1st edition

Basic Python Courses

Competitive POSNCOM Ver.

Contents

เนื้อหาที่ไม่มี * คือเนื้อหาที่ไม่ออกข้อสอบสอวน.คอมพิวเตอร์

การดาวน์โหลด Compiler สำหรับภาษา Python

1. เลือกใช้ Compiler เป็น thonny สามารถดาวน์โหลดได้ผ่าน www.thonny.org

2. เลือกใช้ compiler เป็น pycharm สามารถดาวน์โหลดได้ผ่าน https://www.jetbrains.com/pycharm/

3. เลือกใช้ Compiler เป็น jupyter notebook (แนะนำ) เนื่องจากสามารถเขียนโปรแกรมเป็น cell แยกกันได้ สามารถดาวน์โหลดผ่าน anaconda และเปิด jupyter notebook

0. โครงสร้างโปรแกรมและการทำงานเบื้องต้น

```
import math

n = input()
print(n)
#Program Break
```

การทำงานของโปรแกรม จะทำงานจากบนลงล่าง, บางคำสั่งเป็นคำสั่งเลือกทำ, บางคำสั่งเป็นคำสั่งวนซ้ำ(Loop), Comment

1. Basic Data types

String คือชนิดตัวแปรประเภทข้อความ ใช้สัญลักษณ์ “string” หรือ ‘string’ ก็ได้

- “Game”
- “123456789”
- “Hello world”
- “”

Number

1. Integer (จำนวนเต็ม) ย่อด้วย int เช่น 1234, 0 เป็นต้น
2. Floating point ย่อด้วย float เช่น 3.14, 0.001 เป็นต้น

หมายเหตุ string + string จะได้ string ต่อกัน, int + string ไม่ได้ ต้องทำการแปลง data type ก่อน

```
print('pixelmath') #pixelmath
print('dew'+'pixelmath') #dewpixelmath
print('dew'+123) #error
```

```
pixelmath
dewpixelmath
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[7], line 3
      1 print('pixelmath') #pixelmath
      2 print('dew'+'pixelmath') #dewpixelmath
----> 3 print('dew'+123)

TypeError: can only concatenate str (not "int") to str
```

จากบรรทัดที่ 1 จะแสดงผลว่า pixelmath, 2 แสดงผลว่า dewpixelmath, 3 จะเกิด error เนื่องจาก data type ไม่เหมือนกัน (string + int)

การแสดงผล print

การแสดงผลสามารถทำได้หลายแบบ เช่น

```
print(1) #1
print(1+2) #3
print('dew' + ' '+ 'pixel') #dew pixel
print(3,2) #3 2
print(f"your name is {'dew'}")
```

```
1
3
dew pixel
3 2
your name is dew
```

หมายเหตุ การเขียนแบบที่ 5 อาจมีประโยชน์ในอนาคต

ตัวแปร (Variables)

- ตัวแปรเป็นทีเก็บข้อมูล
- สามารถเปลี่ยนค่าในตัวเก็บได้
- ประกาศตัวแปรโดย Variable_name = Value

```
nickname = 'Pakdee'
is_num = True
pi = 3.14
a = 1
print(nickname, is_num, pi, a)
```

```
Pakdee True 3.14 1
```

การตั้งชื่อตัวแปร

- ประกอบด้วยตัวอักษร ตัวเลข หรือเครื่องหมายขีดเส้นใต้ได้
- Upper case กับ Lower case ถือว่าแตกต่างกัน
- ห้ามขึ้นต้นชื่อด้วยตัวเลข
- ห้ามเป็นคำสงวน เช่น print, input หรือคำสั่งของโปรแกรม

```
123_nickname = 'Pakdee' #error เนื่องจากมีตัวเลขนำหน้า
print = 12 #error เนื่องจากเป็นคำสงวน
```

