

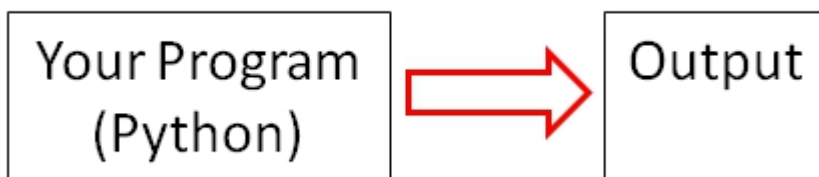
Title: ตำราวิชา Problem Solving and Computer Programming (PSIT) - PSIT Book **Author:** รศ.ดร. โชติพัทธ์ ภรณ์วลัย **Rights:** Copyright 2022 - ใช้เป็นการภายในคณะเทคโนโลยีสารสนเทศ สจล. เท่านั้น **Language:** th-TH **Date:** 9 สิงหาคม 2565

Chapter 1: Simple Programs

A Simple Program (Entryway)

เราใช้งานคอมพิวเตอร์เพื่อให้คอมพิวเตอร์ทำงานในสิ่งที่เราต้องการ สิ่งแรกที่เกิดขึ้นคือ เราต้องรู้ความต้องการของเรา ก่อนว่าต้องการให้คอมพิวเตอร์ทำอะไร สิ่งที่เราต้องทำต่อคือ เราต้องบอกคอมพิวเตอร์ให้ทำงานตามที่เราต้องการ การจะบอกคอมพิวเตอร์ ถือเป็นการสื่อสารอย่างหนึ่ง เช่นเดียวกับเมื่อเราต้องการให้เพื่อนเราช่วยทำงานตามที่เราต้องการ เมื่อมีการสื่อสาร ก็ต้องมีภาษาที่ใช้ในการสื่อสาร ในกรณีสื่อสารกับคอมพิวเตอร์ เราจำเป็นที่จะต้อง ใช้ภาษาที่คอมพิวเตอร์เข้าใจ ในเอกสารการสอนนี้ เราจะใช้ภาษา Python เขียนอธิบายความต้องการของเราลงไป เพื่อให้คอมพิวเตอร์ทำงาน สิ่งที่เราเขียนลงไปทั้งหมดเพื่อให้คอมพิวเตอร์ทำงานตามที่เราได้ระบุไว้ เรียกโดยรวมว่า โปรแกรม

ตัวอย่าง โปรแกรมที่ง่ายที่สุด คือการเขียน โปรแกรมเพื่อสั่งให้คอมพิวเตอร์แสดงผลข้อความตามที่ได้กำหนดไว้ บนหน้าจอคอมพิวเตอร์ ในตัวอย่างนี้ เราจะเขียนโปรแกรมให้แสดงผลคำว่า "Output" บนหน้าจอคอมพิวเตอร์



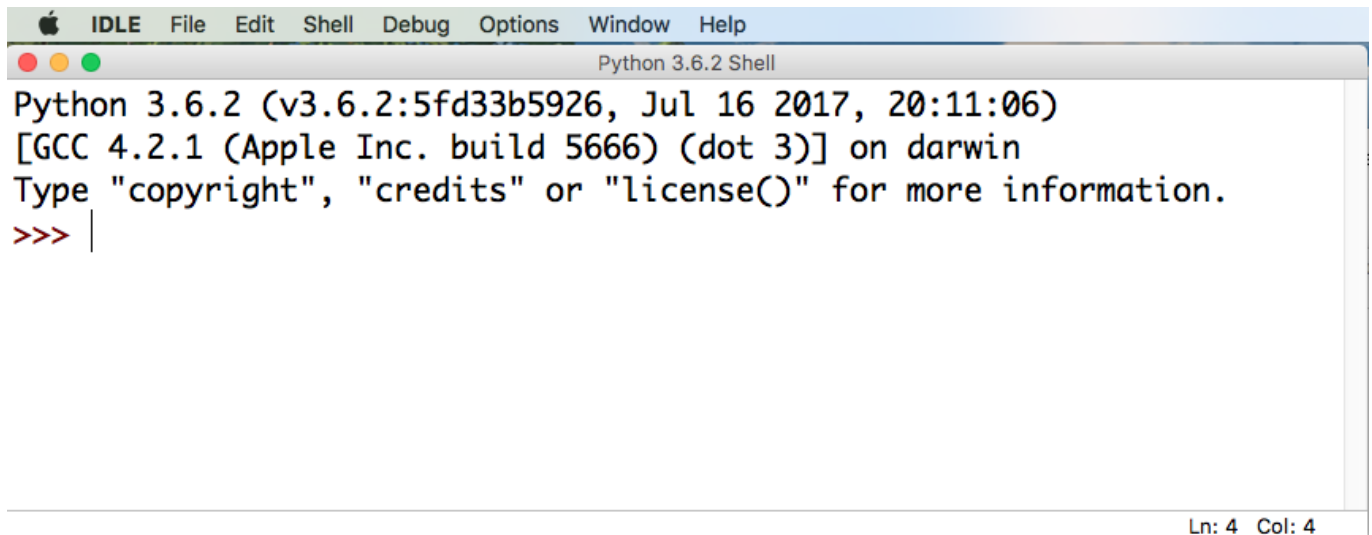
หมายเหตุ ภาษา Python ในปัจจุบันมี 2 version หลักๆ คือ version 2 และ version 3 ในหนังสือเล่มนี้ เราจะใช้ Python version 3

การสั่งให้คอมพิวเตอร์ทำงานด้วยภาษา python สามารถทำได้ 2 วิธีหรือ 2 mode คือ

1. interactive mode หรือ โหมดโต้ตอบ
2. script mode หรือ โหมดสคริปต์

Interactive Mode

เราจะมาทดลองเขียนโปรแกรมด้วยโหมด interactive กันก่อน ให้เปิดโปรแกรม Idle จะได้หน้าต่างดังรูป



```
Python 3.6.2 (v3.6.2:5fd33b5926, Jul 16 2017, 20:11:06)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> |
```

Ln: 4 Col: 4

หมายเหตุ รูปตัวอย่างให้หมดในบทนี้ จะเป็นรูปแบบบนระบบปฏิบัติการ macOS สำหรับระบบปฏิบัติการอื่นๆเช่น Windows หรือ Linux ก็จะได้รูปและผลลัพธ์ที่คล้ายกัน

สังเกตบรรทัดแรกจะเห็นว่าขณะนี้เรากำลังใช้งานโปรแกรมภาษา Python เวอร์ชัน 3.6.2 อยู่ บรรทัดสุดท้ายแสดงเครื่องหมาย >>> เรียกว่า python prompt เป็นการบอกว่ากำลังรอรับคำสั่งแบบโต้ตอบกับผู้ใช้ให้ทดลองพิมพ์ข้อความต่อไปนี้ลงไป แล้วกด Enter เพื่อเป็นการสั่งให้คอมพิวเตอร์เริ่มทำงาน

```
print("Output")
```

หมายเหตุ `print` เป็นชื่อของฟังก์ชัน (function) ใช้ในการแสดงผลข้อความออกทางหน้าจอ

>>' is shown again." data-bbox="68 527 932 751"/>

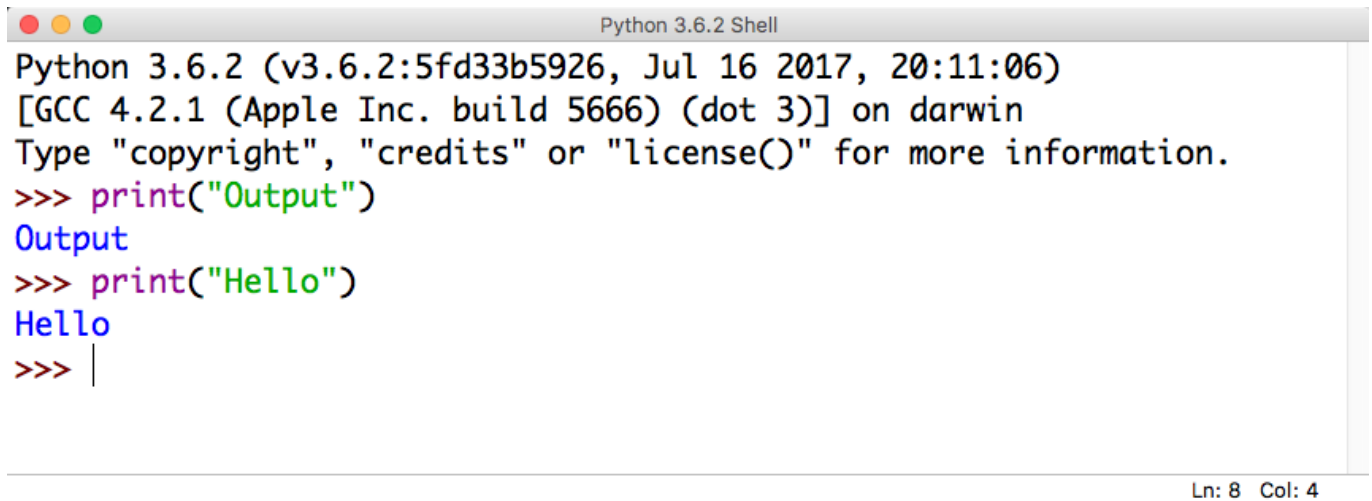
```
Python 3.6.2 (v3.6.2:5fd33b5926, Jul 16 2017, 20:11:06)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print("Output")
Output
>>>
```

Ln: 6 Col: 4

ถ้าเราเขียนโปรแกรมได้ถูกต้อง คอมพิวเตอร์ก็จะเข้าใจ และสามารถแสดงผลออกมา เมื่อแสดงผลคำว่า "Output" บนหน้าจอแล้ว ก็จะแสดงเครื่องหมาย python prompt หรือ >>> เพื่อรอรับคำสั่งจากผู้ใช้ต่อไป

จะสังเกตเห็นได้ว่า การทำงานของคอมพิวเตอร์ในโหมด Interactive นี้รับคำสั่งจากผู้ใช้ทีละคำสั่ง เมื่อทำงานเสร็จก็จะรอรับคำสั่งถัดไปจากผู้ใช้ ซึ่งในตัวอย่างด้านล่าง ผู้ใช้ต้องการให้คอมพิวเตอร์แสดงข้อความว่า "Hello" จึงส่งคำสั่งดังกล่าวเข้าไปอีกครั้ง

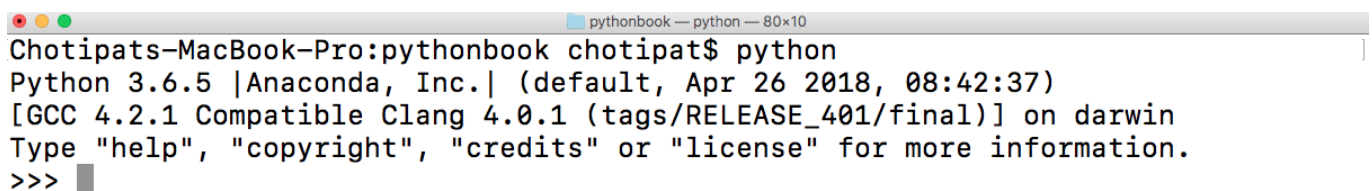
```
print("Hello")
```



```
Python 3.6.2 Shell
Python 3.6.2 (v3.6.2:5fd33b5926, Jul 16 2017, 20:11:06)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print("Output")
Output
>>> print("Hello")
Hello
>>> |
```

