>> func1(x)
>>> print(x)
    [[1, 2], [2,3]]
```

The value of variable x was NOT affected by the execution of func()

# Using Immutable Objects as Function Arguments[1]

- If the object passed as an argument to a function is **immutable**, the function gets a **reference** to the immutable object
- If the the immutable object has no mutable components, no changes to the object are possible (no "side effects" occur)

```
def func(my_input):
    my_input *= 2

>>> x = 5
>>> func(x)
>>> print(x)
    5
```

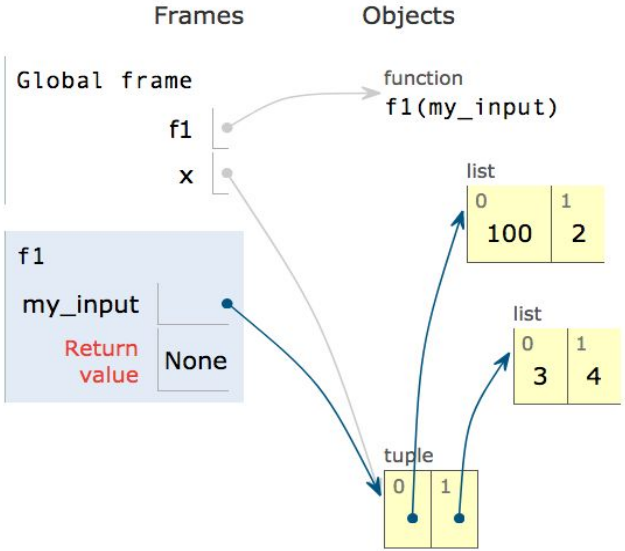
The value of variable `x` was not affected by the execution of `func2()`

# Using Immutable Objects as Function Arguments[2]

If the the immutable object has mutable components, changes to the mutable components are possible and these changes can become visible outside the function

```
def f1(my_input):
    my_input[0][0] = 100

>>> x = ([1,2],[3,4])
>>> f1(x)
>>> print(x)
    ([100, 2], [3, 4])
```




The value of variable `x` was affected by the execution of `f1()`

# Recap: Aliasing and Functions – 1

- In Python, arguments are passed to functions **by reference.**
  - In other words, **links** to objects are passed to functions, **NOT copies** of the objects.

- If the object being passed to a function is:
  - **mutable** (e.g., a list, dictionary, set), if a function modifies the object passed as an argument, the change is visible outside the function (the function is said to produce "side effects");
  - **immutable** (e.g., integer, string, tuple): it is not possible to change anything about that object in the function (no side effects)

# Recap: Aliasing and Functions – 2

Aliasing can be helpful, but if you do not intend for the function to change a mutable object, create a copy of the mutable object and pass that to functions, or use copies of the mutable object inside functions and change those.

Aliasing:

```
>>> x = [1, 2, 3, 4]
>>> y = x
>>> print(x is y)
    True
```

Copying 1:

```
>>> y = [1, 2, 3, 4]
>>> x = y.copy()

>>> print(x is y)
    False
```

Copying 2:

```
>>> x = [1, 2, 3, 4]
>>> y = x[:]

>>> print(x is y)
    False
```

Copying 3:

```
>>> x = [1, 2, 3, 4]
>>> y = list(x)
>>> print(x is y)
    False
```

# APS106

UNIVERSITY OF TORONTO

## Review of Lecture

## Advanced Functions

if nothing else, write #cleancode

# Optional Parameters – Default Values

- Parameters can be set up in function definitions to have default values
  => these values are optional and you don't need to specify them when calling the function

- **Example: sep, end, file** and **flush** are optional parameters in **`print()`**

```
>>> help (print)
 print(...)
 print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

 Prints the values to a stream, or to sys.stdout by default.
 Optional keyword arguments:
 file:  a file-like object (stream); defaults to the current sys.stdout.
 sep:   string inserted between values, default a space.
 end:   string appended after the last value, default a newline.
 flush: whether to forcibly flush the stream.
```

# Optional Parameters in Built-in Functions

- **Example:** in function **`print()`**, the default values of the optional parameters **sep** and **end** are a blank space and `'\n'` (the newline marker), respectively:

```
>>> print("one", "two", "three", "four")
    one two three four
```

- We can specify different values for **sep** and **end**:

```
>>> print("one", "two", "three", "four", sep = '...', end = "!!!")
    one...two...three...four!!!!
```

# Optional Parameters in User-defined Functions

- Optional parameters can also be defined in user-defined functions