Data types Expression

- str ข้อความ
- int จำนวนเต็ม
- float จำนวนจริง มีจุดทศนิยม
- bool ค่าความจริง (True) / (False)
- list ชุดข้อมูลเรียงเป็นลำดับ และเก็บข้อมูลย่อยภายในข้อมูลได้ เช่น

```
cartoon_name = ['Luigi','Donkeykong','Mario','Toad']
money_list = [1, True, 3.14, 'Game']
num = [1,2,3,4,5,6,7]
```

ตัวอย่างการประกาศตัวแปร

```
age = 12 #int
name = 'prachak' #string
check_num = True #boolean
pi = 3.14 #float
birth_date = [12,10,13] #list
cartoon_name = ['Mario', 'Toad'] #list

print(age) #12
print(check_num) #True
print(age+pi) #15.14
```

```
12
True
15.14
```

Type Conversion (การแปลงประเภทของข้อมูล)

- บางข้อมูลสามารถแปลงเป็น string ได้ด้วย str
- บางข้อมูลสามแปลง int กับ float ไปมาได้

```
a = 12
b = 24
print(a+b) #36
print(str(a)+ " "+str(b)) #1224
```

```
36
12 24
```

Operators and input

การรับข้อมูลจะใช้คำสั่ง input() ในการรับค่าเข้ามา หากไม่กำหนด type conversion ของ input ระบบจะรับ string เข้ามาเสมอหากกำหนดชนิดตัวแปรจะดำเนินการได้ดังนี้

```
name = input()
name = str(input()) #เหมือนกับด้านบนเนื่องจากหากไม่กำหนดอะไรเลยจะรับสตริง
age = int(input()) #รับข้อมูลเป็นจำนวนเต็ม
pi = float(input()) #รับข้อมูลเป็นทศนิยม
```

ตัวอย่างโปรแกรมหาพื้นที่สามเหลี่ยม

```
b = float(input()) #รับความยาวฐาน เป็น float
h = float(input()) #รับความสูง เป็น float
area = 1/2 * b * h #คำนวณพื้นที่
print(area) #แสดงผลพื้นที่
print(f"The area is {area}")
print(f"The area is {1/2 * b * h}")
```

```
 7
 12
42.0
The area is 42.0
The area is 42.0
```

Basic Arithmetic Operations

Operator	ความหมาย
+	บวก
-	ลบ
*	คูณ
/	หาร
//	หารปัดเศษทิ้ง (Floor)
%	เศษจากการหาร
**	ยกกำลัง
=	เท่ากับ

Tips การใช้งาน = สามารถกำหนดให้ค่าสองค่าสลับกันได้ เช่น

```
a = 1
b = 2
a,b = b,a
print(a,b)
```

```
2 1
```

Tips การเขียนโปรแกรม

```
a = -a #เปลี่ยนจากลบเป็นบวก
floor_a = int(a) #ปัดเศษลง
decimal = a - int(a) #เอาตัวทศนิยมเท่านั้น
decimal = a%1 #ค่าหลังจุดทศนิยมของ a
unit = a%10 #เลขหลักหน่วยของ a
```

