14 Aug 2019

ANL 251 Python Programming

L4:Functions, Methods and Packages (Supplementary readings)

Kevin Kam Fung Yuen, PhD

Senior Lecturer

School of Business

Singapore University of Social Sciences

kfyuen@suss.edu.sg,
kevinkf.yuen@gmail.com

Class Schedule

S/N	Course Code	Group	Delivery Style	Date	Day	From	To	Venue
1	ANL251	T06	SEMINAR	24-Jul-2019	Wed	07:00 PM	10:00 PM	HQ BLK B-LAB B.4.15
2	ANL251	T06	37	Wed	31/07/2019	HQ BLK C	SR C.6.09/C.6.10	SR C.6.09
3	ANL251	T06	37	Wed	07/08/2019	HQ BLK C	SR C.3.14	
4	ANL251	T06	37	Wed	14/08/2019	HQ BLK C	SR C.4.12/C.4.13	
5	ANL251	T06	37	Wed	21/08/2019	HQ BLK C	SR C.4.12/C.4.13	SR C.4.12
6	ANL251	T06	37	Wed	28/08/2019	HQ BLK C	SR C.6.09/C.6.10	SR C.6.09

Topics to be Covered	Learning Outcomes to be Achieved*	Summary and Discussion of Key Concepts, Theories, Principles.	Class Activities to Enhance Learning
Study Unit 4: Functions, Methods and Packages	1. Apply the Python built-in functions. 2. Compose and use user-defined functions 3. Use the Python built-in types and the associated methods 4. Explain the concepts of packages and modules, and how Python manages and imports packages/modules 5. Solve problems using appropriate Python standard libraries	9: Functions and methods 10: Packages and modules	Access e-learning material: Study Unit 4 and Textbook Exercises 15 ~ 21, and recommended online readings. Seminars: discussion and activities to reinforce students' understanding

Textbook

- Don't forget to refer the study guides and slides in Canvas
- Zed A. Shaw, Learn Python 3 **the Hard Way**: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code, Addison-Wesley Professional, July 2017
- https://learnpythonthehardway.org/python3/
- http://www.informit.com/promotions/book-registration-learn-python-3-the-hard-way-141409 (videos)

- Toby Donaldson, Starting out with Python, Third Edition, 2014

1. BUILT-IN FUNCTIONS

Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

https://docs.python.org/3/library/functions.html

Quiz 1

What does each of the following evaluate?

max(8, 30, 50, 20 + 32)
abs(-23.1)

```
>>> max(8, 30, 50, 20+32)
52
>>> abs(-23.1)
23.1
```

Discussion 1

Check the information of each built-in function. How many arguments each can take?

```
max(...)
    max(iterable, *[, default=obj, key=func]) -> value
    max(arg1, arg2, *args, *[, key=func]) -> value

    With a single iterable argument, return its biggest item. The
    default keyword-only argument specifies an object to return if
    the provided iterable is empty.
    With two or more arguments, return the largest argument.
```

```
pow(x, y, z=None, /)
    Equivalent to x**y (with two arguments) or x**y % z (with three arguments)

    Some types, such as ints, are able to use a more efficient algorithm when
    invoked using the three argument form.
```

```
round(...)
    round(number[, ndigits]) -> number

    Round a number to a given precision in decimal digits (default 0 digits).
    This returns an int when called with one argument, otherwise the
    same type as the number. ndigits may be negative.
```

Passing Arguments to Functions

- Argument: piece of data that is sent into a function
  - Function can use argument in calculations
  - When calling the function, the argument is placed in parentheses following the function name

Passing Arguments to Functions (cont'd.)

