Python for Engineering Students (Q1)

บวรทัต เด่นดำรงกุล

1 Input / Output

1.1 Output

ในการ Output ออกมาทางหน้าจอจะใช้คำสั่ง `print()`

```
1 print('Hello, World!')
```

โดยหากใช้คำสั่ง `print` หลายครั้งจะได้คำในคำสั่ง `print` ออกมาคนละบรรทัดกัน

```
1 print('Hello')
2 print('Hello')
```

หากต้องการทำให้อยู่บรรทัดเดียวกันใช้คำสั่ง `end=''` มาต่อท้าย

```
1 print('Hello', end='')
2 print('World')
```

สังเกตว่าหากเปลี่ยนคำใน `''` เป็นคำอื่น ๆ เช่น `'*'` จะได้ `Hello*World` ออกมาแทน

สามารถใช้ , ในการคั่นระหว่างสิ่งที่ต้องการแสดงออกมาได้

```
1 print('Hello', 'World') # Hello World
```

จะได้ `Hello World` แต่หากต้องการใช้ตัวอักษรอื่นคั่นระหว่างแต่ละสิ่งที่ต้องการแสดงสามารใช้ `sep='x'` โดย `x` เปลี่ยนได้ตามต้องการ (โดยปกติแล้วเป็น Space)

```
1 print('Hello', 'World', sep="*") # Hello*World
```

1.2 การ Comment

ในการ Comment ในภาษา Python ใช้เครื่องหมาย # (Sharp, ชาร์ป) ไว้ข้างหน้า

```python
print("hello")
# print("hi")
print("world")
```

1.3 ตัวแปร Variable

ตัวแปร(เบื้องต้น) มีอยู่หลัก ๆ 3 ชนิด คือ

ชนิดของตัวแปร	ชนิดของข้อมูลที่เก็บ
`int`	จำนวนเต็ม
`float`	ทศนิยม
`str`	ตัวอักษร (`string`)

คำสั่ง `type()` สามารถใช้ตรวจสอบชนิดของตัวแปรได้

```python
print(type(3))
print(type("Test"))
print(type(321.123))
```

ในการสร้างตัวแปรในภาษา Python นั้นสามารถเขียนในรูปแบบ

`<VARIABLE_NAME> = <VALUE>`

ตัวอย่างเช่น

```python
a = 10
b = "Hello"
print(a, b) # 10 Hello
```

ซึ่ง Python เป็นภาษาแบบ Dynamic Type จึงสามารถเปลี่ยนชนิดของตัวแปรได้เรื่อย ๆ

หลักการตั้งชื่อตัวแปรสามารถตั้งชื่อได้ด้วยตัวอักษร ตัวเลข หรือเครื่องหมาย _ (underscore) โดยตัวเลขไม่สามารถใช้เป็นตัวแรกได้ และชื่อที่ตั้งต้องไม่ซ้ำกับ reserved words

หลักการใช้ = คือ เอาค่าทางขวามาใส่ตัวแปรทางซ้าย

```python
a = 3
b = 4
print(a, b) # 3 4
a = b
print(a, b) # 4 4
```

1.3.1 Type Casting

ในบางครั้งที่รับประกันได้ว่าสามารถแปลงชนิดของตัวแปรได้สามารถทำการ Type Cast ได้โดยการเอาชนิดที่ต้องการมาครอบ

```
a = "12"
print(type(a)) # <class 'str'>
a = int(a)
print(type(a)) # <class 'int'>
```

1.4 Input

ในการ Input จะใช้คำสั่ง `input()` โดยจะรับมาทีละบรรทัด

```
s = input()
print("Input:", s)
```

1.5 การดำเนินการบนจำนวนชนิดต่าง ๆ

สามารถใช้ตัวดำเนินการมาตรฐานได้เช่น `+ - * / ()` และลำดับการทำงานของตัวดำเนินการทั้งหมดจะทำตามหลักคณิตศาสตร์ (คูณ-หาร/บวก-ลบ จากซ้ายไปขวา)

```
a = 10
b = 3
print(a * b) # 30
```

หมายเหตุ 1. `/` จะเป็นการหารแบบได้ทศนิยมออกมาเสมอไม่ว่าจะเป็น `int` หาร `int` ก็ตาม ♦