Ln: 8 Col: 4

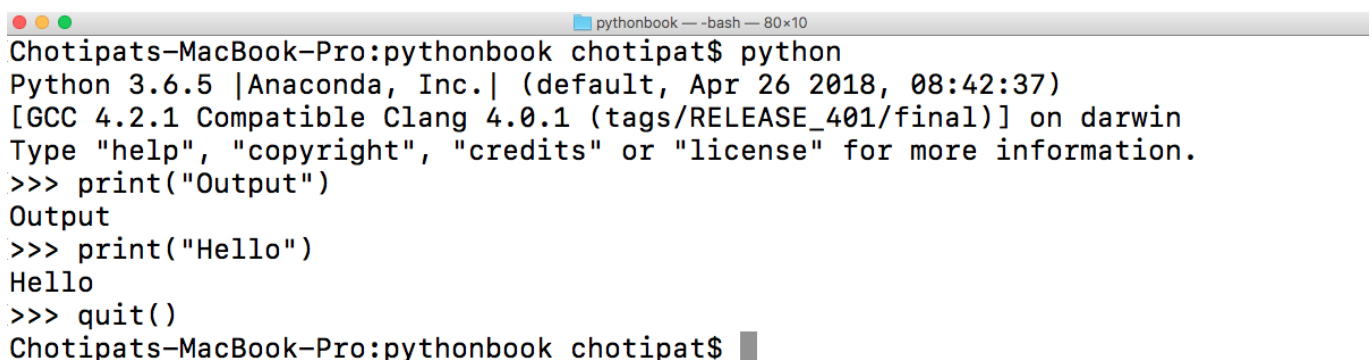
ในการทำงานแบบ Interactive mode นั้นสามารถเรียกทำงานได้อีกทางผ่านทาง Command Prompt (ของ Windows) หรือ terminal (ของ Mac) ในตัวอย่างนี้ ให้พิมพ์คำว่า `python`



```
pythonbook — python — 80x10
Chotipats-MacBook-Pro:pythonbook chotipat$ python
Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

เราสามารถสั่งคำสั่งแบบ Interactive ได้เหมือนที่ทำบนโปรแกรม Idle

ถ้าต้องการออกจากโปรแกรม python ให้พิมพ์คำว่า `quit()`



```
pythonbook — -bash — 80x10
Chotipats-MacBook-Pro:pythonbook chotipat$ python
Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Output")
Output
>>> print("Hello")
Hello
>>> quit()
Chotipats-MacBook-Pro:pythonbook chotipat$ █
```

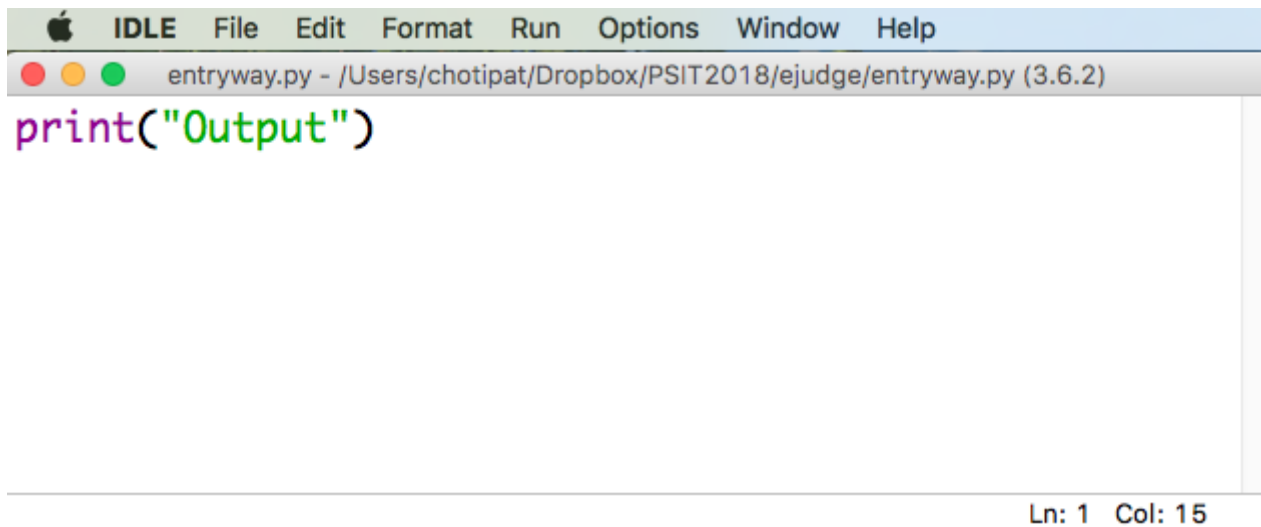
ในรูปข้างบนนี้ จะสังเกตเห็นว่า version ของ python เป็น 3.6.5 แต่บน idle เป็น 3.6.2 ที่แตกต่างกันเป็นเพราะว่าในเครื่องที่ใช้อยู่มีการติดตั้ง โปรแกรม python 3.6.2 และโปรแกรม anaconda ซึ่งได้รวม python 3.6.5 เข้ามาด้วย ดังนั้นในเครื่องตอนนี้จะมี python ติดตั้งอยู่ทั้ง 2 version

Script Mode

ใน Script mode เราจะเขียนชุดคำสั่ง หรือหลายๆคำสั่งเข้าไว้ด้วยกันในไฟล์ แล้วจึงส่งชุดคำสั่งนั้นให้ python ทำการประมวลและทำงาน การทำงานก็จะทำงานในชุดคำสั่ง ในตามลำดับที่ได้เขียนไว้

ทดลองเขียนโปรแกรมใน script mode ด้วยการ click ที่ **File -> New File** ในโปรแกรม Idle ตัวโปรแกรมจะเปิดหน้าต่างใหม่ขึ้นมา เราสามารถเขียนโปรแกรม เช่นการพิมพ์ `print("Output")` ลงไปในหน้าต่างนี้ และทำการ save

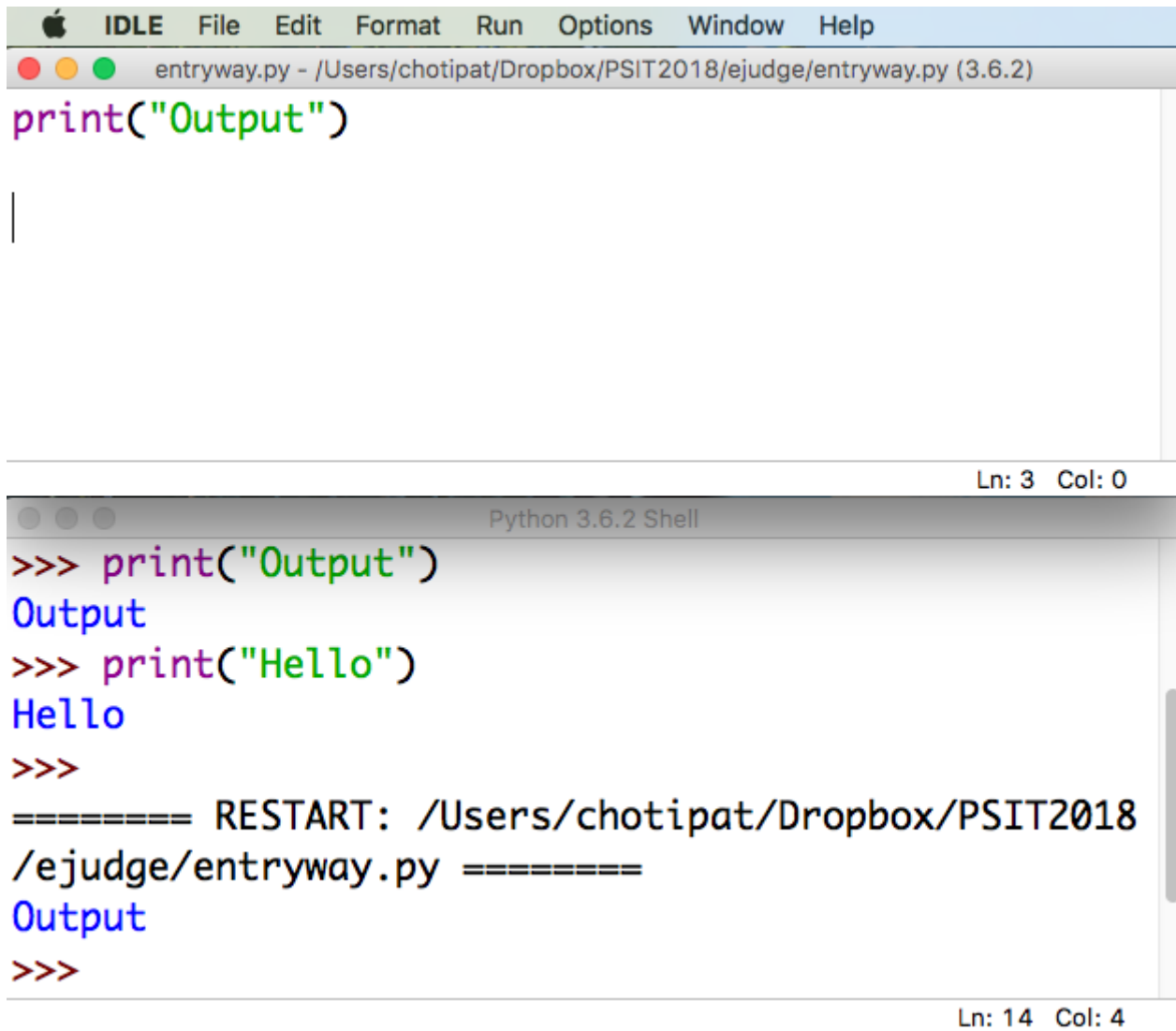
ไฟล์ที่เขียนด้วยการ click **File** -> **Save** หรืออาจจะกดปุ่ม **Command S** (macOS) หรือ **Control S** (Windows และ Linux) แล้วแต่ระบบปฏิบัติการที่ใช้งานอยู่ ในที่นี้เราจะตั้งชื่อไฟล์ว่า entryway.py (โปรแกรมภาษา Python จะมีการใช้ File extension เป็น py เสมอ)



The screenshot shows the Python IDLE environment. The menu bar at the top includes Apple, IDLE, File, Edit, Format, Run, Options, Window, and Help. The title bar indicates the file is 'entryway.py' located at '/Users/chotipat/Dropbox/PSIT2018/ejudge/entryway.py' using Python version 3.6.2. The main text area contains the code `print("Output")`. The status bar at the bottom right shows 'Ln: 1 Col: 15'.

จากนั้นเราสามารถสั่งให้ Idle เริ่มทำงานตามคำสั่งที่ระบุใน entryway.py ด้วยการ click **Run** -> **Run Module** หรือกดปุ่ม **F5** ก็ได้

ผลลัพธ์ของการคำสั่งที่อยู่ใน entryway.py คือการแสดงคำว่า **Output** จะถูกแสดงในหน้าต่างหรือ window ที่ชื่อว่า **Shell** ของ Idle ซึ่งเป็น window เดียวกับที่ใช้ในการทำงานแบบ Interactive mode



The image shows two windows from the Python IDLE application. The top window is the script editor, titled 'entryway.py - /Users/chotipat/Dropbox/PSIT2018/ejudge/entryway.py (3.6.2)'. It contains a single line of Python code: `print("Output")`. The bottom window is the 'Python 3.6.2 Shell'. It shows the execution of the script. The first command is `>>> print("Output")`, which outputs 'Output'. The second command is `>>> print("Hello")`, which outputs 'Hello'. After a third `>>>` prompt, a restart message appears: `===== RESTART: /Users/chotipat/Dropbox/PSIT2018/ejudge/entryway.py =====`. This is followed by the output 'Output' and another `>>>` prompt. The status bar at the bottom of the shell window indicates 'Ln: 14 Col: 4'.

โปรดสังเกตว่า

1. ก่อนจะมีประมวลผลโปรแกรมและแสดงผล โปรแกรม Idle จะมีการ restart shell ที่ใช้ใน run โปรแกรม entryway.py ก่อนทุกครั้ง และ
2. Menu ของหน้าต่างที่ใช้ในการเขียน script mode หรือหน้าต่าง Interactive mode จะไม่เหมือนกัน เช่น menu **Run** จะมีบนหน้าต่างที่เป็น Editor ของ Script mode เท่านั้น แต่ไม่มีบนหน้าต่าง Shell ของ Interactive mode

อีกวิธีหนึ่งของการเขียนโปรแกรมแบบ Script mode คือการเลือกใช้ Editor ใดก็ได้ (ไม่จำเป็นต้องใช้ Editor Window ของ Idle) เพื่อเขียนโปรแกรม ในที่นี้ขอแนะนำโปรแกรม Editor 2 โปรแกรม ได้แก่ 1. Visual Studio Code และ 2. Sublime Text ทั้ง 2 โปรแกรมนี้สามารถ download มาใช้ได้ฟรี โดยไม่เสียค่าใช้จ่าย เมื่อเขียนโปรแกรมใน editor และทำการ Save ลงไปในไฟล์แล้ว เราสามารถเอาไฟล์นั้นมาทำงาน หรือที่เรียกว่า **Run** หรือ **Execute** ได้โดยการเปิดโปรแกรม **Command Prompt** (Windows) หรือ **Terminal** (macOS และ Linux) แล้วสั่งโปรแกรมให้ทำงานโดยพิมพ์