- Parameter variable: variable that is assigned the value of an argument when the function is called
  - The parameter and the argument reference the same value
  - General format:

```
def function_name(parameter):
```

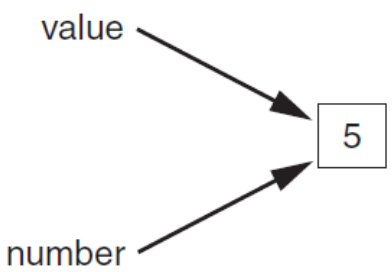
  - Scope of a parameter: the function in which the parameter is used

Passing Arguments to Functions

Figure 5-14 The `value` variable and the `number` parameter reference the same value

```
def main():
    value = 5
    show_double(value)


def show_double(number):
    result = number * 2
    print(result)
```



```
>>> max(8,30,50,20+32)
52
>>> max((8,30,50,20+32))
52
>>> max([8,30,50,20+32])
52
>>> max(8)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

```
>>> 2 ** 3
8
>>> 2 ** 3 % 5
3
>>> pow(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: pow expected at least 2 arguments, got 1
>>> pow(2, 3)
8
>>> pow(2, 3, 5)
3
>>> pow(2, 3, 5, 4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: pow expected at most 3 arguments, got 4
>>>
```

```
>>> round(2.1)
2
>>> round(2.4, 3.6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'float' object cannot be interpreted as an integer
>>> round(2.4, 3)
2.4
>>> round(2.43456234, 3)
2.435
>>> round([34.12345,2.43456234], 3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: type list doesn't define __round__ method
```

2. USER-DEFINED FUNCTIONS

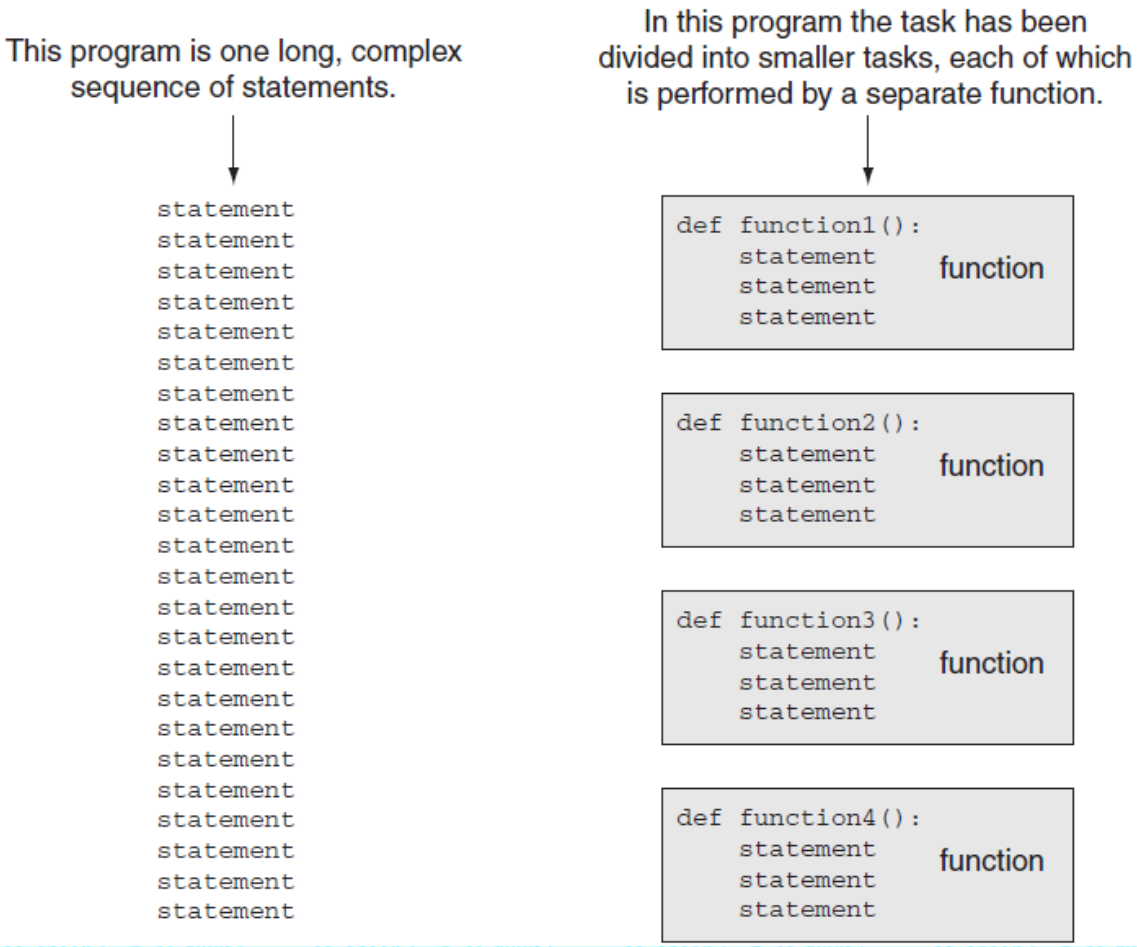
Function: group of statements within a program that perform as specific task

Usually one task of a large program

Functions can be executed in order to perform overall program task

Known as *divide and conquer* approach

Figure 5-1 Using functions to divide and conquer a large task



Benefits of Modularizing a Program with Functions

- The benefits of using functions include:
  - Simpler code
  - Code reuse
    - write the code once and call it multiple times
  - Better testing and debugging
    - Can test and debug each function individually
  - Faster development
  - Easier facilitation of teamwork
    - Different team members can write different functions

Function naming rules:

- Cannot use python key words as a function name, e.g. if, else, for , not, False, or, with, return. https://www.w3schools.com/python/python_ref_keywords.asp
- Cannot contain spaces
- First character must be a letter or underscore
- All other characters must be a letter, number or underscore
- Uppercase and lowercase characters are distinct

Defining and Calling a Function

- Function name should be descriptive of the task carried out by the function
  - Often includes a verb
- Function definition: specifies what function does