การดำเนินการอัพเดตหรือแก้ไขค่าปัจจุบัน

a = 10	a = 10
a = a + 5	a += 5
a = a – 1	a -= 1

a = a * 2	a *= 2
a = a % 10	a %= 10
a = a **2	a **= 2
a = a//10	a //= 10

Math module

ฟังก์ชัน	คำอธิบาย	ตัวอย่าง
abs(x)	ค่าสัมบูรณ์ของ x	abs(-5) == 5
acos(x)	อาโคซายน์ของ x (หน่วยเป็นเรเดียน)	acos(0.5) == 1.5707963267948966
asin(x)	อาซีนของ x (หน่วยเป็นเรเดียน)	asin(0.5) == 0.523598775598292
atan(x)	อาเทนต์ของ x (หน่วยเป็นเรเดียน)	atan(1) == 0.7853981633974483
atan2(y, x)	อาเทนต์สองตัวแปร โดยพิจารณาถึงสัญญาณของ y และ x	atan2(1, -1) == -1.5707963267948966
ceil(x)	ปัดเศษ x ขึ้นเป็นจำนวนเต็มที่ใกล้ที่สุด	ceil(4.2) == 5
cos(x)	โคไซน์ของ x (หน่วยเป็นเรเดียน)	cos(0) == 1.0
cosh(x)	ไฮเปอร์โบลิกโคไซน์ของ x	cosh(1) == 1.5430801850445917
degrees(x)	แปลงเรเดียนเป็นองศา	degrees(1.5707963267948966) == 90.0
exp(x)	e ยกกำลัง x	exp(1) == 2.7182818284590452
fabs(x)	ค่าสัมบูรณ์ของ x (เหมือนกับ abs(x))	fabs(-5) == 5
floor(x)	ปัดเศษ x ลงเป็นจำนวนเต็มที่ใกล้ที่สุด	floor(4.2) == 4
fmod(x, y)	หาเศษเหลือจากการหาร x หารด้วย y	fmod(5, 2) == 1
frexp(x)	แยกเลขฐานสองของ x ออกเป็น mantissa และ exponent	frexp(10.0) == (1.0, 3)
gamma(x)	ฟังก์ชันแกมมา	gamma(5) == 120.0
gcd(x, y)	หาตัวหารร่วมมากของ x และ y	gcd(18, 8) == 2
hypot(x, y)	หาความยาวของเส้นทแยงมุมจากจุด (0, 0) ไปยังจุด (x, y)	hypot(3, 4) == 5.0
isinf(x)	ตรวจสอบว่า x เป็นค่าอนันต์หรือไม่	isinf(float('inf')) == True
isnan(x)	ตรวจสอบว่า x เป็นค่า NaN หรือไม่	isnan(float('nan')) == True
pow(x, y)	ยก x กำลัง y	pow(2, 3) == 8
radians(x)	แปลงองศาเป็นเรเดียน	radians(90) == 1.5707963267948966
random()	สุ่มตัวเลขทศนิยมระหว่าง 0.0 ถึง 1.0	random()
randrange(start, stop, step)	สุ่มจำนวนเต็มจาก start ถึง stop (ไม่รวม stop) โดยเว้นช่วง step	randrange(0, 10, 2) == [0, 2, 4, 6, 8]
seed(a)	ตั้งค่าค่าเมล็ดพันธุ์สำหรับตัวสร้างตัวเลขสุ่ม	seed(10)

Built-in Function ที่น่าสนใจและเป็นประโยชน์

ฟังก์ชัน	คำอธิบาย	ตัวอย่าง

print(x)	แสดงผลลัพธ์ x บนหน้าจอ	print("Hello, World!")
input(prompt)	รับข้อมูลอินพุตจากผู้ใช้	name = input("What is your name? ")
type(x)	แสดงชนิดข้อมูลของ x	type("Hello") == <class 'str'>
len(x)	หาความยาวของ x (สำหรับสตริง list tuple ฯลฯ)	len("Hello") == 5
range(start, stop, step)	สร้างวัตถุ range ซึ่งสามารถวนซ้ำค่าจาก start ไปยัง stop โดยเว้นช่วง step	for i in range(10): print(i)
list()	สร้างรายการ	x = list([1, 2, 3])
tuple()	สร้าง tuple	x = tuple((1, 2, 3))
set()	สร้าง set	x = set([1, 2, 3])
dict()	สร้าง dictionary	x = dict(name="John", age=30)
if (condition):	ตรวจสอบเงื่อนไขและดำเนินการตามเงื่อนไขนั้น	if age >= 18: print("You are an adult.")
else:	ดำเนินการตามเงื่อนไขที่ตรงกันข้ามกับเงื่อนไข if	else: print("You are a minor.")
for (variable) in (iterable):	วนซ้ำค่าใน iterable	for i in range(10): print(i)
while (condition):	ทำซ้ำบล็อกโค้ดจนกว่าเงื่อนไขจะไม่เป็นจริง	i = 0; while i < 10: print(i); i += 1
def function_name(parameters):	สร้างฟังก์ชัน	def my_function(x): return x * 2
return(value)	ส่งค่าผลลัพธ์จากฟังก์ชัน	result = my_function(5); print(result)

