ANL251 Python Programming

Study Unit 4

Functions, Methods and Packages

Learning Outcomes and Learning Resources

1. Apply the Python built-in functions.
   - SU4 Chapter 1.1
   - https://docs.python.org/3/library/functions.html
2. Compose and use user-defined functions
   - SU4 Chapter 1.2
   - Textbook Videos and Exercises 18, 19 and 21
3. Use the Python built-in types and the associated methods
   - SU4 Chapter 2
   - Textbook Videos and Exercises 15 ~ 17
   - https://docs.python.org/3/library/stdtypes.html#string-methods

Learning Outcomes and Learning Resources

4. Explain the concepts of packages and modules, and how Python manages and imports packages/modules
   - SU4 Chapter 3
5. Solve problems using appropriate Python standard libraries
   - SU4 Chapter 3.1
   - https://docs.python.org/3/library/
   - https://docs.python.org/3/library/datetime.html#module-datetime

Seminars: discussion and activities to reinforce students' understanding

1. Built-in Functions

Discussion

built-in functions (SU4 Chapter 1.1, https://docs.python.org/3/library/functions.html)

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()	

Quiz

What does each of the following evaluate?

max(8, 30, 50, 20 + 32)

abs(-23.1)

Discussion

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.
```

2. User-defined Functions

Recap

User-defined functions (SU4 Chapter 1.2, Textbook Videos and Exercises 18, 19 and 21)

```
def cheese_and_crackers(cheese_count, boxes_of_crackers):
    print(f"You have {cheese_count} cheeses!")
    print(f"You have {boxes_of_crackers} boxes of crackers!")

print("Listing the stock")

print("We can just give the function numbers directly:")
cheese_and_crackers(20, 30)

print("OR, we can use variables from our script:")
amount_of_cheese = 20
amount_of_crackers = 30
cheese_and_crackers(amount_of_cheese, amount_of_crackers)

print("\nUpdating the stock")

print("We can do math inside directly:")
cheese_and_crackers(amount_of_cheese + 10, amount_of_crackers - 15)

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

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

print("OR, we can just give the variables updated by the function:")
amount_of_cheese_updated = add(amount_of_cheese, 10)
amount_of_crackers_updated = subtract(amount_of_crackers, 15)
cheese_and_crackers(amount_of_cheese_updated, amount_of_crackers_updated)

print("OR, we can call functions inside another function directly:")
cheese_and_crackers(add(amount_of_cheese, 10), subtract(amount_of_crackers, 15))
```

Figure 4.6 Defining and calling user-defined functions

Recap

Note on the code in Figure 4.6

- After the colon, all the lines that are indented four spaces will become attached to the defined function.
- When a return statement executes, it passes a value back to the caller and exits the function.
- The naming convention of a function is the same as variable names (in SU1).
- The variables inside the function definition (e.g. *cheese_count*, *boxes_of_crackers*, *a* and *b*) are called local variables. They are temporary variables made just for the function's run. When the function exits, these temporary variables go away.

Quiz

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

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

result = increment(5)
```

Discussion

- 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.

Discussion

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)
```

Discussion

What is printed by the code below?

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

result = add(1, 3)
```

Discussion

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.
```

3. Built-in Types and the Methods

Recap

Built-in types and the methods

Methods of **str** objects (SU4 Chapter 2.1, https://docs.python.org/3/library/stdtypes.html#string-methods)

```
>>> 'a,b,c'.split(',')
['a', 'b', 'c']
>>> 'a,b,c'.split(',', maxsplit=1)
['a', 'b,c']
>>> 'a,,b,c,'.split(',')
['a', '', 'b', 'c', '']
>>> 'a b c'.split()
['a', 'b', 'c']
>>> ' a b  c   '.split()
['a', 'b', 'c']
```

Figure 4.2 Using the string method split()

Recap

Note:

- When we need to do a rather standard task in Python, **built-in functions** are one way we may look for.
- But some basic tasks instead are implemented as **objects**' **methods**.
  - Variables in various data types or data structures are so-called Python **objects**.
  - Python objects come with their so-called "**methods**", functions that belong to Python objects.
  - Each object has specific methods associated, depending on the type of the object.

Discussion

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.

Recap

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 (SU4 Chapter 2.2, Textbook Videos and Exercises 15 ~ 17)

```
from_file = input("Type the filename of the full access log: ")
to_file = input("Type the filename to store the 401 error log: ")
in_file = open(from_file)
out_file = open(to_file, 'w')

for line in in_file:
    if line.find(' 401 ') != -1:
        out_file.write(line)

print("Checking 401 error done.")
out_file.close()
in_file.close()
```

Figure 4.4 Reading and writing text files

Discussion

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()
```

Discussion

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

4. Managing and Importing Packages / Modules

Recap

Managing and importing packages/modules (SU4 Chapter 3)

Note:

– Each Python script is a so-called **module**. These modules specify functions, methods and new Python types for solving particular tasks.

– To help organise modules and provide a naming hierarchy, Python has a concept of **packages**.

- You can think of package as a directory of Python scripts.
- Like file system directories, packages are organised hierarchically, and packages may themselves contain sub-packages, as well as regular modules.

Recap

Importing packages/modules (SU4 Chapter 3.1)

Note:

Most Python functions are not immediately available as built-ins. They must be imported before usage. There are two ways of importing a module

– When using syntax like

```
import sound.effects.echo
```

Anything inside must be referenced with its full name.

```
sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
```

– When using syntax like

```
from sound.effects.echo import echofilter
```

Anything inside can be used as

```
echofilter(input, output, delay=0.7, atten=4)
```

Discussion

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?

Recap

Managing packages/modules (SU4 Chapter 3.2)

Note:

- There are many Python packages available from the internet but not installed along with the Python environment. To use those Python packages, you'll first have to **install** them on your system using **pip3**(Mac OS) or **pip**(Windows).
- Then you will be able to **import** them in the same as the standard library.

5. Python Standard Libraries

Recap

Python standard library (SU4 Chapter 3.1, https://docs.python.org/3/library/)

The standard library **datetime** (https://docs.python.org/3/library/datetime.html#module-datetime)

```
>>> 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

Discussion

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)

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)

Discussion

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.