การดำเนินการกับตัวแปรเดิมสามารถทำในรูปแบบสั้น ๆ ได้ เช่น

```
a = 10
a += 10 # a = a + 10
a -= 2 # a = a - 2
a *= 4 # a = a * 4
a /= 2 # a = a / 2
```

และมีตัวดำเนินการที่ไม่คุ้นเคยเช่น การยกกำลัง การหารเอาเศษ(Modulo) และ การหารแบบไม่เอาเศษเลย(ปัดเศษลงทั้งหมด)

1.5.1 การยกกำลัง

การยกกำลังใช้ตัวดำนเนินการ **

```
print(10 ** 3) # 1000
```

1.5.2 การหารเอาเศษ Modulo

การหารเอาเศษจะใช้ตัวดำเนินการ %

```
print(10 % 3) # 1
print(10 % 4) # 2
```

1.5.3 การหารแบบไม่เอาเศษและไม่เอาทศนิยม

การหารเอาเศษจะใช้ตัวดำเนินการ //

```
print(10 / 3)  # 3.3333333333333335
print(10 // 3) # 3
```

1.6 Math Module

ในภาษา Python จะมี Math Module ให้เรียกใช้เพื่อให้สามารถใช้คำสั่งทางคณิตศาสตร์ต่าง ๆ ได้ ก่อนการเรียกใช้จำเป็นจะต้อง `import` เข้ามาก่อนด้วยการ

```
import math
```

ตัวอย่างคำสั่งที่น่าสนใจ

```
import math

print(math.sin(0))
print(math.e, math.pi)
print(math.sqrt(9))
print(math.log(4, 2)) # math.log(VALUE, BASE)
```

หมายเหตุ 2. หน่วยของ `math.sin()` และตรีโกณต่าง ๆ เป็น radian หากต้องการแปลงองศาเป็น radian ให้ใช้ `math.radians(degree)` ใส่ใน `math.sin()` ♦

การหารากที่ n ของ x สามารถเขียนในรูป

$$\sqrt[n]{x} = x^{\frac{1}{n}}$$

จึงสามารถใช้การยกกำลังเศษส่วนแทนได้

```
print(1000 ** (1 / 3))
```

2 String & List

2.1 String

String คือสายอักขระ (ตัวแปรประเภท `str`) เช่น “123”, “Hello”

2.1.1 String Operations

การหาความยาวใช้คำสั่ง `len()`

```
s = "Hi"
print(len(s)) # 2
t = "Hello"
print(len(t)) # 5
```

การต่อ String 2 Strings เข้าด้วยกันใช้การบวกได้เลย

```
s = "123" + "321"
print(s) # 123321
```

ในการต่อกันซ้ำ ๆ สามารถใช้คำสั่ง `* n` เมื่อ `n` เป็นจำนวนครั้งได้

```
a = "51" * 10
print(a) # 51515151515151515151
```

2.1.2 String Indexing

String เก็บตัวอักษรหลายตัวก็จริง แต่จริง ๆ แล้ว String สร้างกล่องขึ้นมาจำนวนเท่ากับความยาวเพื่อให้แต่ละกล่องเก็บตัวอักษร 1 ตัวเท่านั้น

เราจึงสามารถใช้ประโยชน์จากสิ่งนี้ในการเรียกแค่ตัวอักษรบางตัวของ String ได้ โดยรูปแบบการเรียกจะใช้

String	P	y	t	h	o	n
Index (Forward)	0	1	2	3	4	5
Index (Backward)	-6	-5	-4	-3	-2	-1

สังเกตว่า Index แบบ Forward จะมีค่าตั้งแต่ 0 ถึง `len(s) - 1` เมื่อ `s` คือสตริงและ Backward มีค่าตั้งแต่ `-len(s)` ถึง `-1`

ในการเรียก Index จะใช้ `[i]` เมื่อ `i` คือ Index ที่ต้องการเรียก

```
a = "Python"
print(a[0], a[-4]) # P t
```

2.1.3 String Slicing

การ Slicing คือการ “หั่น” String ออกมาด้วยวิธีการคล้าย ๆ วน Loop โดยมีรูปแบบการเขียนคือ `[start:stop:step]` โดยการเลือกของ String จะเลือกจาก Index `start` ทุก ๆ `step` ตัวไปจนกว่าจะเท่ากับหรือมากกว่า `stop`