```
$ python entryway.py
```

```
ejudge — bash — 65x11
Chotipats-MacBook-Pro:ejudge chotipat$ pwd
/Users/chotipat/Dropbox/PSIT2018/ejudge
Chotipats-MacBook-Pro:ejudge chotipat$ ls
entryway.py
Chotipats-MacBook-Pro:ejudge chotipat$ python entryway.py
Output
Chotipats-MacBook-Pro:ejudge chotipat$
```

ตัวอย่างในรูป เราได้สั่ง **Run** โปรแกรม **python** ที่ Folder หรือ Directory ที่มีไฟล์ **entryway.py** อยู่ ถ้า **Run** โปรแกรม **python** ที่ Directory อื่น ก็จำเป็นที่จะต้องระบุ **PATH** ของไฟล์ **entryway.py** ให้ถูกต้องด้วย

ผลลัพธ์คำว่า **Output** จะถูกแสดงใน **Terminal** เมื่อแสดงเสร็จ ก็จะแสดง **SHELL Prompt** ในที่นี้คือเครื่องหมาย **\$** เพื่อรอรับคำสั่งเพื่อ **Run** โปรแกรมอื่น หรือคำสั่งที่มีอยู่ใน **SHELL** เช่น **ls** หรือ **pwd** จากผู้ใช้ต่อไป

ใน **eJudge** จะมีโจทย์ชื่อ **Entryway** อยู่ ซึ่งเป็นโจทย์ที่กำหนดให้ผู้เขียนโปรแกรมส่งคำว่า **Output** มาให้คอมพิวเตอร์ เพื่อ print ออกมาทางหน้าจอ และไม่ได้รับค่าใดๆจากผู้ใช้

รูปด้านล่างแสดง Sample Case ที่อยู่ในโจทย์ **Entryway**



Sample Case

Sample Input

Sample Output

Output

เมื่อทดลองส่งไฟล์ **entryway.py** ขึ้น **eJudge** เพื่อทำการตรวจ จะเห็นได้ว่าโปรแกรมทำงานถูกต้องตรงตามที่คาดไว้ (มีการแสดงผลคำว่า **Output** จำนวน 1 บรรทัด) แต่ว่าคุณภาพ (Quality) ของโปรแกรมยังได้ 75% (ซึ่งยังไม่เต็ม 100%) ดังรูปด้านล่าง

The screenshot shows the <e>Judge web interface. On the left is a dark sidebar with navigation links: Home, Course index, Course 63, and Submission. The main content area is titled 'Submission' and shows details for submission #305662. The submission is for problem '001: Entryway' by user 'chotipat(Chotipat Pornavalai)' in Python. It has a correctness score of 100 points, a bonus score of 100 points, and a quality score of 75%. A slider for 'Quality by Inst.' is shown with a value of 0 and a button to 'Update'. The summary score is 150 points and the submission time is 2016-08-02 11:38:43. Below the details, there is a 'Details' section showing 'Case 1 [#11585] : Passed' with a time of 0.04230400 sec. At the bottom, there is a 'Code' section with a code editor showing a single line of Python code: `print("Output")`. A 'Code Simulator' button is visible in the top right corner of the code editor area.

Submission Detail	
ID	#305662
Problem	001: Entryway
Username	chotipat(Chotipat Pornavalai)
Language	Python
Correctness Score	100 Points
Bonus Score	100 Points
Quality	75% How to improve your code
Quality by Inst.	<input type="range" value="0"/> 100 <button>Update</button>
Summary Score	150 Points
Time	2016-08-02 11:38:43

Details

Case 1 [#11585] : Passed 0.04230400 sec.

Code Code Simulator

```
1 | print("Output")
```

เมื่อ click ที่ [How to improve your code](#) จะทราบว่าในโปรแกรม `entryway.py` ยังไม่ได้มีการเขียนส่วนที่เรียกว่า **Module Docstring** ดังรูปด้านล่าง

How to improve your code.



Warning: option include-ids is deprecated and ignored.

Warning: option symbols is deprecated and ignored.

***** Module code

C: 1, 0: Missing module docstring (missing-docstring)

Report

=====

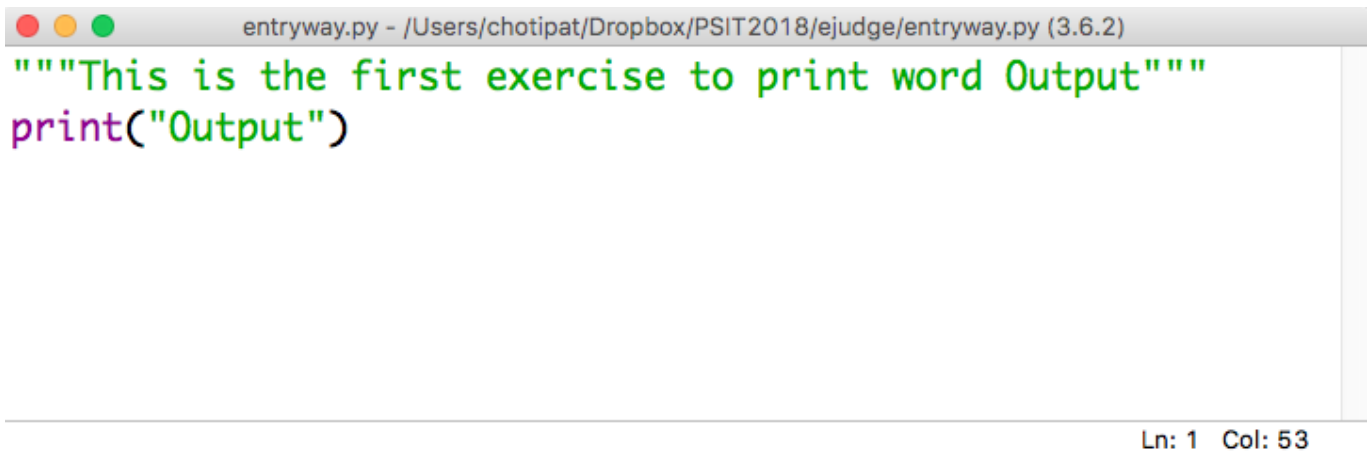
2 statements analysed.

Statistics by type

type	number	old number	difference	%documented	%badname
module	1	1	=	0.00	0.00
class	0	0	=	0	0
method	0	0	=	0	0
function	0	1	-1.00	0	0

โปรแกรมที่เขียนใน entryway.py ทั้งหมด จะถือว่าเป็น 1 module ดังนั้น Module Docstring จะเป็นข้อความที่ใช้ในการอธิบายว่า โปรแกรมที่เขียนในไฟล์นี้เกี่ยวกับอะไร โดย Module Docstring จะเขียนไว้ที่บรรทัดแรกของไฟล์ และมีการใช้เครื่องหมาย Double Quote หรือ Single Quote จำนวน 3 ตัว ล้อมรอบข้อความ Docstring (`""" ข้อความ """` หรือ `''' ข้อความ '''`)

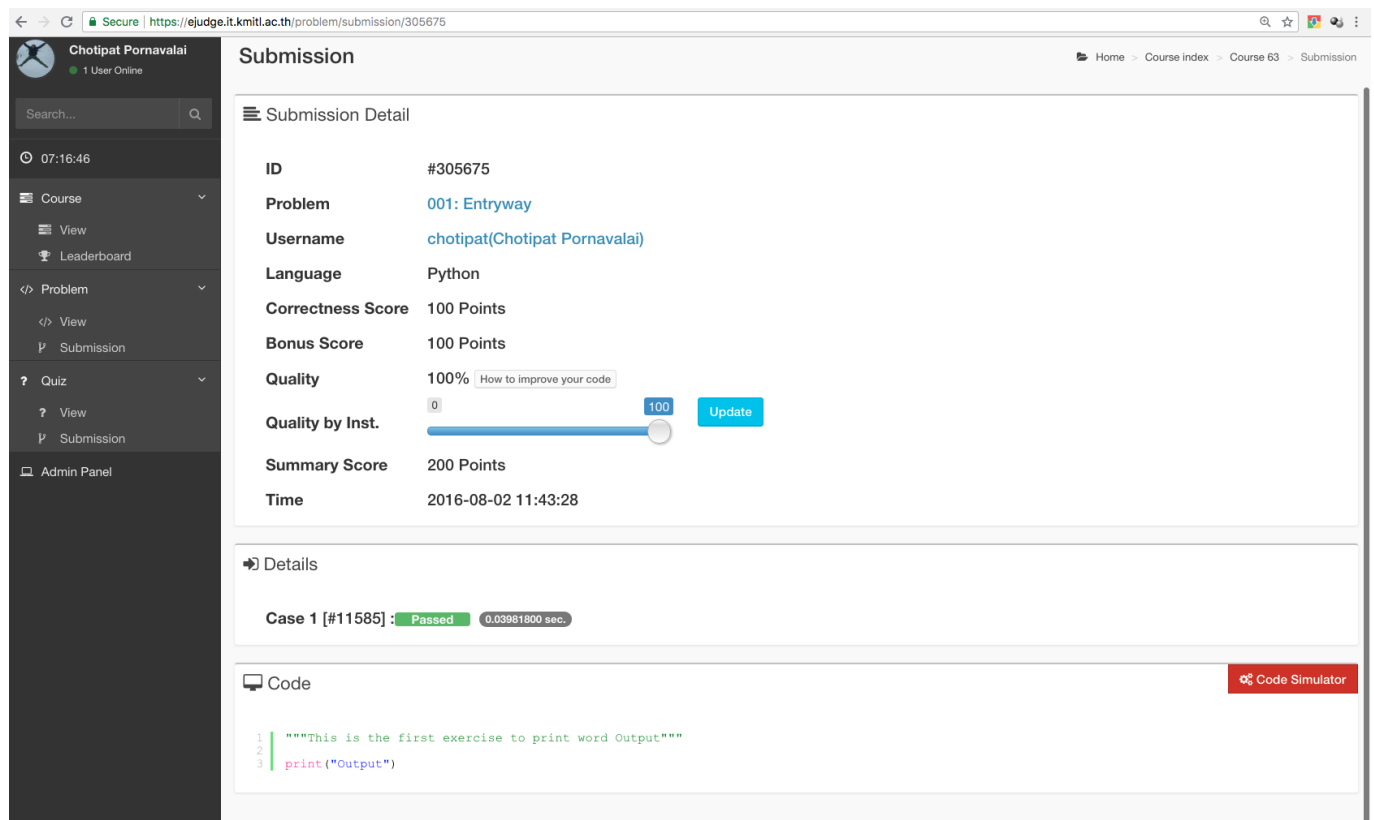
ดังตัวอย่างในรูป