```
def function_name():
        statement
        statement
```

Defining and Calling a Function (cont'd.)

- Function header: first line of function
  - Includes keyword `def` and function name, followed by parentheses and colon
- Block: set of statements that belong together as a group
  - Example: the statements included in a function

Defining and Calling a Function (cont'd.)

- Call a function to execute it
  - When a function is called:
    - Interpreter jumps to the function and executes statements in the block
    - Interpreter jumps back to part of program that called the function
      - Known as function return

Indentation in Python

- Each block must be indented
  - Lines in block must begin with the same number of spaces
    - Use tabs or spaces to indent lines in a block, but not both as this can confuse the Python interpreter
    - IDLE automatically indents the lines in a block
  - Blank lines that appear in a block are ignored

Quiz 2

After the code below has been executed, what value does the variable result refer to?

```
def increment(x):
        return x + 1

result = increment(5)
```

```
3   def increment(x):
4       return x + 1
5
6   result = increment(5)
7   print(result)
```

```
6
```

```
>>> def increment(x):
...     return x + 1
...
>>> result = increment(5)
>>> print(result)
6
```

What if print(x) ?

Local Variables

- Local variable: variable that is assigned a value inside a function
  - Belongs to the function in which it was created
    - Only statements inside that function can access it, error will occur if another function tries to access the variable
- Scope: the part of a program in which a variable may be accessed
  - For local variable: function in which created

Local Variables (cont'd.)

- Local variable cannot be accessed by statements inside its function which precede its creation
- Different functions may have local variables with the same name
  - Each function does not see the other function's local variables, so no confusion

Discussion 2

- Define a function double that returns two times the number it is passed.
- Define a function area that returns a triangle's area from given base and height.
- One triangle has a base of length 3.8 and a height of length 7.0. A second triangle has a base of length 3.5 and a height of length 6.8. Calculate which of two triangles' areas is bigger.

```
3   # function double
4   def double(num):
5       return 2 * num
6
7   print("double(6): ", double(6))
8
9   # function area
10  def area(base, height):
11      return base * height / 2
12
13  triangle1 = area(3.8, 7.0)
14  triangle2 = area(3.5, 6.8)
15  print(f"triangle1: {triangle1},triangle2: {triangle2}")
```

```
double(6):  12
triangle1: 13.299999999999999,triangle2: 11.9
```

Discussion 3

What is the outcome of executing the code below? After the code below has been executed, what value does the variable result refer to?