```
def func(x, y = 2):
    return x * y


# the y parameter takes on the default value 2
print('Using the default value:', func(3))

# the default value of y is overriden
print('Default value overridden:', func(3, 5))
```

```
When using the default value func() returns 6
When the default value is overridden, func() returns 15
```

# APS106

UNIVERSITY OF TORONTO

## Review of Lecture

## Program Input and Output – Files

if nothing else, write #cleancode

# Opening Files

The general form for opening a file is:

```
open(file_path, mode)
```

`mode` can be:

- `'r'` for reading
- `'w'` for writing
- `'a'` for appending
- `'r+'` for reading and writing
- `'a+'` for appending and reading

`open()` returns a **file object**.

# Writing to a File

- The following statement opens the file for writing (note parameter **mode** is set to 'w')

```
myfile = open("test.txt", "w")
```

**my_file** is a **file object**

- To write something to the file we can use the **write** method of the file object:

```
myfile.write("My first file ...")
```

- Once we have finished with the file, we need to close it:

```
myfile.close()
```

# Reading a file

| Approach | Code | When to use it |
|---|---|---|
| The read approach | `myfile = open(filename, 'r')`<br>`contents = myfile.read()`<br>`myfile.close()` | When you want to read the whole file at once and use it as a string |
| The readline approach | `myfile = open(filename, 'r')`<br>`contents = ''`<br>`line = myfile.readline()`<br>`contents = contents + line`<br>`while line != '':`<br>`  contents += myfile.readline()`<br>`myfile.close()` | When you want to process only part of a file |
| The for line in file approach | `myfile = open(filename, 'r')`<br>`contents = ''`<br>`for line in myfile:`<br>`  contents += line`<br>`myfile.close()` | When you want to process every line in the file one at a time |
| The readlines approach | `myfile = open(filename, 'r')`<br>`lines = myfile.readlines()`<br>`myfile.close()` | When you want to examine each line of a file by index |

# File I/O with Context Managers – the **`with`** statement

Every time we open a file we must also close it.

- Python allows us to use **context managers**, which will automatically close files when the context is exited.
  - We can create context managers using the **with** statement.

**`open()`** returns a **context manager** object

**Example:**

```
with open('test.txt', 'r') as file:
    contents = file.read()

>>> contents
    'Hello world!\nHello world again!'
```

- There's no need to call the `close()` method within the **`with`** statement.
- The file opened in the context manager used by the **`with`** statement will be closed automatically when the context is exited.

# File I/O – Working with CSV Files

CSV files are text files containing values separated by commas.

- To read the contents of a CSV file `'grades.csv'` using a context manager:

```
with open('grades.csv', 'r') as file:
    contents = file.read()

>>> contents
    'Name,Test1,Test2,Final\nJohn,100,50,
    29\n,Mark,76,32,33\nSam,25,75,95\n'
```

Sample Spreadsheet

| Name | Test1 | Test2 | Final |
|---|---|---|---|
| John | 100 | 50 | 29 |
| Mark | 76 | 32 | 33 |
| Sam | 25 | 75 | 95 |

NOTE: `contents` is a string containing all the data in the CSV file (the rows are separated by *new line* characters and the “columns” are separated by commas)

# File I/O – Working with CSV Files using the CSV `reader` Class of the **`csv` module**[1]

- The `csv` module provides functions that work with CSV files
- Method **`csv.reader()`** takes a csv file object as input and returns a csv **reader object** that holds the content of the file
  - The csv reader object can be **iterated through**

# File I/O – Working with CSV Files using the CSV reader Class of the CSV module[2]

Open the csv file and create a csv **reader** object

```
import csv

with open('grades.csv', 'r') as csvfile:
    grades_reader = csv.reader(csvfile)

    row_num = 1
    for row in grades_reader:
        print('Row #', row_num, ':', row)
        row_num += 1
```

Iterate through each row in the csv **reader** object

Display the content of the file to the standard output

Output

```
Row # 1 : ['Name', 'Test1', 'Test2', 'Final']
Row # 2 : ['John', '100', '50', '29']
Row # 3 : ['Mark', '76', '32', '33']
Row # 4 : ['Sam', '25', '75', '95']
```

APS106

UNIVERSITY OF TORONTO

# Practice Problems

if nothing else, write #cleancode

slido

# What does the following code output?

ⓘ Start presenting to display the poll results on this slide.

# Review Practice Problem 1

## Q1. What does the following code output?