```
entryway.py - /Users/chotipat/Dropbox/PSIT2018/ejudge/entryway.py (3.6.2)
"""This is the first exercise to print word Output"""
print("Output")
```

Ln: 1 Col: 53

เมื่อทำการส่งขึ้น eJudge เพื่อตรวจใหม่อีกครั้ง จะได้คะแนน Quality เต็ม 100% ดังรูปด้านล่าง



Submission

Submission Detail

ID	#305675
Problem	001: Entryway
Username	chotipat(Chotipat Pornavalai)
Language	Python
Correctness Score	100 Points
Bonus Score	100 Points
Quality	100% How to improve your code
Quality by Inst.	<div><div></div></div> 100 Update
Summary Score	200 Points
Time	2016-08-02 11:43:28

Details

Case 1 [#11585]: Passed 0.03981800 sec.

Code [Code Simulator](#)

```
1 """This is the first exercise to print word Output"""
2 print("Output")
```

การเขียนโปรแกรมใน eJudge จะมีการตรวจคุณภาพของโปรแกรมที่ส่งเข้าด้วยตาม standard PEP8 ซึ่งจะมีการกำหนดรูปแบบการเขียนโปรแกรม Python ให้เป็นไปตามมาตรฐานเดียวกัน ในกรณีข้างบน คือ โปรแกรมในทุกๆ Module ควรมีการเขียน Docstring เพื่ออธิบายว่า Module นั้นคืออะไร ทำหน้าที่อะไร นอกจากนี้ยังมีการกำหนดมาตรฐานอื่นๆ เช่น หลักการตั้งชื่อตัวแปร ต้องมีขนาดอย่างน้อยก็ตัวอักษร หลักการเว้นวรรคต่างๆ ในโปรแกรม เป็นต้น รายละเอียดของ PEP8 จะได้กล่าวถึงในบทถัดๆ ไป

Exercise 1 (MoreEntryway)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ MoreEntryway ซึ่งมีการกำหนด Input Specification และ Output Specification และ Sample Input และ Sample Output ไว้ในรูปด้านล่าง

Specification

Input Specification

ไม่มีค่าส่งเข้าในโจทย์ข้อนี้

Output Specification

จำนวน 20 บรรทัด
 ประกอบด้วย Output_หมายเลขบรรทัด

Sample Case

Sample Input

Sample Output

Output_1
 Output_2
 Output_3
 Output_4
 Output_5
 Output_6
 Output_7
 Output_8
 Output_9
 Output_10
 Output_11
 Output_12
 Output_13
 Output_14
 Output_15
 Output_16
 Output_17
 Output_18
 Output_19
 Output_20

Data Types

ข้อมูลที่สามารถประมวลผลในคอมพิวเตอร์ด้วยภาษา Python จะมีอยู่ด้วยกันหลายชนิด ถึงตอนนี้เราได้มีการใช้ข้อมูลเพียงชนิดเดียวที่เรียกว่า String (เขียนย่อเป็น `str`) หรือข้อความ ยกตัวอย่างเช่น คำว่า "Output" และ "Hello"

ข้อความ String จะเป็นข้อความที่ล้อมรอบด้วย Single Quote หรือ Double Quote ดังนั้น จะเขียน "Output" หรือ 'Output' ก็ได้ แต่จะใช้ปนกันไม่ได้ เช่น ใช้ Double Quote ข้างหน้า แต่ใช้ Single Quote ข้างหลัง เช่น "Output"

ในกรณีที่ในข้อความมี Single Quote อยู่ ก็ให้ใช้ Double Quote ล้อมรอบข้อความ String แทน เช่น

```
"It's fun to learn PSIT"
```

และในทางกลับกัน ถ้าในข้อความมี Double Quote อยู่ ก็ให้ใช้ Single Quote ล้อมรอบข้อความ String เช่น

```
'Students said "They love PSIT very much".'
```

แต่ถ้าต้องการใช้ Double Quote ล้อมรอบ ก็สามารทำได้เช่นกันโดยให้มีเครื่องหมาย Backslash อยู่ด้านหน้า Double Quote ที่อยู่ภายในข้อความ String ก่อน เช่น

```
"Students said \"They love PSIT very much\"."
```

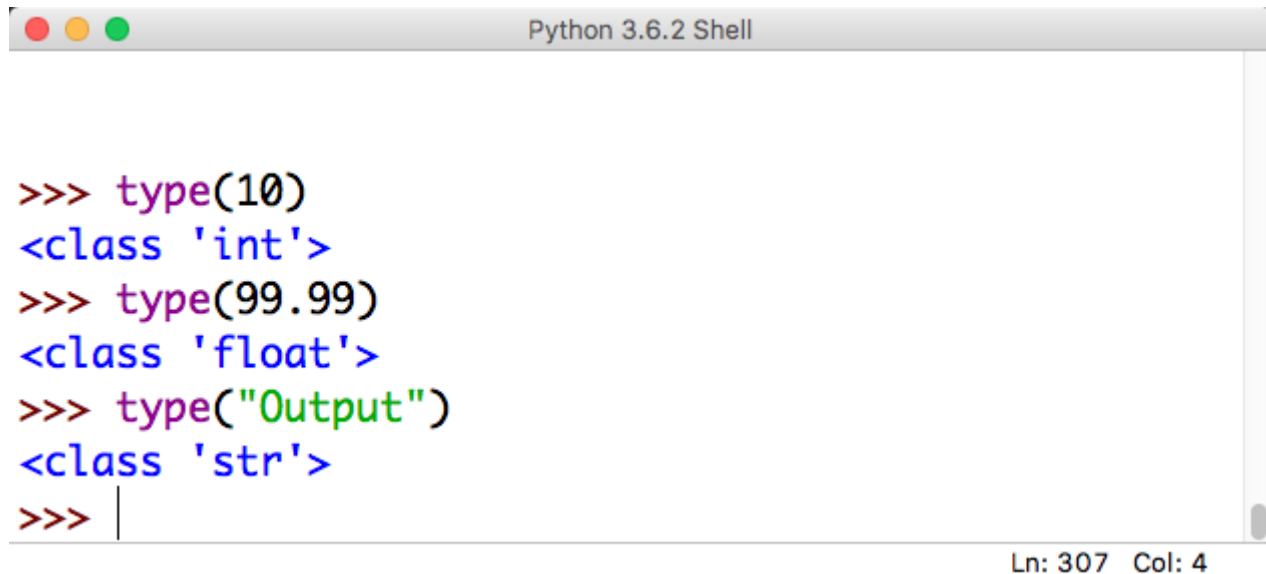
เช่นเดียวกับตัวอย่างข้างบน ถ้าต้องการใช้ Single Quote ล้อมรอบข้อความ String ก็สามารถใช้ Backslash กับ Single Quote ที่อยู่ในข้อความได้เช่นกัน

'It's fun to learn PSIT'

ข้อมูล (Data) ประเภทอื่นที่ทำความรู้จักถัดไปคือ ข้อมูลประเภทตัวเลข หรือ **Numerical** ซึ่งมีอยู่หลายชนิด เช่น Integer Number (**int**) และ Floating Point Number (**float**)

ข้อมูลชนิด **int** คือข้อมูลตัวเลขจำนวนเต็ม ส่วนข้อมูลชนิด **float** จะเป็นข้อมูลตัวเลขที่มีทศนิยม

เราสามารถตรวจสอบชนิดของข้อมูลโดยใช้ Function ที่ชื่อว่า **type()** ดังตัวอย่างในรูปด้านล่าง



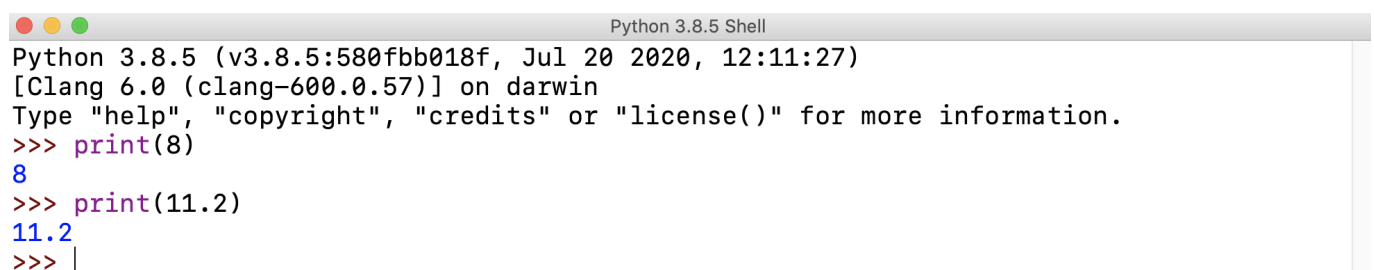
```
Python 3.6.2 Shell

>>> type(10)
<class 'int'>
>>> type(99.99)
<class 'float'>
>>> type("Output")
<class 'str'>
>>> |
```

Ln: 307 Col: 4

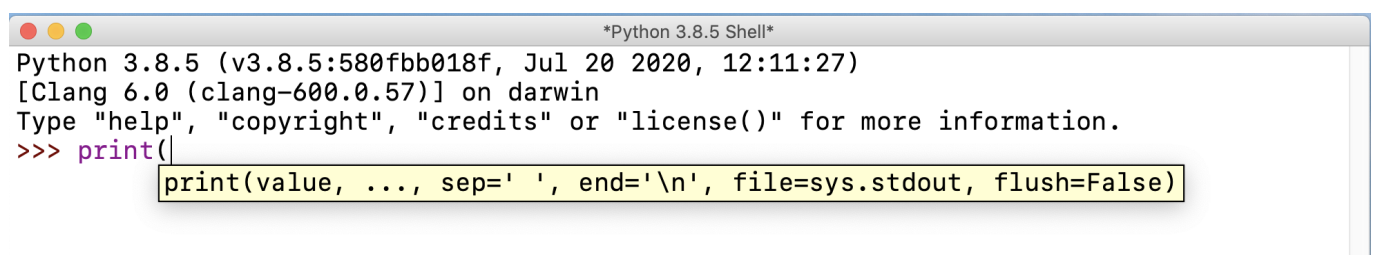
ข้อมูลประเภทตัวเลข เช่น **int** และ **float** จะไม่มีการใช้ Double Quote หรือ Single Quote ล้อมรอบ และสามารถนำข้อมูลประเภทนี้ไปใช้ในการประมวลผลทางคณิตศาสตร์ได้

เราสามารถแสดงผลข้อมูลประเภทตัวเลขด้วย **print** ได้เช่นกัน ไม่จำเป็นต้องเป็นชนิดข้อความ **str** แต่เพียงอย่างเดียว โดยการส่งค่า **value** เช่น **8** หรือ **11.2** ไปให้ฟังก์ชัน **print** เช่น