ลำดับในการคำนวณภาษา Python

Python มีลำดับการคำนวณที่กำหนดไว้ชัดเจน โดยจะคำนวณนิพจน์ย่อยจากวงเล็บด้านในสุดไปด้านนอก จากซ้ายไปขวา และทำตามลำดับความสำคัญของตัวดำเนินการ ดังนี้

1. วงเล็บ

- นิพจน์ภายในวงเล็บจะถูกคำนวณก่อนเสมอ
- วงเล็บเหลี่ยม [] วงเล็บปีกกา () และวงเล็บปีกกาคู่ {} มีลำดับความสำคัญเท่ากัน

2. ตัวดำเนินการยกกำลัง

- x**y คำนวณ x ยกกำลัง y

3. การคูณและการหาร

- / คูณก่อน หารทีหลัง
- ตัวดำเนินการเหล่านี้มีลำดับความสำคัญเท่ากัน หมายความว่าจะดำเนินการจากซ้ายไปขวา

4. การบวกและการลบ

- + บวกก่อน ลบทีหลัง
- ตัวดำเนินการเหล่านี้มีลำดับความสำคัญเท่ากัน หมายความว่าจะดำเนินการจากซ้ายไปขวา

ลิสต์ (เบื้องต้น)

ลิสต์ เป็น ประเภทข้อมูลที่สามารถเก็บข้อมูลย่อย ๆ ด้านในสามารถมีข้อมูลได้ทุก data type เช่น bool, int, float, string หรือแม้กระทั่ง List ซ้อนลิสต์ก็สามารถทำได้

```
name = ['Steve', 'Roger', 'James', 'Mark']
age = [1,2,3,4,5,6]
numeric = [3.14, 1.2, 6.7]
Boolean_list = [True, True, False]
all_list = ['Pixel', 5.12, 7, True, [1,2,3]]
nested_list = [[10,2], [65,2], [5,2], [89,6]]
```

การนับจำนวนสมาชิกในลิสต์ สามารถใช้คำสั่ง len()

```
name = ['Steve', 'Roger', 'James', 'Mark']
print(len(name)) #4
age = [1,2,3,4,5]
print(len(age)) #5
nested = [[1,2],[5,6]]
print(len(nested)) #2
```

```
4
5
2
```

ตำแหน่งของ List (List indexing)

```
List_x = [1,'dew',True,3.14,1,2,4,5]
```

Value	1	dew	True	3.14	1	2	4	5
Index	0	1	2	3	4	5	6	7
Index	-8	-7	-6	-5	-4	-3	-2	-1

โดยสามารถเข้าถึงเลข value ได้ผ่านทางเลข index และใช้สัญลักษณ์ [] แสดงค่าของ index นั้น เช่น

```
List_x = [1,'dew',True,3.14,1,2,4,5]
print(List_x[1]) #dew
print(List_x[-1]) #5
print(List_x[2:-1]) #[True, 3.14, 1, 2, 4]
print(List_x[1:5:2]) #['dew', 3.14]
print(len(List_x)) #8
```

สังเกตได้ว่า สามารถใช้ : เพื่อบ่งบอก [start : stop : step] ของการแจงข้อมูลในลิสต์ได้เช่นเดียวกัน

```
a = [1,2,3,4,5,6]
print(max(a), min(a), sum(a))
```