String	P	y	t	h	o	n
Index (Forward)	0	1	2	3	4	5
Index (Backward)	-6	-5	-4	-3	-2	-1

โดยหากเว้น `start` ไว้จะถือว่าเป็น 0, เว้น `stop` ไว้จะถือว่าเป็นขนาดของ String, เว้น `step` ไว้จะถือว่าเป็น 1

```
a = "Python"
print(a[0:3:1]) # Pyt
print(a[1::2]) # yhn
print(a[::3]) # Ph
```

แต่ถ้าค่า `step` ติดลบ หากเว้น `start` ไว้จะถือว่าเป็น `-1`, เว้น `stop` ไว้จะถือว่าเป็นขนาดของ `-len(s)-1` เมื่อ `s` คือ String ที่ต้องการ Slice

```
a = "Python"
print(a[::-1]) # nohtyP
print(a[-1:-4:-2]) # nh
```

2.2 List

List คือกล่องเก็บข้อมูลที่สามารถเก็บข้อมูลหลาย ๆ ชนิดไว้ได้

2.2.1 การสร้าง List

การสร้าง List เปล่าไม่มีอะไรเลย สามารถทำได้โดย

```
l = list()
# or
l = []
```

การสร้าง List แบบมีบางอย่างอยู่ข้างในตั้งแต่แรก สามารถทำได้โดย

```
l = [3, "hi", 12.3]
```

2.2.2 List Operations

คำสั่งส่วนมากคล้าย String

การหาความยาวใช้คำสั่ง `len()`

```
l = [3, "hi", 12.3]
print(len(l)) # 3
```

การต่อ List 2 Lists เข้าด้วยกันใช้การบวกได้เลย

```
s = [3, "hi", 12.3] + [1, "t"]
print(s) # [3, "hi", 12.3, 1, "t"]
```

ในการต่อกันซ้ำ ๆ สามารถใช้คำสั่ง `* n` เมื่อ `n` เป็นจำนวนครั้งได้

```
a = [3] * 5
print(a) # [3, 3, 3, 3, 3]
```

2.2.3 List Indexing

List จะเก็บข้อมูลคล้าย ๆ กับ String โดยจะเก็บเป็นกล่อง ๆ

List	31	“Hello”	123.3	“Test”	3.14	999
Index (Forward)	0	1	2	3	4	5
Index (Backward)	-6	-5	-4	-3	-2	-1

หลักการใช้เหมือนกับ String

```
l = [31, "Hello", 123.3, "Test", 3.14, 99]
print(a[2]) # Hello
```

2.2.4 List Slicing

เหมือนกับ String

2.2.5 การเปลี่ยนค่าข้างใน List

สามารถใช้หลักการของ = ได้เลย

```
l = [31, "Hello", 123.3, "Test", 3.14, 99]
l[1] = -1
print(l) # [31, -1, 123.3, "Test", 3.14, 99]
```

หมายเหตุ 3. ตัวแปรประเภท String ไม่สามารถแก้ไขค่าในช่องใดช่องหนึ่งเหมือน List ได้ ♦

2.2.6 การใส่ค่าเพิ่มใน List

การใส่ค่าเข้าไปเพิ่มใน List จะใช้ method `append()` เพื่อใส่ค่าเข้าไป

```
l = [1, "T"]
l.append('12')
print(l) # [1, 'T', '12']
```

2.3 การแยก String ออกเป็น List

ใช้ method `split()` ของ String เพื่อทำการตัด String ออกมาเป็น List โดยจะตัดด้วย Space โดยจะได้ List of Strings ออกมา

```
s = "Hello World Hi 1234"
l = s.split()
print(l) # ["Hello", "World", "Hi", "1234"]
```

แต่หากต้องการตัดด้วยตัวอักษรอื่น ๆ ให้ใส่ตัวนั้น ๆ ในวงเล็บ

```
s = "Hello*World*Hi"
l = s.split("*")
print(l) # ["Hello", "World", "Hi"]
```