```
Python 3.8.5 Shell
Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> print(8)
8
>>> print(11.2)
11.2
>>> |
```

เมื่อเราพิมพ์คำว่า **print()** จะมีข้อความหรือตัวช่วยอธิบายการใช้งานของฟังก์ชัน **print** ดังแสดงด้วยข้อความกรอบสีเหลืองในรูปด้านล่าง (เราเรียกข้อความที่แสดงในกรอบสีเหลืองนี้ว่า **Call Stack Visibility**)



```
*Python 3.8.5 Shell*
Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> print(
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

จากข้อความตัวช่วย **value, ...** จะแสดงให้เห็นได้ว่า ค่า หรือ **value** ในฟังก์ชัน **print** สามารถมีได้หลายค่าเนื่องจากการเขียนเครื่องหมาย **,** และ **...** อยู่ ดังนั้น เราจึงสามารถให้แสดงผล 3 ค่า ทั้งตัวเลข **int float** และ **str** ได้โดยการส่งค่า 3 คำนี้นี้ (แยกค่าแต่ละค่าด้วยเครื่องหมาย comma **,**) ในฟังก์ชัน **print** ดังตัวอย่างด้านล่าง

```

Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> print(8, 11.2, "Hello")
8 11.2 Hello
>>> |

```

นอกจากนี้แล้ว ข้อความตัวช่วยของ `print` ยังแสดงให้เห็นว่า เราสามารถใส่ค่า Separator หรือ `sep` ได้ด้วย โดยค่านี้จะมีค่าเริ่มต้น หรือที่เรียกว่า ค่า `default` เป็น ช่องว่าง 1 ช่อง (`sep=' '`) ดังนั้นการแสดงผลข้อความ 3 ค่า จะมีการเว้นวรรค 1 ช่องระหว่างค่าทั้ง 3 ค่านั้น หากเราต้องการเปลี่ยนค่า `sep` เป็นค่าอื่นที่ไม่ใช่ช่องว่างหรือเว้นวรรค 1 ช่อง สามารถทำได้ด้วยตัวอย่างด้านล่าง โดยในตัวอย่างนี้จะเห็นได้ว่าเมื่อเปลี่ยนค่า `sep` เป็น `***` จะทำให้แยกค่า 3 ค่าที่ส่งเข้ามาใน `print` ด้วย `***`

ข้อความตัวช่วยของ `print` จะบอกให้เราทราบด้วยว่ายังมี Parameter อีก 1 ตัวที่เราสามารถปรับเปลี่ยนได้ คือ `end` ซึ่งมีค่าเริ่มต้นเป็น `\n` อ่านว่า Backslash N ซึ่งมีความหมายว่า ขึ้นบรรทัดใหม่ ดังนั้นเมื่อ `print` แสดงข้อความครบหมดแล้ว จะขึ้นบรรทัดใหม่เสมอ หากเราไม่ต้องการขึ้นบรรทัดใหม่สามารถทำได้โดยการเปลี่ยน `\n` เป็นค่าอื่นๆที่ต้องการเช่น `' '` หรือข้อความ `str` ที่ไม่มีอะไรอยู่ในเครื่องหมาย Single Quote เลย หรืออาจจะเปลี่ยนค่าอื่นเช่น `'^^^'` ก็จะแสดง `'^^^'` แทนการขึ้นบรรทัดใหม่ ดังตัวอย่างด้านล่าง

```

Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> print(8, 11.2, "Hello")
8 11.2 Hello
>>> print(8, 11.2, "Hello", sep='***')
8***11.2***Hello
>>> print(8, 11.2, "Hello", sep='***', end='^^^')
8***11.2***Hello^^^
>>>

```

ค่า หรือ `value` ในฟังก์ชัน `print` สามารถเขียนในรูปของ `Expression` หรือ นิพจน์ได้ เช่น `print(1+2)` จะแสดงผลเป็นค่า `3` เราสามารถใช้ตัวดำเนินการทางคณิตศาสตร์ หรือที่เรียกว่า operator เช่น บวก ลบ คูณ หาร ยกกำลัง ได้

การกระทำทางคณิตศาสตร์ในคอมพิวเตอร์ จะกำหนดลำดับการคำนวณไว้ด้วยกฎที่เรียกว่า **PEMDAS** โดยให้ความสำคัญกับ **P, E, [M, D], [A, S]** ตามลำดับ จาก สูงที่สุดไปต่ำที่สุด

P คือ **Parenthesis** ใช้เครื่องหมายวงเล็บ `()` **E** คือ **Exponential** หรือ การยกกำลัง ใช้สัญลักษณ์ operator เป็น `**` เช่น `2**3` หมายความว่า 2 ยกกำลัง 3 **[M, D]** คือ **Multiplication** หรือการคูณ และ **Division** หรือการหาร ใช้สัญลักษณ์ operator `*` และ `/` ตามลำดับ แต่ถือว่ามีความสำคัญเท่ากัน เช่น `2*3` หมายความว่า 2 คูณ 3 และ `10/2` หมายความว่า 10 หารด้วย 2 **[A, S]** คือ **Addition** หรือการบวก และ **Subtraction** หรือการลบ ใช้สัญลักษณ์ operator เป็น `+` และ `-` แต่ถือว่ามีความสำคัญเท่ากัน เช่น `3+4` หมายความว่า 3 บวก 4 และ `11 - 1` หมายความว่า 11 ลบด้วย 1

เนื่องจาก `()` มีความสำคัญลำดับสูงสุด ดังนั้นจะกระทำหรือคำนวณใน `()` ก่อนเสมอ ในกรณีมีวงเล็บซ้อนกัน ก็จะทำคำนวณในวงเล็บในสุดก่อนเสมอ เมื่อกระทำในวงเล็บ แล้วก็กระทำ **Exponential** ก่อน **Multiplication** หรือ **Division** เพราะ **Exponential** มีลำดับที่สูงกว่า ตามลำดับของ **PEMDAS** ไปเรื่อยๆ

นอกจาก operator `+`, `-`, `*`, `/`, `**` แล้ว ในภาษา Python ยังมี operator ที่สำคัญอีก 2 ตัว คือ `//` หรือเรียกว่า Integer Division และ `%` หรือที่เรียกว่า Modulo หรือสั้นๆว่า Mod

`//` จะเป็นตัวดำเนินการกับ Integer เป็นการหารไม่แบบเอาเศษ เช่น `5//3` จะมีค่า `1` เนื่องจาก 5 หาร 3 ได้ผลเป็น 1 เศษ 2 แต่การหารแบบ Integer Division จะไม่เอาเศษ ดังนั้นผลลัพธ์จึงมีค่าเป็น 1

% จะเป็นตัวดำเนินการกับ Integer เป็นการหารแบบเอาเศษ เช่น $5\%3$ จะมีค่า 2 เนื่องจาก 5 หาร 3 ได้ผลเป็น 1 เศษ 2 แต่การหารแบบ Modulo จะเอาแต่เศษ ไม่เอาผลหารที่เป็นจำนวนเต็ม ดังนั้นผลลัพธ์จึงมีค่าเป็น 2

ตัว operator ทั้ง `//` และ `%` จะมีความสำคัญในระดับ [M, D] เช่นเดียวกับ operator `*` และ `/`

ในกรณีที่มีการกระทำกับตัวกระทำ (Operator) ที่มีความสำคัญเท่ากัน จะกระทำจากตัวกระทำ (Operator) จากซ้ายไปขวา ยกตัวอย่างเช่น ถ้ามี operator `+` และ `-` ซึ่งมีลำดับความสำคัญเท่ากัน ก็จะทำ operator ทางซ้ายก่อน ยกเว้นกรณียกกำลังซ้อนกัน ให้ทำจากขวามาซ้าย เช่น $2**2**3$ จะมีค่าเท่ากับ 256 หรือเท่ากับ $2**8$ กล่าวคือให้ทำ $2**3$ ซึ่งอยู่ด้านขวาก่อน ได้ผลลัพธ์เป็น 8 ก่อน ถ้ากระทำจากซ้ายมาขวา จะได้ผลเป็น $2**2$ หรือ 4 แล้วนำไปยกกำลัง 3 ได้เป็น $4**3$ ได้ผลลัพธ์เป็น 64

ใน eJudge มีโจทย์ชื่อว่า FollowTheLead ที่ต้องการให้เขียนโปรแกรมแสดงผลลัพธ์ของ 10 นิพจน์ ในรูปด้านล่าง

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 \quad (1)$$

$$10 - 9 - 8 - 7 - 6 - 5 - 4 - 3 - 2 - 1 \quad (2)$$

$$1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 \times 0 + 1 + 1 \quad (3)$$

$$1 \times 2 + 3 \times 4 \quad (4)$$

$$11 + 22 - 33 \times 44 \div 55 \quad (5)$$

$$(7 + 2 - 3) \times 4 \quad (6)$$

$$(42 - 11) \times (7 \times 2 + 4^7) \quad (7)$$

$$\frac{3 \times (9^2 + (2)(5) - 10)}{(2)(7)} \quad (8)$$

$$\left(\frac{7-1}{2+4}\right) \times \left(\frac{10}{51}\right) \quad (9)$$

$$\frac{(4000 \times 50^4 \times 87) + (4000 \times 5)}{\sqrt{(72 - 111)^2 + (10 - 314)^2}} \quad (10)$$

ยกตัวอย่างนิพจน์ที่ 4 ของ **FollowTheLead** จะมีการกระทำทั้งหมด 3 ขั้นตอนตามลำดับดังนี้

1. ให้กระทำการคูณก่อนการบวก แต่เนื่องจากการคูณ 2 ครั้ง คือ 1×2 และ 3×4 ซึ่งเป็นการคูณเหมือนกัน จึงมีลำดับเท่ากัน ถ้ามีลำดับเท่ากัน ให้กระทำจากซ้ายไปขวา ดังนั้นจึงกระทำ 1×2 ก่อน ได้ผลลัพธ์เป็น 2
2. ต่อเนื่องจากข้อ 1 ข้างบน ลำดับถัดมาจึงกระทำ 3×4 ได้ผลลัพธ์เป็น 12
3. เมื่อกระทำการคูณจนหมดแล้ว จึงมาทำการกระทำในลำดับต่ำกว่าคือการบวกและการลบ ดังนั้นจึงนำ $2 + 12$ และได้ผลลัพธ์เป็น 14 เป็นคำตอบของนิพจน์ที่ 4

ในทางคณิตศาสตร์ เรามักใช้เครื่องหมาย \times แทนการคูณ แต่ในโปรแกรมคอมพิวเตอร์เราจะใช้เครื่องหมาย $*$ แทน

นอกจากนี้แล้ว การเขียนการกระทำทางคณิตศาสตร์ สามารถมีช่องว่างหรือไม่ก็ได้ เช่น $3*4$ สามารถเขียนเป็น $3 * 4$ หรือ $3* 4$ หรือ $3 *4$ หรือ $3 * 4$ เป็นต้นก็ได้ และได้ผลลัพธ์เท่ากันคือ 12 ดังนั้นในการเขียนโปรแกรมที่ดีเพื่อให้อ่านง่าย เรามักจะมีการใส่ช่องว่าง ยกตัวอย่าง $1*2+3*4$ ถ้าไม่ต้องการใช้วงเล็บ ก็ควรเขียนเป็น

```
1*2 + 3*4
```