```
def add(number1, number2):
    print(number1 + number2)

result = add(1, 3)
```

```
3  def add(number1, number2):
4      print(number1 + number2)
5
6  result = add(1, 3)
7
8  print("what is result? > ", result)
9
```

```
4
what is result? >  None
```

Void Functions and Value-Returning Functions

- A void function:
  - Simply executes the statements it contains and then terminates.
- A value-returning function:
  - Executes the statements it contains, and then it returns a value back to the statement that called it.
    - The `input`, `int`, and `float` functions are examples of value-returning functions.

Discussion 4

What is printed by the code below?

```
def add(number1, number2):
    return number1 + number2
    print("hello")

result = add(1, 3)
```

```
3   def add(number1, number2):
4       return number1 + number2
5       print("hello")
6
7   result = add(1, 3)
8   # add one line
9   print("what is result? > ", result)
10
```

```
PS D:\_0SUSS\ANL251Python\MyCode\L4> python L2D4.py
what is result? >  4
```

Discussion 5

Observe the description of the below two built-in functions. Add description for your user-defined function count_vowels, which is to count the vowels in a given string.

def count_vowels(word):

```
pow(x, y, z=None, /)
    Equivalent to x**y (with two arguments) or x**y % z (with three arguments)

    Some types, such as ints, are able to use a more efficient algorithm when
    invoked using the three argument form.
```

```
round(...)
    round(number[, ndigits]) -> number

    Round a number to a given precision in decimal digits (default 0 digits).
    This returns an int when called with one argument, otherwise the
    same type as the number. ndigits may be negative.
```

```
5   def count_vowels(words):
6       """Count the vowels in a given string.
7
8       Parameters:
9       word (str): input the the words in string
10
11      return:
12      int: the number of vowels
13
14      """
15
16      # initial the value of number of vowels
17      num_vowels = 0
18
19      # count the number of vowels
20      for char in words:
21          if char in 'aeiou':
22              num_vowels = num_vowels + 1
23
24      return num_vowels
```

```
PS D:\_0SUSS\ANL251Python\MyCode\L4> python L4D5.py
show .__doc__
Count the vowels in a given string.

    Parameters:
    word (str): input the the words in string

    return:
    int: the number of vowels


--------------------
show help()
Help on function count_vowels in module __main__:

count_vowels(words)
    Count the vowels in a given string.

    Parameters:
    word (str): input the the words in string

    return:
    int: the number of vowels

--------------------
type(words_count) <class 'int'>
The the number of vowels for count the vowels in a given string is 10
```

```
27  # print description
28  print("show .__doc__")
29  print(count_vowels.__doc__)
30  print("-" * 20)
31  print("show help()")
32  help(count_vowels)
33
34  # test by function call
35  print("-" * 20)
36  words = "count the vowels in a given string"
37  words_count = count_vowels(words)
38  print("type(words_count)",type(words_count))
39  print("The the number of vowels for", words,"is", words_count)
40
```

3. BUILT-IN TYPES AND THE METHODS

Discussion 6

methods of str obects

https://docs.python.org/3/library/stdtypes.html#string-methods

Given jams = " Jam tomorrow and jam yesterday - but never jam today."

- write an expression that produces a new string which removes the leading whitespaces in the string that jams refers to.
- write an expression that produces the index of 'tomorrow' in the string that jams refers to.
- write an expression that produces the number of occurrences of "jam" *ignoring letter case* in the string that jams refers to.