3 Condition

3.1 ประโยคเงื่อนไข

ประโยคเงื่อนไขคือประโยคที่จะคืนค่า `True` หรือ `False` โดย `True` คือ จริง และ `False` คือ เท็จ

ประโยคเงื่อนไขสามารถเขียนได้โดยง่ายด้วย `>` `<` `>=` `<=` แต่หากต้องการตรวจสอบว่าเท่ากับหรือไม่ต้องใช้ `==` ไม่สามารถใช้ `=` ตัวเดียวได้ เนื่องจาก `=` เป็นการเอาค่าทางขวามาใส่ทางซ้าย

ในการตรวจสอบว่าตัวแปรสองตัวมีค่าไม่เท่ากันหรือไม่ใช้ `!=` ในการตรวจสอบ

```
print(1 < 2) # True
print(2 >= 3) # False
a = 2
b = 3
print(a == b) # False
print(a != b) # True
print(a == 2) # True
```

ในใส่นิเสธ การกลับประโยคจากจริงเป็นเท็จ เท็จเป็นจริง หรือการ negation สามารถใช้คำสั่ง `not`

```
a = 1
print(not (a == 1)) # False
print(not (a > 1)) # True
```

ประโยคเงื่อนไขสามารถเขียนเป็นช่วงก็ได้

```
a = 2
print(1 <= a <= 3) # True
print(-1 <= a <= 1) # False
```

ในการตรวจสอบว่าค่านั้น ๆ อยู่ใน List หรือไม่ใช้คำสั่ง `in`

```
l = [1, 2, 3, 4, 5]
print(2 in l) # True
print(7 in l) # False
```

การเปรียบเทียบ String จะเปรียบเทียบด้วยลำดับดังนี้

1. ตัวอักษรตัวพิมพ์ใหญ่มีค่าน้อยกว่าตัวพิมพ์เล็ก
2. ตัวอักษรมีค่ามากน้อยเรียงตามภาษาอังกฤษ
3. หากเป็นตัวเลขมีค่าตามตัวนั้น ๆ
4. จะเปรียบเทียบทีละตัวจากซ้ายไปขวา
5. หากเปรียบเทียนไปเรื่อย ๆ จนมีฝั่งใดฝั่งหนึ่งไม่เหลือตัวอักษรให้พิจารณาแล้วจะให้ตัวที่หมดก่อนน้อยกว่า

```python
print("A" < "C") # True
print("ABC" < "aA") # True
print("bbbb" < "aaaa") # False
```

3.2 การเชื่อมประโยคเงื่อนไข

ในการเชื่อมประโยคเงื่อนไขจะมีการ และ(AND) และ หรือ(OR) โดยหลักการเหมือนตรรกศาสตร์ คือ หากเป็นการ AND จะต้องจริงทั้งคู่ถึงจะจริง และ OR เป็นจริงเพียงฝั่งใดฝั่งหนึ่งก็พอ

หลักการเขียน AND และ OR

```
CONDITION1 and CONDITION2
CONDITION1 or CONDITION2
```

3.3 If

รูปแบบการเขียน If สามารถเขียนได้โดย

```
if CONDITION:
    DO SOMETHING
```

เมื่อ CONDITION คือ เงื่อนไขที่ต้องการ และสังเกตว่าคำสั่งที่ต้องการทำเมื่อเงื่อนไขถูกจะถูก TAB มา 1 ครั้ง

ประโยคเงื่อนไขแบบ If จะทำงานเมื่อเงื่อนไขที่ตั้งไว้ถูก โดยคำสั่งที่ต้องการทำเมื่อใช้คำสั่ง If

```python
A = 3
B = 2
if A > B:
    print("A > B")
```

แต่หากไม่ถูกจะไม่ทำอะไร

3.4 Else

รูปแบบการเขียน Else สามารถเขียนได้โดย

```
if CONDITION:
    DO SOMETHING
else:
    DO SOMETHING
```

สังเกตว่าต้องมี If ขึ้นก่อนเสมอและสำหรับ 1 If มีได้เพียง 1 Else เท่านั้น

หลักการทำงานของ Else คือหากเงื่อนไขใน If ไม่ถูกสามารใช้ Else มาเพื่อรองรับในกรณีที่ไม่ถูกได้

```python
A = 2
B = 3
if A > B:
    print("A > B")
else:
    print("A < B")
```