การเขียนเช่นนี้เพื่อให้เห็นได้ง่ายขึ้นว่าให้ทำการคูณ $1*2$ และ $3*4$ ตามลำดับ แล้วจึงนำผลคูณของทั้ง 2 ค่า มาบวกกันทีหลัง

Exercise 2 (FollowTheLead)

ให้ผู้เรียนลองทดลองเขียนโปรแกรมข้อ **FollowTheLead** ให้ผ่านใน eJudge โดยเขียนอีก 9 นิพจน์ที่เหลือด้วยตนเอง

A Simple Program with Variables

จาก Exercise **FollowTheLead** นิพจน์ที่ 10 เป็นนิพจน์ที่ซับซ้อนที่สุดและมีโอกาสในการเขียนผิดมากที่สุด เนื่องจากจะต้องมีการใช้วงเล็บซ้อนวงเล็บอยู่หลายครั้ง ดังตัวอย่างของคำตอบด้านล่าง

```
print(((4000*50**4*87)+(4000*5))/((72-111)**2+(10-314)**2)**0.5)
```

ถ้าเป็นการกตเครื่องคิดเลข เราอาจจะมีการกดเพื่อหาคำตอบเป็นส่วนๆ แล้วจดไว้ แล้วจึงค่อยนำตัวเลขที่จดไว้ มากดต่อเพื่อหาคำตอบ เช่นเดียวกัน เราสามารถเก็บผลคำตอบเป็นส่วนๆ ได้เช่นเดียวกัน ในการเขียน โปรแกรมคอมพิวเตอร์ ที่เก็บค่าหรือคำตอบได้ จะเรียกว่า **Variable** หรือ ตัวแปร

ดังนั้นคำตอบของนิพจน์ที่ 10 สามารถเขียนได้โดยใช้ตัวแปรดังนี้

```
temp1 = 4000*50**4*87 + 4000*5 temp2 = (72-111)**2 + (10-314)**2 temp3 = temp2**0.5 print(temp1/temp3)
```

จะเห็นว่าการสร้างตัวแปร 3 ตัว ได้แก่ **temp1 temp2 temp3** เพื่อเก็บผลการคำนวณบางส่วนของนิพจน์

การเขียนเครื่องหมาย = หมายความว่า เป็นการเอาค่าที่อยู่ทางด้านขวาของเครื่องหมาย = ไปไว้ที่ตัวแปรที่อยู่ด้านซ้ายของเครื่องหมาย =

เราไม่สามารถเขียนแบบด้านล่างนี้ได้ เนื่องจากตัวแปรต้องอยู่ด้านซ้ายของเครื่องหมาย =

```
4000*50**4*87 + 4000*5 = temp1
```

เนื่องจากเครื่องหมาย = ของการเขียนโปรแกรม มีความหมายว่าเป็นการกำหนดค่า ไม่ได้หมายความว่าเท่ากัน ดังนั้น บรรทัดที่มีการใช้เครื่องหมาย = จะเรียกว่า **Assignment Statement**

ดังนั้นเราสามารถเขียน

```
temp1 = 0
```

```
temp1 = temp1 + 1
```

ได้ โดยไม่ผิดแต่อย่างใด บรรทัดแรกหมายความว่า เอาค่า 0 ให้กับ **temp1** และบรรทัดที่สองหมายความว่า เอาค่า 1 ไปบวกกับ **temp1** ที่มีค่าเป็น 0 ในบรรทัดแรก ได้คำตอบเป็น 1 แล้วเอาไปเก็บในตัวแปร **temp1** อีกครั้ง ตอนนี้ **temp1** ใน

บรรทัดที่สองจะมีค่าเป็น 1

เราสามารถเขียนย่อบรรทัดที่สองข้างบนเป็น

```
temp1 += 1
```

ได้ และสามารถใช้ operator อื่นๆ เช่น * / - ได้เช่นกัน ยกตัวอย่างเช่น

```
temp1 *= 10
```

```
temp1 /= 2
```

```
temp1 -= 5
```

เป็นต้น

การตั้งชื่อตัวแปรมีกฎอยู่หลายข้อ หลักการตั้งทั่วไปคือ

1. ขึ้นต้นด้วยตัวอักษรภาษาอังกฤษ แม้จะใช้ตัวพิมพ์ใหญ่แต่ปกติจะให้ตัวเล็กทั้งหมด และห้ามขึ้นต้นด้วยตัวเลข หรือ อักขระพิเศษ ยกเว้น `_` สามารถใช้ได้ (แต่มีความหมายพิเศษ)
2. ห้ามเว้นวรรค ถ้าต้องการสร้างตัวแปรชื่อให้นักศึกษาให้ตั้งชื่อว่า `student_name` โดยใช้ `_` แทนการเว้นวรรค
3. ตัวถัดไปเป็นตัวอักษรภาษาอังกฤษหรือตัวเลขก็ได้ แต่ห้ามใช้อักขระพิเศษยกเว้น `_` เท่านั้น
4. ควรมีอย่างน้อย 3 ตัวอักษร ยกเว้นกรณีเป็นตัวนับ (Counter) สามารถใช้ 1 ตัวอักษรได้ แต่ก็ไม่ควรยาวเกินไป และควรมีความหมายสื่อถึงข้อมูลที่ตัวแปรจัดเก็บ
5. ห้ามใช้คำ (keyword) ที่อยู่ในรายการคำสงวน (reserved words) เป็นชื่อตัวแปร รูปด้านล่าง แสดงรายชื่อคำสงวนในภาษา Python 3.8.5



Exercise 3 (SaveComputeRepeat)

ให้ผู้เรียนลองทดลองเขียนโปรแกรมข้อ `SaveComputeRepeat` ใน eJudge โปรแกรมนี้ให้คำนวณหาค่าของ days hours minutes seconds milliseconds จากค่า milliseconds ที่โจทย์ให้มา

Hint 1

โจทย์ข้อนี้กำหนดให้พิมพ์ค่าหลายๆค่าในบรรทัดเดียวกัน สามารถทำได้โดยการใส่ค่าเหล่านั้นลงไปโดยแยกค่าแต่ละตัวด้วย `comma` ดังตัวอย่างด้านล่าง (สังเกตได้ว่า เราสามารถเอาตัวแปรเป็นค่า `value` ใน `print` ได้ เนื่องจากตัวแปรเป็นตัวเก็บค่าอยู่แล้ว) `a = 1 b = 2 c = 'Hello' print(a, b, '3', c, 'World')`

จะได้ผลการพิมพ์เป็น

```
1 2 3 Hello World
```

โดย `print` จะรับค่า `value` เข้าไป 5 ค่า ซึ่งเราเรียกว่า `Argument` และพิมพ์ค่าทั้ง 5 ค่านี้ในบรรทัดเดียวกัน แยกกันด้วยช่องว่าง 1 ช่อง ค่า `a b c` เป็นค่าที่เก็บในตัวแปร จะถูกนำมาแสดงผลเวลาพิมพ์

Hint 2

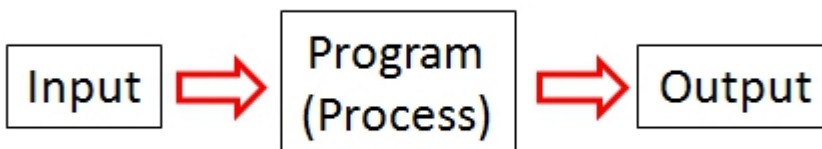
โจทย์ข้อนี้จะต่างข้อก่อนหน้านี้ตรงที่จะมีการเก็บค่าในตัวแปร ซึ่งถ้าส่งเข้าไปที่ eJudge จะพบว่าได้ Quality ไม่เต็ม 100% (Invalid constant name) ทั้งนี้เป็นเพราะข้อกำหนดใน PEP8 กำหนดว่า โปรแกรมควรมีการกำหนดค่าตัวแปรไว้ใน Function หรือถ้ามีการเก็บค่าไว้นอก Function ควรเก็บให้เป็น Constant หรือเป็นค่าคงที่ เท่านั้น เนื่องจากใน PEP8 กำหนดว่า ถ้าต้องการสร้าง constant ให้ตั้งชื่อตัวแปร constant นั้นด้วยตัวพิมพ์ใหญ่ทั้งหมดเท่านั้น แต่ในกรณีของโจทย์ข้อนี้ ค่าที่เป็น constant หรือค่าคงที่ มีเพียงค่าเดียวคือ START_HERE = 492137954293754252786 ซึ่งควรจะเขียนเป็นตัวพิมพ์ใหญ่ทั้งหมด แต่ตัวแปรอื่นๆ ในข้อนี้ไม่ใช่ constant

How to improve your code. ×

```
Warning: option include-ids is deprecated and ignored.
Warning: option symbols is deprecated and ignored.
***** Module code
C: 11, 0: Final newline missing (missing-final-newline)
C: 1, 0: Missing module docstring (missing-docstring)
C: 1, 0: Invalid constant name "start_here" (invalid-name)
C: 2, 0: Invalid constant name "milliseconds" (invalid-name)
C: 3, 0: Invalid constant name "seconds" (invalid-name)
C: 4, 0: Invalid constant name "milliseconds" (invalid-name)
C: 5, 0: Invalid constant name "minutes" (invalid-name)
C: 6, 0: Invalid constant name "seconds" (invalid-name)
C: 7, 0: Invalid constant name "hours" (invalid-name)
C: 8, 0: Invalid constant name "minutes" (invalid-name)
C: 9, 0: Invalid constant name "days" (invalid-name)
C: 10, 0: Invalid constant name "hours" (invalid-name)
```

เนื่องจากในโปรแกรมข้างบนทั้งหมด เรายังไม่ได้เรียนการสร้าง Function ขึ้นมาเอง ดังนั้นการทำโจทย์ข้อนี้หรือข้ออื่นๆ ในบทนี้ อาจจะยังมีบางข้อที่ทำให้ไม่ได้ Quality ครบ 100% และเราจะเรียนรู้การสร้างและใช้ Function ขึ้นเองในบทต่อไป

A Simple Program with Input



การรับข้อมูลเข้า สามารถทำได้โดยใช้ function ชื่อว่า input() ซึ่งมีการใช้งานแบบง่ายได้ดังนี้

```
word = input()
```

เมื่อโปรแกรม run บรรทัดนี้ ก็จะรอรับข้อความที่ผู้ใช้ใส่เข้าไป จนถึงการกดปุ่ม **Enter** ข้อความนั้นก็จะถูกเก็บในตัวแปร **word**

แม้ว่าข้อความที่พิมพ์เข้ามาโดยผู้ใช้จะเป็นตัวเลข การจัดเก็บข้อมูลใน **word** จะเป็นข้อมูลชนิด **String** หรือ **str** เท่านั้น ดังนั้นถ้าผู้ใช้พิมพ์ว่า **123** การจัดเก็บในตัวแปร **word** จะเป็น **'123'**

จากนั้นเราสามารถนำตัวแปร **word** ไปประมวลผลหรือ **print** ได้

Exercise 4 (StillJumping)

ให้ผู้เรียนทำข้อ **StillJumping** ใน eJudge

Exercise 5 (JumpAround)

ให้ผู้เรียนทำข้อ **JumpAround** ใน eJudge

ข้อนี้จะมีความแตกต่างจากข้อ **StillJumping** ตรงที่จะมีการประมวลผลข้อมูลที่ได้รับเข้าจากผู้ใช้ก่อนที่จะมีการ print ออกหน้าจอ และการประมวลในข้อนี้เป็นการประมวลผลทางคณิตศาสตร์ ดังนั้นข้อมูลที่ได้รับเข้ามาจะต้องถูกแปลงชนิดจาก **Str** เป็นข้อมูลชนิดตัวเลขเช่น **Int** หรือ **Float** ก่อน