```
>>> jams = "   Jam tomorrow and jam yesterday - but never jam today."
>>>
>>> print("before: ", jams)
before:     Jam tomorrow and jam yesterday - but never jam today.
>>> jams.strip(" ")
'Jam tomorrow and jam yesterday - but never jam today.'
>>> print("after:", jams)
after:    Jam tomorrow and jam yesterday - but never jam today.
>>>
>>> jams.find("tomorrow")
7
>>> jams.find("tomorow")
-1
>>>
>>> jams.count("jam")
2
>>> jams.lower()
'   jam tomorrow and jam yesterday - but never jam today.'
>>> jams.lower().count("jam")
3
>>>
```

Discussion 7

access_log

```
64.242.88.10 - - [07/Mar/2004:16:05:49 -0800] "GET /twiki/bin/edit/Main/Double_bounce_sender?topicparent=Main.ConfigurationVaria
64.242.88.10 - - [07/Mar/2004:16:06:51 -0800] "GET /twiki/bin/rdiff/TWiki/NewUserTemplate?rev1=1.3&rev2=1.2 HTTP/1.1" 200 4523
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision HTTP/1.1" 200 6291
64.242.88.10 - - [07/Mar/2004:16:11:58 -0800] "GET /twiki/bin/view/TWiki/WikiSyntax HTTP/1.1" 200 7352
64.242.88.10 - - [07/Mar/2004:16:20:55 -0800] "GET /twiki/bin/view/Main/DCCAndPostFix HTTP/1.1" 200 5253
64.242.88.10 - - [07/Mar/2004:16:23:12 -0800] "GET /twiki/bin/oops/TWiki/AppendixFileSystem?template=oopsmore&param1=1.12&param2
64.242.88.10 - - [07/Mar/2004:16:24:16 -0800] "GET /twiki/bin/view/Main/PeterThoeny HTTP/1.1" 200 4924
64.242.88.10 - - [07/Mar/2004:16:29:16 -0800] "GET /twiki/bin/edit/Main/Header_checks?topicparent=Main.ConfigurationVariables HT
64.242.88.10 - - [07/Mar/2004:16:30:29 -0800] "GET /twiki/bin/attach/Main/OfficeLocations HTTP/1.1" 401 12851
64.242.88.10 - - [07/Mar/2004:16:31:48 -0800] "GET /twiki/bin/view/TWiki/WebTopicEditTemplate HTTP/1.1" 200 3732
64.242.88.10 - - [07/Mar/2004:16:32:50 -0800] "GET /twiki/bin/view/Main/WebChanges HTTP/1.1" 200 40520
```

Figure 4.3 A sample of Apache web log

(Source: http://www.monitorware.com/en/logsamples/apache.php)

methods of file objects

https://docs.python.org/3/tutorial/inputoutput.html#methods-of-file-objects

Refer to the text file in Figure 4.3. After executing the code below, what value does log_line[3] refer to?

```
log_file = open("access_log", "r")
log_line = log_file.read()
```

```
3   log_file = open("access_log", "r")
4   log_line = log_file.read()
5
6   print(log_line[0])
7   print(log_line[1])
8   print(log_line[2])
9   print(log_line[3])
10  print(type(log_line))
11  print(type(log_line[3]))
```

```
6
4
.
2
<class 'str'>
<class 'str'>
```

access_log

```
64.242.88.10 - - [07/Mar/2004:16:05:49 -0800] "GET /twiki/bin/edit/Main/Double_bounce_sender?topicparent=Main.ConfigurationVaria
64.242.88.10 - - [07/Mar/2004:16:06:51 -0800] "GET /twiki/bin/rdiff/TWiki/NewUserTemplate?rev1=1.3&rev2=1.2 HTTP/1.1" 200 4523
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision HTTP/1.1" 200 6291
64.242.88.10 - - [07/Mar/2004:16:11:58 -0800] "GET /twiki/bin/view/TWiki/WikiSyntax HTTP/1.1" 200 7352
64.242.88.10 - - [07/Mar/2004:16:20:55 -0800] "GET /twiki/bin/view/Main/DCCAndPostFix HTTP/1.1" 200 5253
64.242.88.10 - - [07/Mar/2004:16:23:12 -0800] "GET /twiki/bin/oops/TWiki/AppendixFileSystem?template=oopsmore&param1=1.12&param2
64.242.88.10 - - [07/Mar/2004:16:24:16 -0800] "GET /twiki/bin/view/Main/PeterThoeny HTTP/1.1" 200 4924
64.242.88.10 - - [07/Mar/2004:16:29:16 -0800] "GET /twiki/bin/edit/Main/Header_checks?topicparent=Main.ConfigurationVariables HT
64.242.88.10 - - [07/Mar/2004:16:30:29 -0800] "GET /twiki/bin/attach/Main/OfficeLocations HTTP/1.1" 401 12851
64.242.88.10 - - [07/Mar/2004:16:31:48 -0800] "GET /twiki/bin/view/TWiki/WebTopicEditTemplate HTTP/1.1" 200 3732
64.242.88.10 - - [07/Mar/2004:16:32:50 -0800] "GET /twiki/bin/view/Main/WebChanges HTTP/1.1" 200 40520
```

Figure 4.3 A sample of Apache web log

(Source: http://www.monitorware.com/en/logsamples/apache.php)

Discussion 8

methods of file objects (https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files)

classlist.txt is a text file with 50 lines, each one containing one student name.

- Write code to print all the student names in order, sorted alphabetically.
- Write code to print student names until the first name starting with X appears. If no student name starts with X, all student names will be printed.
- There are two new students: Jack Chen, Mike Tan. Add them to the end of classlist.txt