```
def my_fun1 (x, key, value):
    x[key] = value
    return x

def my_fun2(x, key, value):
    x[key] = value

x = {}
my_fun1 (x, 1, 'hi')
my_fun2 (x, 2, 'bi')
print (x)
```

A. `{1: 'hi'}`
B. `{1: 'hi', 2: 'bi'}`
C. `{2: 'bi'}`
D. `{2: 'bi', 1: 'hi'}`
E. None of the above

# Review Practice Problem 1

Q1. What does the following code output?

```
def my_fun1 (x, key, value):
    x[key] = value
    return x

def my_fun2(x, key, value):
    x[key] = value

x = {}
my_fun1 (x, 1, 'hi')
my_fun2 (x, 2, 'bi')
print (x)
```

A. {1: 'hi'}
B. {1: 'hi', 2: 'bi'}
C. {2: 'bi'}
D. {2: 'bi', 1: 'hi'}
E. None of the above

slido

# What does the following code output?

ⓘ Start presenting to display the poll results on this slide.

# Review Practice Problem 2

Q2. What does the following code output?

```
def fun1 (lst1 = []):
    lst1.append(4)
    lst1 = lst1 + [5]

lst1 = [1,2,3]
fun1 (lst1)
print(lst1)
```

A. [1, 2, 3]
B. None
C. [1, 2, 3, 4, 5]
D. [1, 2, 3, 4]
E. Error

# Review Practice Problem 2

Q2. What does the following code output?

```
def fun1 (lst1 = []):
    lst1.append(4)
    lst1 = lst1 + [5]

lst1 = [1,2,3]
fun1 (lst1)
print(lst1)
```

A. [1, 2, 3]
B. None
C. [1, 2, 3, 4, 5]
D. [1, 2, 3, 4]
E. Error

# Review Practice Problem 3

Q3. What does the following code output?

```
def fun (tup, item = (4) ):
    tup += item
    return tup

tup = (1,2,3)
tup2 = fun (tup)
print (tup)
print (tup2)
```

A. (1, 2, 3)
(1, 2, 3, 4)

B. (1, 2, 3, 4)
(1, 2, 3, 4)

C. (1, 2, 3)
(1, 2, 3, 4) (4,)

D. None
(1, 2, 3, 4)

E. Error

slido

# What does the following code output? Note: ' | ' refers to a new line

ⓘ Start presenting to display the poll results on this slide.

# Review Practice Problem 3

Q3. What does the following code output?

```
def fun (tup, item = (4) ):
    tup += item
    return tup

tup = (1,2,3)
tup2 = fun (tup)
print (tup)
print (tup2)
```

A. (1, 2, 3)
(1, 2, 3, 4)

B. (1, 2, 3, 4)
(1, 2, 3, 4)

C. (1, 2, 3)
(1, 2, 3, 4) (4,)

D. None
(1, 2, 3, 4)

E. Error

# Coding Question 1

Write a Python function `count_vowels` to count the frequency of vowels in a file. `count_vowels takes` takes as input a file path and returns a dictionary with items vowel : vowel frequency. (The input file is provided on Tutorial Quercus page. )

Usage example:

```
>>> count_vowels(</path/to/file>)
    {'a': 29, 'e': 40, 'i': 40, 'o': 29, 'u': 4}
```

# Coding Question 2

Write a Python function `maximal_subsets()` that takes in a string representing the path to a csv file where each line is a set of numbers and returns a list with the maximal subsets in that file. A **maximal subset** is a set that is not a subset of another.

For example, when run on a file with content:

```
1, 2, 3
1, 2, 3
1, 2
2, 3
2, 4
```

**`maximal_subsets()`** returns

```
[{1, 2, 3}, {1, 2, 3}, {2, 4}]
```

# Coding Question 2: Step 1

The each line of the input is a set of numbers

Step 1: Convert the file into a list of sets.

For example:

```
3, 4, 6, 7, 9, 10
2, 4, 5, 6, 7, 10
9, 2, 10
2, 3, 7, 8, 9
8, 5, 6
2, 11
8
3, 4, 5, 7, 10
8, 11, 5
3, 4, 5, 8, 9
3, 7
8
8, 10
3, 5, 6, 8, 9
```

```
[{3, 4, 6, 7, 9, 10}, {2, 4, 5, 6, 7, 10}, {2, 9, 10},
... ]
```

# Coding Question 2: Step 2

Step 2: Given the list of sets, find the maximal subsets (i.e. remove all sets that are subsets of other sets in the list).

Example:

the maximal subsets in:

```
[{1, 2, 3}, {1, 2, 3}, {1, 2}, {2, 3}, {2, 4}]
```

are:

```
[{1, 2, 3}, {1, 2, 3}, {2, 4}]
```

slido

# Any Questions?

ⓘ Start presenting to display the poll results on this slide.