ถ้าต้องการแปลงข้อมูลเป็น **Int** ให้ใช้ function ชื่อว่า **int()** และถ้าต้องการแปลงข้อมูลเป็น **Float** ให้ใช้ function ชื่อว่า **float()** ดังตัวอย่างด้านล่าง

```
word = '123'
```

```
num_int = int(word)
```

```
num_float = float(word)
```

num_int จะมีค่าเป็น **123** และ **num_float** จะมีค่าเป็น **123.0** ตามลำดับ

โปรดระวังว่า **'123'** และ **123** ไม่เหมือนกัน ตัวแรกเป็น string เพราะมี single quote ล้อมรอบ ส่วนตัวที่สองเป็น Integer

ในทางกลับกัน หากเรามี ตัวแปร **score = 100** ซึ่งเก็บค่า Integer ไว้ หากต้องการแปลงค่าเป็น string ก็สามารทำได้เช่นกัน โดยใช้ function ชื่อว่า **str()** เช่น **score_str = str(score)** โดย **score_str** จะเก็บค่า string **'100'** ไว้ในตัวแปร

Exercise 6 (Timing)

ให้ผู้เรียนทำข้อ **Timing** ใน eJudge

Exercise 7 (Boomerang)

ให้ผู้เรียนทำข้อ **Boomerang** ใน eJudge

Exercise 8 (Gift I)

ให้ผู้เรียนทำข้อ **Gift I** ใน eJudge

Format String

การแสดงผลข้อความ เราสามารถปรับรูปแบบของข้อความให้ตรงกับความต้องการได้ โดยในภาษา Python สามารถทำได้หลายวิธี เช่น

1. String Formatting (% operator)
2. String Formatting (str.format)
3. String Formatting (f-strings)

วิธีที่ 1: % operator

วิธีนี้เป็นวิธีแบบดั้งเดิม ที่มีแนวคิดมาจากภาษา C จะพบเห็นใน Code เก่าๆ หรือ Legacy code ของภาษา Python เช่น โปรแกรมที่เขียนด้วยภาษา Python version 2

```
#1
print("#1: This is integer output %d" % 14)
#2
print("#2: Integer with given <width> %8d" % 231)
#3
print("#3: Integer (left aligned) %-8d" % 231)
#4
print("#4: Integer (padding 0) %08d" % 231.9954)
#5
print("#5: Floating point output %f" % 3.1415)
#6
print("#6: Forcing floating point output %f" % 17)
#7
print("#7: Floating point with given <precision> %.9f" % 231.22)
#8
print("#8: Shortest representation of floating point output %g" % 31.148720000)
#9
print("#9: String output %s" % "Hello World")
#10
print("#10: More than one format sequences %s %f %d" % ("Hello", 5.18, 60))
```

จากรูปด้านบน แสดงตัวอย่าง string formatting ทั้งหมด 10 ตัวอย่าง

ข้อความที่ตามหลังเครื่องหมาย # จะเป็นข้อความ comment ซึ่งตัว Python จะไม่ได้นำไปประมวลผลใดๆ ยกตัวอย่างเช่น #1 เป็นต้น ข้อความ comment จะเป็นข้อความที่ผู้เขียน โปรแกรม เขียนเพื่อ comment ส่วนของ โปรแกรมไว้ เพื่อให้สามารถเข้าใจส่วนของ โปรแกรมนั้นได้ง่ายขึ้น

ในวิธีนี้ เราจะใช้เครื่องหมาย % ซึ่งดูเหมือนจะเป็นเครื่องหมาย Modulo ที่ใช้ในการคำนวณทางคณิตศาสตร์ เพื่อหาเศษจากการหาร ที่ได้กล่าวถึงไปด้านบน แต่ในที่นี้เครื่องหมาย % เมื่อใช้กับ string จะถูกเรียกว่า **format operator** โดยข้อความ string ที่อยู่ด้านหน้าเครื่องหมาย % หรือตัวถูกดำเนินการด้านหน้า (**first operand**) จะถูกเรียกว่า **format string**

จากคำอธิบายข้างบน เราสามารถเขียนสรุปการใช้ได้ดังด้านล่างนี้

first operand % second operand

หรือ

format string % (value1, value2, ...)

ในที่นี้ **second operand** จะเป็นข้อมูลชนิดที่เรียกว่า **tuple** ซึ่งจะได้กล่าวถึงภายหลังต่อไป

ภายใน **format string** จะมีสิ่งที่เรียกว่า **format sequence** 1 ตัว หรือมากกว่า ก็ได้เอาไว้กำหนด **format** ของค่าของ **second operand** และจะต้องมีจำนวน **format sequence** เท่ากับจำนวน **value** ที่อยู่ใน **second operand** เช่น ตัวอย่างที่ #1 ในรูปด้านบน จะมีการแสดงผล 14 ที่ตำแหน่ง **%d** ซึ่งในที่นี้ **%d** เอาไว้ **format** ตัวเลขจำนวนเต็ม ในที่นี้ **'This is integer output %d'** คือ **format string** หรือ **first operand** และ **%d** คือ **format sequence** และ (14) คือ **second operand** (ในกรณีมีค่าเดียวเช่น 14 อาจจะใส่วงเล็บล้อมรอบ 14 หรือไม่ก็ได้)

หากต้องการ **format** เป็น จำนวนที่มีจุดทศนิยม หรือ **float** ให้ใช้ **%f** เป็น **format sequence** ได้เป็นต้น เช่น ในตัวอย่างที่ #6

ในตัวอย่างที่ #10 ในรูปด้านบน จะเห็นว่าใน **format string** หรือ **first operand** จะมี **format sequence** อยู่ 3 ตัว ได้แก่ **%s %f และ %d** และใน **second operand** จะมีค่า **value** อยู่ 3 ค่า ได้แก่ **"Hello" 5.18 และ 60** ตามลำดับ

และในตัวอย่างที่ #10 ถ้าหากว่าจำนวน **value** มีไม่เท่ากับ 3 ค่า (กล่าวคือ น้อยกว่าหรือมากกว่าจำนวนของ **format sequence**) จะทำให้โปรแกรมเกิด Error ได้ และหากชนิดของข้อมูล ไม่ตรงกับ **format sequence** เช่น **"%d " % 'hello'** โปรแกรมก็จะเกิด Error เช่นกัน ประเภทของ Error แบบต่างๆ จะได้กล่าวถึงในภายหลัง

รูปแบบของ **format sequence** จะเป็นดังนี้

```
%[<flags>][<width>][.<precision>]<type>
```

ตารางด้านล่างแสดงตัวอย่างของ **format sequence** ที่ใช้บ่อย

format sequence	Description
%d หรือ %i	จำนวนเต็ม (int) - ในตัวอย่างนี้ d และ i คือ <type> ใน format sequence
%f	จำนวนจริง (float) - ในตัวอย่างนี้ f คือ <type> ใน format sequence
%s	ข้อความ (str) - ในตัวอย่างนี้ s คือ <type> ใน format sequence
%8d	จำนวนเต็ม (int) ขนาดกว้าง 8 ตัว เรียงชิดด้านขวา (8 character wide, right aligned) - ในตัวอย่างนี้ 8 คือ <width> และ d คือ <type>
%05d	จำนวนเต็ม (int) ขนาดกว้าง 5 ตัว ชิดด้านขวา เต็มเลข 0 ด้านหน้าถ้าไม่ครบ 5 ตัวอักษร (5 character wide, right aligned, padding with 0s) เช่น ค่า 123 จะแสดงเป็น 00123 - ในตัวอย่างนี้ 0 คือ <flag> และ 5 คือ <width> และ d คือ <type>
%-8d	จำนวนเต็ม (int) ขนาดกว้าง 8 ตัว เรียงชิดด้านซ้าย (8 character wide, left aligned) ในตัวอย่างนี้เครื่องหมายลบ - คือ <flag> และ 8 คือ <width> และ d คือ <type>
%12f	จำนวนจริง (float) ขนาดกว้าง 12 ตัว (ความกว้างรวมจุดทศนิยมด้วย) (12 character wide) - ในตัวอย่างนี้ 12 คือ <width> และ f คือ <type> ส่วนค่า <precision> ไม่ได้ระบุไว้ ดังนั้นจะใช้ค่าเริ่มต้น หรือ ค่า default ซึ่งมีค่าเป็น 6
%.4f	จำนวนจริง (float) มีเลขทศนิยม 4 ตัวหลัก (4 digit after decimal) - ในตัวอย่างนี้ 4 คือ <precision> และ f คือ <type>

format sequence	Description
%6.2f	จำนวนจริง (float) มีขนาดกว้างรวม 6 ตัว และมีทศนิยม 2 หลัก (6 total character wide, 2 after decimal) เช่น ค่า 123.4567 จะแสดงเป็น 123.46 (ความกว้างรวม 6 คือรวมจุดทศนิยมด้วย) ถ้าเปลี่ยน format sequence ให้ความกว้างน้อยกว่า 6 เช่น %5.2f ก็ยังแสดงผลแบบเดียวกัน แสดงว่าถ้าจำนวนนั้นมีขนาดกว้างกว่าขนาดที่กำหนดไว้ขนาดที่กำหนดไว้จะไม่มีผลใดๆ - ในตัวอย่างนี้ 6 คือ <width> และ 2 คือ <precision> และ f คือ <type>
%6.4e	จำนวนจริงในรูปแบบ exponential เช่น ค่า 123.456789 จะแสดงเป็น 1.2346e+02 - ในตัวอย่างนี้ 6 คือ width และ 4 คือ precision และ e คือ <type> (กรณีของ <type> e ค่า precision เป็นจำนวนหลักของทศนิยม)
%g	แสดงผล floating point คล้ายๆกับ %f หรือ %e โดยจะแสดงผลแบบที่สั้นหรือกระชับกว่า โดยจะแสดงผลค่าเหมือนกับ %e ถ้าด้วยกำลังน้อยกว่า -4 (ได้แก่ -5, -6, ...) หรือมากกว่า precision หากไม่ใช่ให้แสดงผลคล้ายกับ %f - ในตัวอย่างนี้ g คือ <type> (กรณีของ <type> g ค่า precision คือ จำนวน digit ก่อนและหลังจุดทศนิยรวมกัน โดยค่า default precision มีค่าเท่ากับ 6) เนื่องจากนิยามค่าของ precision ของ %f กับ %g ไม่เหมือนกัน ดังนั้นผลที่ได้ของ %g จึงอาจจะไม่เหมือนกับ %f แต่มีความคล้ายกับ %f ได้

ในรูปแบบของ format sequence จะเห็นได้ว่า **flag** **width** และ **precision** จะถูกล้อมรอบด้วยเครื่องหมายวงเล็บก้ามปู [] หรือที่เรียกว่า **square bracket** ซึ่งมีความหมายความว่า สิ่งที่อยู่ใน [] จะมีหรือไม่มีก็ได้ แต่ **type** ไม่ได้ถูกล้อมรอบด้วย [] ดังนั้นจะสังเกตได้ว่าตัวอย่าง **format sequence** ในตารางด้านบน ทุกตัวอย่างจะต้องมี **type** เสมอ โดยบางตัวอย่าง อาจจะมี **flag** หรือ **width** หรือ **precision** หรือไม่มีก็ได้