List Method

append()	เพิ่มค่าเข้าไปในลิสต์ (ต่อท้าย)
clear()	เคลียร์ข้อมูลทั้งหมดในลิสต์
copy()	ทำซ้ำ ลิสต์
count()	นับจำนวนข้อมูลที่อยู่ภายในวงเล็บ
index()	ส่งค่า index ของข้อมูลที่อยู่ในวงเล็บ
insert()	แทรกข้อมูลเข้าไปในลิสต์ ที่ index ภายในวงเล็บ
pop()	ลบข้อมูลเลข index ภายในวงเล็บ
remove()	ลบข้อมูล จาก value ที่อยู่ภายในวงเล็บ
reverse()	กลับลำดับของสมาชิกในลิสต์นี้
sort()	เรียงลำดับข้อมูล (default คือ จากน้อยไปมาก)

สตริง

สตริง คือ data type เก็บข้อความ โดยมีการ indexing เหมือนกันกับลิสต์

```
s = "Hello"; t = "World"
x = [1,2,3,4,5]; y = [6,7,8]
print(s+t) #การบวกสตริงจะนำข้อความมาต่อกัน HelloWorld
print(x+y) #การบวกลิสต์คือการนำลิสต์มาต่อกัน [1, 2, 3, 4, 5, 6, 7, 8]
print(len(s), len(s+t), len(x), len(x+y)) #5 10 5 8
print(s[1],x[2], s[-1], s[-2], x[-1], x[-2]) #e 3 o l 5 4
```

```
HelloWorld
[1, 2, 3, 4, 5, 6, 7, 8]
5 10 5 8
e 3 o l 5 4
```

สังเกตว่า การนำ สตริง+สตริง = การนำข้อความมาต่อกัน, ลิสต์ + ลิสต์ = นำลิสต์มาต่อกัน

```
s = "Thailand, Land of smile"
print(s[1])
print(s[1:12])
print(s[1:])
print(s[:1])
print(s[::2])
```

```
h
hailand, La
hailand, Land of smile
T
Taln,Ln fsie
```

ตัวอย่าง โปรแกรมแสดง week day โดย input ตัวเลขระบุวันที่

```
week_day = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
num = int(input())
print(week_day[num-1])
```

```
 6
Sat
```

ตัวอย่าง ใส่ตัวเลข 3 ตัวเข้าไปในลิสต์ แล้วหาผลบวกทั้งหมดในลิสต์โดยใช้คำสั่ง sum()

```
empty_list = []
a = int(input())
b = int(input())
c = int(input())
empty_list.append(a)
empty_list.append(b)
empty_list.append(c)
print(sum(empty_list))
```

```
 1
 2
 3
6
```

ตัวอย่าง โปรแกรม input สินค้าที่จะ shopping

```
shopping_list = []

while True:
  item = input("Enter an item for your shopping list (or 'q' to quit): ")
  if item.lower() == 'q':
    break
  shopping_list.append(item)

print("Your shopping list:")
for item in shopping_list:
  print(item)
```

```
Enter an item for your shopping list (or 'q' to quit):  Shampoo
Enter an item for your shopping list (or 'q' to quit):  Toothbrush
Enter an item for your shopping list (or 'q' to quit):  q
Your shopping list:
Shampoo
Toothbrush
```

ด้านบนถือว่าเป็นโปรแกรมที่สมบูรณ์แล้ว เนื่องจากยังไม่ได้เรียน while loop, for loop ฉะนั้นยังไม่ต้องสนใจ เอาเป็นว่าให้เข้าใจหลักการทำงานของโปรแกรมโดยคร่าว ๆ ก็เพียงพอแล้ว หลักการทำงานโปรแกรมคร่าว ๆ คือ

1. สร้าง list ว่างเพื่อรอการเพิ่มข้อมูล

2. ใช้ while loop รับข้อมูลจนกว่า input จะเป็น ‘q’