3.5 Elif

รูปแบบการเขียน Elif (else if) สามารถเขียนได้โดย

```
if CONDITION:
    DO SOMETHING
elif CONDITION2:
    DO SOMETHING
else:
    DO SOMETHING
```

ในการใช้ Elif สามารถใช้ได้หลังมี If เท่านั้นและต้องอยู่ก่อน Else (ถ้ามี)

Else if สามารถมีกี่ตัวก็ได้เหมือนเงื่อนไขรอง ๆ ลงมาจนกว่าจะถึง Else

```
A = 3
B = 3
if A > B:
    print("A > B")
elif A == B:
    print("A is equals to B")
else:
    print("A < B")
```

4 Loops

4.1 While

หลักการทำงานของ Loop While คือ ทำไปเรื่อย ๆ จนกว่าเงื่อนไขจะผิดโดยมีหลักการเขียน คือ

```
while CONDITION:
    DO SOMETHING
```

ตัวอย่างการใช้

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

จากตัวอย่างจะทำการแสดงค่า 1 ถึง 10 บรรทัดละ 1 ตัว

4.2 For

หลักการทำงานของ Loop For จะมี 2 แบบคือใช้กับ `range()` และใช้กับ String หรือ List

4.2.1 การใช้งานกับ Range

ในการใช้งานร่วมกับ `range()` จะใช้หลักการคล้าย ๆ List / String Slicing คือ Start, Stop, Step

```
for i in range(START, STOP, STEP):
    DO SOMETHING
```

โดย `i` คือ ค่าที่ใช้วน Loop For

โดยจะ**ต้อง**กำหนด START และ STOP แต่หากไม่ระบุ STEP จะมีค่า 1 เสมอ และหลักการทำงานเหมือนกันกับ Slicing คือเริ่มจาก START เพิ่มทีละ STEP ไปจนกว่าจะเท่ากับหรือเกิน STOP

```python
for x in range(1, 10, 3):
    print(x)
```

จะแสดงผล 1 4 7 คนละบรรทัดกันเนื่องจากค่าโดดขึ้นทีละ 3 พอค่าเท่ากับ 10 ก็เลิกทำ เนื่องจากเท่ากับ 10 พอดี

```python
for x in range(1, 6, 2):
    print(x)
```

จะแสดงผล 1 3 5 คนละบรรทัดกันเนื่องจากค่าโดดขึ้นทีละ 2 พอค่าเท่ากับ 7 ก็เลิกทำเนื่องจากมากกว่า 6

4.2.2 การใช้งานกับ String หรือ List

ในการใช้งานกับ String หรือ List จะใช้งานในรูปแบ

```
for x in a:
    DO SOMETHING
```

เมื่อ `x` คือค่าที่ได้ในแต่ละครั้ง `a` คือ String หรือ List ที่ต้องการวนตาม

```python
for x in "Hello":
    print(x)
```

จะได้ `Hello` ออกมาบรรทัดละ 1 ตัวอักษร

4.3 Break & Continue

หากต้องการหยุด Loop ที่กำลังทำอยู่สามารถใช้คำสั่ง `break` ได้

```python
i = 1
while True:
```

```
    if i > 10:
        break
    print(i)
    i += 1
```

โปรแกรมข้างต้นจะแสดงค่า 1 ถึง 10 เนื่องจากตอนที่ i เท่ากับ 11 จะเกิดการ break หยุด Loop While นี้ออก

หรือหากต้องการทำ Loop ต่อโดยข้ามขั้นตอนด้านล่างทั้งหมดสามารถใช้ continue ได้

```
i = 1
while i <= 10:
    if i == 8:
        i += 2
        continue
    print(i)
    i += 1
```

ผลที่ได้จะมี 1 2 3 4 5 6 7 10 คนละบรรทัดกันเนื่องจากตอนที่ i เท่ากับ 8 ค่าของ i จะเพิ่มอีก 2 แต่ไม่ทำข้างล่างต่อแล้วกลับไปตรวจสอบเงื่อนไขข้างบนต่อเลย

หมายเหตุ 4. หากมีข้อสงสัยสามารถถามได้ที่ Discord: hydrolyzed ♦