เนื่องจากเครื่องหมาย % ถูกใช้ใน **format sequence** ดังนั้นถ้าหากต้องการพิมพ์คำว่า 'The average score is 99%' เราต้องทำการ **escape** เครื่องหมาย % โดยการใส่เครื่องหมาย %% หรือ เครื่องหมาย % 2 ตัวติดกัน และเขียนโปรแกรมดังนี้ 'The average score is %d%%' % 99

Exercise 9 (FormatSequence)

ให้ผู้เรียนลองเขียนตัวอย่างของ **format string** ทั้ง 10 ตัวอย่าง ลงใน Interactive mode ของ Python และทำความเข้าใจการใช้ **format sequence** แบบต่างๆ

ผลลัพธ์ที่ได้จะเป็นตามรูปด้านล่างนี้ ให้ผู้เรียนลองสังเกตผลที่ได้ และทดลองเปลี่ยนค่าต่างๆและดูว่าได้ผลตามที่คาดไว้หรือไม่

Exercise นี้ ไม่จำเป็นต้องส่ง eJudge

```

Python 3.9.6 (v3.9.6:db3ff76da1, Jun 28 2021, 11:49:53)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/chotipat/Google Drive/PSITBook/Version2021/formatsequenceoperator.py
#1: This is integer output 14
#2: Integer with given <width>          231
#3: Integer (left aligned) 231
#4: Integer (padding 0) 00000231
#5: Floating point output 3.141500
#6: Forcing floating point output 17.000000
#7: Floating point with given <precision> 231.220000000
#8: Shortest representation of floating point output 31.1487
#9: String output Hello World
#10: More than one format sequences Hello 5.180000 60
>>>
>>> |

```

วิธีที่ 2: str.format

รูปแบบการใช้ str.format จะเป็นดังนี้

```
%\<template>.format(\<positional arguments>, <keyword_argument>)
```

โดย template คือข้อความ string ที่มี replacement fields อยู่ข้างใน และใน () ของ format สามารถใช้ positional_argument หรือ keyword_argument ก็ได้ โดย replacement field จะใช้สัญลักษณ์ {}

ยกตัวอย่างการใช้ positional argument เป็นดังรูปด้านล่างนี้

```

>>> #1
>>> "The temperature in {} is {} and air quality is {} AQI".format("Bangkok", 32.1, 15)
'The temperature in Bangkok is 32.1 and air quality is 15 AQI'
>>> #2
>>> "The temperature in {0} is {1} and air quality is {2} AQI".format("Bangkok", 32.1, 15)
'The temperature in Bangkok is 32.1 and air quality is 15 AQI'
>>> #3
>>> "The temperature in {2} is {0} and air quality is {1} AQI".format(32.1, 15, "Bangkok")
'The temperature in Bangkok is 32.1 and air quality is 15 AQI'
>>>

```

จากรูปด้านบน {} คือ **replacement field** ซึ่งจะมีหมายเลขอยู่ด้านในหรือไม่ก็ได้ หากไม่มี ลำดับการแสดงผลจะเป็นไปตามลำดับของ positional argument ที่อยู่ภายใน format() ดังนั้นตัวอย่างที่ #1 และ #2 จะให้การแสดงผลที่เหมือนกัน เพราะ "Bangkok" คือลำดับที่ 0 และ 32.1 คือลำดับที่ 1 และ 15 คือลำดับที่ 2 ตามลำดับเหมือนกัน ดังนั้นจะใส่ตัวเลขใน {} หรือไม่ใส่ก็ได้ การไม่ใส่หมายเลขใน **replacement field** เรียกว่า **automatic field numbering**

แต่ในตัวอย่างที่ #3 จะมีการแสดงผลเป็นไปตามลำดับหมายเลขที่อยู่ภายใน **replacement field** กล่าวคือ 32.1 จะเป็นลำดับที่ 0 และ 15 เป็นลำดับที่ 1 และ "Bangkok" เป็นลำดับที่ 2 เพื่อให้การแสดงผลเหมือนกับตัวอย่างที่ #1 และ #2

เราสามารถมีจำนวน replacement field น้อยกว่าหรือเท่ากับจำนวน argument ใน format() ได้ แต่จะมากกว่าไม่ได้ และจะให้ตัวเลขใน replacement field นอกขอบเขตของลำดับของ argument ไม่ได้เช่นกัน ดังอย่างในรูปด้านล่าง ตัวอย่างที่ #1 เป็นการเขียนที่ valid แต่ตัวอย่างที่ #2 และ #3 เป็นการเขียนที่เกิด Error ขึ้น

```
>>> #1
>>> "The temperature in {} is {}".format("Bangkok", 32.1, 15)
'The temperature in Bangkok is 32.1'
>>> #2
>>> "The temperature in {0} is {1} and air quality is {2} AQI".format("Bangkok", 32.1)
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    "The temperature in {0} is {1} and air quality is {2} AQI".format("Bangkok", 32.1)
IndexError: Replacement index 2 out of range for positional args tuple
>>> #3
>>> "The temperature in {0} is {1} and air quality is {3} AQI".format("Bangkok", 32.1, 15)
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    "The temperature in {0} is {1} and air quality is {3} AQI".format("Bangkok", 32.1, 15)
IndexError: Replacement index 3 out of range for positional args tuple
>>>
```

เราสามารถใช้ keyword argument ได้เช่นกัน ดังแสดงในตัวอย่างรูปด้านล่าง

```
>>> #1
>>> "The temperature in {city} is {temp} and air quality is {aqi} AQI".format(city="Bangkok", temp=32.1, aqi=15)
'The temperature in Bangkok is 32.1 and air quality is 15 AQI'
>>> #2
>>> "The temperature in {city} is {temp} and air quality is {aqi} AQI".format(temp=32.1, aqi=15, city="Bangkok")
'The temperature in Bangkok is 32.1 and air quality is 15 AQI'
>>>
```

จะเห็นว่าในตัวอย่างที่ #1 และ #2 keyword city, temp และ aqi จะเรียงลำดับใน argument อย่างไรก็ได้

Format ของ replacement field จะเป็นดังนี้

```
{[<name>] [!<conversion>] [:<format_spec>]}
```

และ format spec มี format ดังนี้

```
: [[<fill>] <align>] [<sign>] [#] [0] [<width>] [<group>] [.<prec>] [<type>]
```

ซึ่งมี component ต่างๆรวมกัน 10 component

ในที่นี่เราจะแสดงตัวอย่างการใช้ format spec บางตัวอย่างที่ใช้บ่อยๆ รายละเอียดอื่นๆสามารถศึกษาเพิ่มเติมได้ที่ <https://realpython.com/python-formatted-output/>

```
>>> #1
>>> "The temperature in {city:s} is {temp:.5f} and air quality is {aqi:d} AQI".format(city="Bangkok", temp=32.1, aqi=15)
'The temperature in Bangkok is 32.10000 and air quality is 15 AQI'
>>> #2
>>> "The temperature in {city:s} is {temp:10.5f} and air quality is {aqi:10d} AQI".format(city="Bangkok", temp=32.1, aqi=15)
'The temperature in Bangkok is 32.10000 and air quality is 15 AQI'
>>> #3
>>> "The temperature in {city:>20s} is {temp:>10.5f} and air quality is {aqi:>10d} AQI".format(city="Bangkok", temp=32.1, aqi=15)
'The temperature in Bangkok is 32.10000 and air quality is 15 AQI'
>>> #4
>>> "The temperature in {city:<20s} is {temp:<10.5f} and air quality is {aqi:<10d} AQI".format(city="Bangkok", temp=32.1, aqi=15)
'The temperature in Bangkok is 32.10000 and air quality is 15 AQI'
>>>
```

จากตัวอย่างด้านบนเครื่องหมาย "<" และ ">" เป็นค่า align แสดงว่าชิดซ้าย (left-aligned) หรือชิดขวา (right-aligned) และเลขทศนิยมสามารถแสดงจำนวนจุดทศนิยมได้ด้วย .<prec> จากตัวอย่างจะให้เห็นแสดงทศนิยมจำนวน 5 หลัก นอกจากนี้ความกว้างของการแสดงผลในแต่ละค่าสามารถใช้ <width> ได้ เช่นในตัวอย่างกำหนดความกว้างของ city ไว้ 20 และความกว้างของ temp และ aqi ได้ที่ 10 ตามลำดับ

วิธีที่ 3: f-strings

วิธีนี้เรียกชื่อเป็นทางการว่า **"Formatted string literal"** หรือเรียกย่อๆว่า **f-strings** วิธีนี้มีมาใหม่ใน Python 3.6 ซึ่งจะมีประสิทธิภาพมากกว่า (เร็วกว่า) 2 วิธีข้างบน และยังอ่านได้ง่ายกว่า (และเขียนสั้นกว่า) 2 วิธีข้างบน ด้วยเช่นกัน

```
>>> #1
>>> "The temperature in {0} is {1} and air quality is {2} AQI".format("Bangkok", 32.1, 15)
'The temperature in Bangkok is 32.1 and air quality is 15 AQI'
>>> #2
>>> "The temperature in {city} is {temp} and air quality is {aqi} AQI".format(city="Bangkok", temp=32.1, aqi=15)
'The temperature in Bangkok is 32.1 and air quality is 15 AQI'
>>> #3
>>> city = "Bangkok"
>>> temp = 32.1
>>> aqi = 15
>>> f"The temperature in {city} is {temp} and air quality is {aqi} AQI"
'The temperature in Bangkok is 32.1 and air quality is 15 AQI'
>>>
```

จากรูปด้านบนจะเห็นว่าในตัวอย่างที่ #3 ซึ่งใช้ f-strings จะเขียนได้สั้นที่สุด และอ่านเข้าใจง่ายที่สุด เมื่อได้มีการกำหนดตัวแปรได้เรียบร้อยแล้ว เพียงแต่มีการใส่ตัวอักษร **f** ข้างหน้าเครื่องหมาย quote "

นอกจากความสามารถในการใส่ชื่อตัวแปรลงใน **replacement fields** แล้ว ยังสามารถทำอื่นๆได้อีกเช่น Arithmetic expressions, objects of composite types, indexing, slicing, key references, function and method calls, conditional expressions และ object attributes รายละเอียดสามารถศึกษาเพิ่มเติมได้ที่

<https://realpython.com/python-formatted-output/>

F-strings มีข้อจำกัดบ้างเล็กน้อยเช่น f-string expression ไม่สามารถเป็นค่าว่าง (empty) ได้ เช่น {} และนอกจากนี้ใน f-string expression ห้ามมีเครื่องหมาย **** (**backslash**) และเครื่องหมาย **#** เพื่อแสดง comment ใน **f-string** ด้วย

การแสดงผลอื่นๆเช่นการขีดซ้าย ขีดขวา ขนาดความกว้าง และจำนวนจุดทศนิยม ของ **f-strings** จะใช้วิธีการเดียวกับ **format.str** (วิธีที่ 2) ยกตัวอย่างดังแสดงในรูปด้านล่าง

```
>>> city = "Bangkok"
>>> temp = 32.1
>>> aqi = 15
>>> f"The temperature in {city:<10s} is {temp:<20.5f} and air quality is {aqi:<10d} AQI"
'The temperature in Bangkok      is 32.10000          and air quality is 15          AQI'
>>>
```

Exercise 10 (Regulation)

ให้ผู้เรียนทำข้อ **Regulation** ใน eJudge โดยทดลองทำทั้ง 3 วิธี (ทั้งวิธี % operator และ str.format และ f-strings)