3. นำข้อมูลที่รับเข้ามา append ในลิสต์ shopping_list

4. ใช้ for loop แสดงผลทุกสมาชิกในลิสต์ shopping_list

ตัวอย่าง โปรแกรมคำนวณค่าที่มากที่สุดในลิสต์โดยไม่ใช้ max()

```
numbers = [3, 10, 2, 7, 1]

highest_number = numbers[0]

for number in numbers:
  if number > highest_number:
    highest_number = number

print("The highest number in the list is:", highest_number)
```

```
The highest number in the list is: 10
```

โดยหลักการทำงานมีขั้นตอน คือ

1. มีลิสต์ numbers เอาไว้เก็บข้อมูลที่ต้องการดำเนินการ

2. เริ่มต้นกำหนดให้ค่ามากที่สุดเป็นสมาชิกตัวแรกสุดของลิสต์ numbers หรือ numbers[0]

3. วน for loop เพื่อเช็คว่ามี จำนวนใดมากกว่า max หรือไม่ ถ้ามากกว่าก็จะดำเนินการเปลี่ยนค่า max จากเดิมที่เป็น 3 ก็ดำเนินการเปลี่ยนเป็น 10

4. แสดงผลค่า maximum ออกมา

ตัวอย่าง โปรแกรม reverse string จากที่รับค่าเข้ามา

```
def reverse_string(text):
  return text[::-1]

text = input("Enter a string to reverse: ")
reversed_text = reverse_string(text)
print("Original:", text)
print("Reversed:", reversed_text)
```

```
Enter a string to reverse:  Thailand
Original: Thailand
Reversed: dnaliahT
```

หรือเทคนิคในการ reverse string อาจจะใช้ string slicing มาช่วยก็ได้ นั่นคือ ใช้ [::-1] ก็จะให้ผลลัพธ์แบบเดียวกัน

List Comprehension

```
numbers = [x*x for x in range(1, 6)]
print(numbers)
```

```
[1, 4, 9, 16, 25]
```

เขียนลิสต์ Comprehension แบบที่ 1 : สร้างลิสต์ของตัวเลข จากตัวอย่างด้านบน เราสามารถสร้างลิสต์ตามเงื่อนไขที่เขียนต่อท้าย for x in range(1,6) หมายความว่า เมื่อ x มีค่าตั้งแต่ 1,2,3,4 และ 5 อย่าลืม การเขียน range(1,n) จะเป็นค่าตั้งแต่ 1 ถึง n-1 เท่านั้น

```
my_list = [1, 2, 3, 4, 5]
even_numbers = [x for x in my_list if x % 2 == 0]
print(even_numbers)
```

```
[2, 4]
```

เขียนลิสต์ Comprehension แบบที่ 2 : Filtering the list จากตัวอย่างด้านบนพบว่าเรานำลิสต์ที่สร้างไว้ตั้งแต่แรกอยู่แล้วมากรองเอาเฉพาะค่าที่หารด้วย 2 ลงตัว ก็จะได้ผลลัพธ์เป็น 2 และ 4 เท่านั้น

```
name = "Alice"
uppercase_letters = [char.upper() for char in name]
print(uppercase_letters)
```

```
['A', 'L', 'I', 'C', 'E']
```

เขียนลิสต์ Comprehension แบบที่ 3 : การเปลี่ยนแปลงค่า จากตัวอย่างด้านบนพบว่าเราจะเปลี่ยนตัวอักษรพิมพ์เล็ก (Lowercase) ของ name คือ ‘Alice’ เปลี่ยนให้เป็นตัวพิมพ์ใหญ่ทั้งหมดแล้วเก็บเป็น list ชื่อ uppercase_letter

```
rows, cols = 3, 3
multiplication_table = [[x * y for x in range(1, cols + 1)] for y in range(1, rows + 1)]
print(multiplication_table)
```

```
[[1, 2, 3], [2, 4, 6], [3, 6, 9]]
```