```
6   file = open("classlist.txt", "r")
7   student_list = file.readlines()
8   sorted_student_list = sorted(student_list)
9
10  # ASCII charater issue, Â,in the file,  but we use UTF-8 format
11  for item in sorted_student_list:
12      # print(item) # compare
13      print(item.replace('Â', ''))
14
```

```
2  file = open("classlist.txt", "r")
3
4  student_list = file.readlines()
5  #student_list = sorted(student_list) # optional
6
7  for item in student_list:
8      if not item.startswith("X"):
9          # print(item) # compare
10         print(item.replace('Â', ''))
11     else:
12         break
13
```

```
1   # "w": open a new file to write
2   # "a": open a file to append
3   file = open("classlist1.txt", "a")
4   new_students = ["jack Chen", "Mike Tan"]
5
6   for student in new_students:
7       file.write(student)
8       file.write("\n")
9
10  file.close()
11
```

4. MANAGING AND IMPORTING PACKAGES / MODULES

Storing Functions in Modules

- In large, complex programs, it is important to keep code organized
- Modularization: grouping related functions in modules
  - Makes program easier to understand, test, and maintain
  - Make it easier to reuse code for multiple different programs
    - Import the module containing the required function to each program that needs it

Storing Functions in Modules

- Module is a file that contains Python code
  - Contains function definition but does not contain calls to the functions
    - Importing programs will call the functions
- Rules for module names:
  - File name should end in `.py`
  - Cannot be the same as a Python keyword
- Import module using `import` statement

Discussion 9

Call and test your user-defined functions in Python interpreter.

1. Save the two function definitions in a .py file.

```
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b
```

2. Start your Python interpreter from the same directory where you saved the .py file.
3. How to import the function definitions into the Python interpreter?

```
# L4D9a.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a -b
```

```
#L4D9b.py for calling functions in L4D9a.py
import L4D9a

result1 = L4D9a.subtract(3, 4)
print(result1)

result2 = L4D9a.add(4, 5)
print(result2)

```

```
PS D:\_0SUSS\ANL251Python\MyCode\L4> python L4D9b.py
-1
9
```

5. PYTHON STANDARD LIBRARIES

Discussion 10

```
>>> import datetime
>>> now = datetime.date.today()
>>> print(now.strftime("Today is %d %b %Y, %A."))
Today is 21 Mar 2018, Wednesday.
>>> birthday = datetime.date(1964, 7, 31)
>>> age = now - birthday
>>> print(f"You are {age.days//365} years old.")
You are 53 years old.
```

Figure 4.5 Using the standard library datetime

Refer to the code in Figure 4.5

- Will it work if we change the second line to now = date.today()? Why?
- What other change(s) must be done to make the changed program work?
- How to make the third line of code print in a format as "Today is March 21 2018, Wed"?

https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior

```
>>> import datetime
>>> now = date.today()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'date' is not defined
>>>
```

Table 4.1 The meaning of formatting directives used in Figure 4.5

Directive	Meaning	Example
%d	Day of the month as a zero-padded decimal number.	01, 02, …, 31
%b	Month as locale's abbreviated name.	an, Feb, …, Dec (en_US)
%Y	Year with century as a decimal number.	0001, 0002, …, 2013, 2014, …, 9998, 9999
%A	Weekday as locale's full name.	Sunday, Monday, …, Saturday (en_US)

```
>>> import datetime
>>> now = datetime.date.today()
>>> print(now)
2019-08-14
>>> print(now.strftime("today is %d %b %y, %A."))
today is 14 Aug 19, Wednesday.
>>>
>>> birthday = datetime.date(1964, 7, 31)
>>> age = now - birthday
>>> print(f"You are {age.days//365} years old.")
You are 55 years old.
>>>
```

```
>>> import datetime
>>> now = date.today()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'date' is not defined
>>>
```

```python
# date.py
import datetime

def today():
    return datetime.date.today()

```

Another python file

```python
# L4D10b.py
import date
now = date.today()
print(now)
print(now.strftime("Today is %d %b %y, %A."))
print("new format")
print(now.strftime("Today is %B %d %Y, %a"))
```

```
PS D:\_0SUSS\ANL251Python\MyCode\L4> python L4D10b.py
2019-08-14
Today is 14 Aug 19, Wednesday.
new format
Today is August 14 2019, Wed
```

Discussion 11

The standard library **math** https://docs.python.org/3/library/math.html

Define a function area_heron that returns a triangle's area given the lengths of 3 sides using Heron's formula. Note your program needs to check whether the given 3 lengths are able to form a proper triangle.

https://en.wikipedia.org/wiki/Heron%27s_formula

Heron's formula states that the area of a triangle whose sides have lengths a, b, and c is

$$A = \sqrt{s(s-a)(s-b)(s-c)},$$

where s is the semi-perimeter of the triangle; that is,

$$s = \frac{a+b+c}{2}.$$ [2]

Let $\triangle ABC$ be the triangle with sides $a = 4$, $b = 13$ and $c = 15$. The semiperimeter is $s = \frac{1}{2}(a+b+c) = \frac{1}{2}(4+13+15) = 16$, and the area is

$$A = \sqrt{s\,(s-a)\,(s-b)\,(s-c)} = \sqrt{16 \cdot (16-4) \cdot (16-13) \cdot (16-15)}$$
$$= \sqrt{16 \cdot 12 \cdot 3 \cdot 1} = \sqrt{576} = 24.$$

In this example, the side lengths and area are all integers, making it a Heronian triangle. However, Heron's formula works equally well in cases where one or all of these numbers is not an integer.

```python
import math

def Heron_triagule_area(a, b, c):
    s = (a + b + c) / 2
    A_square = s * (s - a) * (s - b) * (s - c)
    A = math.sqrt(A_square)
    return A

test = Heron_triagule_area(4, 13, 15)
print("The expected result should be 24: ", test)
```

```
PS D:\_0SUSS\ANL251Python\MyCode\L4> python L4D11a.py
The expected result should be 24:  24.0
```

```
>>> def Heron_triagule_area(a, b, c):
...     s = (a + b + c) / 2
...     A_square = s * (s - a) * (s - b) * (s - c)
...     A = math.sqrt(A_square)
...     return A
...
>>> Heron_triagule_area(1, 1, 10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in Heron_triagule_area
ValueError: math domain error
>>>
```

These inputs do not make sense

$$A^2 > 0$$

```python
# L4D11b.py
import math

def Heron_triagule_area(a, b, c):
    try:
        s = (a + b + c) / 2
        A_square = s * (s - a) * (s - b) * (s - c)
        A = math.sqrt(A_square)
        return A
    except ValueError:
        print("it is not a triangle.")

test = Heron_triagule_area(4, 13, 15)
print("The expected result should be 24: ", test)

print("error expected: ", Heron_triagule_area(1, 1, 15))
```

```
The expected result should be 24:  24.0
it is not a triangle.
error expected:  None
```

THANK YOU!