เขียนลิสต์ Comprehension แบบที่ 4 : nested list comprehension จากตัวอย่างด้านบนเป็นการสร้าง list ซ้อนใน list อีกที เมื่อ x เริ่มต้นที่ 1 ถึงจำนวน cols, เมื่อ y เริ่มต้นที่ 1 ถึงจำนวน rows ซึ่งบรรทัดแรกของโปรแกรมได้กำหนดค่า rows = 3, cols = 3 มาเรียบร้อยแล้ว หากรันโปรแกรมก็จะได้ผลลัพธ์เป็นตารางสูตรคูณ 3*3

ตัวอย่างการใช้ List Comprehension (น่าจะเป็นประโยชน์)

```
data = [(1, "apple"), (2, "banana"), (3, "cherry")]
fruits = [item[1] for item in data]
print(fruits)
```

จะได้ผลลัพธ์เป็น `['apple', 'banana', 'cherry']` เนื่องจาก access ค่า index เป็น 1 ใน tuple ของ data

```
numbers = [-2, -1, 0, 1, 2]
positives = [num if num > 0 else 0 for num in numbers]
print(positives)
```

จะได้ผลลัพธ์เป็น `[0, 0, 0, 1, 2]` เนื่องจากเป็นการใช้ if-else เข้ามากรอง

```
data = [(1, "apple", True), (2, "banana", False), (3, "cherry", True)]
valid_fruits = [item[1] for item in data if item[2]]
print(valid_fruits)
```

จะได้ผลลัพธ์เป็น `['apple', 'cherry']`

Sorting List (การเรียงลำดับของสมาชิกในลิสต์)

- ใช้ list.sort() ทำให้สมาชิกในลิสต์เรียงลำดับจากน้อย ไป มาก หากต้องการจาก มาก ไป น้อย ให้เขียน script “reverse=True” เข้าไปในวงเล็บของ method .sort
- ใช้ sorted(List) เหมือนกับ .sort แต่สามารถสร้างตัวแปรขึ้นมารับค่าได้
- Sorting List of tuple ให้เขียน key=lambda x : x[0] หรือ x[n] เมื่อ n เป็นตำแหน่งที่เราต้องการ sort

```
numbers = [5, 2, 8, 1, 4]
numbers.sort()
print(numbers) # Output: [1, 2, 4, 5, 8]

numbers.sort(reverse=True)
print(numbers) # Output: [8, 5, 4, 2, 1]
```

```
numbers = [5, 2, 8, 1, 4]
sorted_numbers = sorted(numbers)
print(sorted_numbers) # Output: [1, 2, 4, 5, 8]

sorted_numbers = sorted(numbers, reverse=True)
print(sorted_numbers) # Output: [8, 5, 4, 2, 1]
```

ตัวอย่างการใช้ lambda sorting

```
data =[(3, "apple"), (1, "banana"), (2, "cherry")]
data.sort(key=lambda x: x[1]) # Sort by the second element (fruit name)
print(data)
```

พบว่า การจัดเรียงจะจัดเรียงตามชื่อของผลไม้ จะได้ [(3, 'apple'), (1, 'banana'), (2, 'cherry')] **เนื่องจาก** a มาก่อน b และ b มาก่อน c (จัดเรียงตามพจนานุกรม)

```
data =[(3, "apple"), (1, "banana"), (2, "cherry")]
data.sort(key=lambda x: x[1])
print(data)
data.sort(key=lambda x: x[0])
print(data)
```

สามารถเขียนโดยใช้คำสั่ง sorted(list, __) ได้เช่นเดียวกันหากเราต้องการเก็บไว้ที่ตัวแปรหนึ่ง

```
data =[(3, "apple"), (1, "banana"), (2, "cherry")]
datax =sorted(data, key=lambda x: x[1])
print(datax)
```

List_01 จงเขียนโปรแกรม ตรวจสอบว่าวันไหนขายของขาดทุน โดยกำหนด list ที่มีขนาดเท่ากับ 7 แสดงรายได้ทั้ง 7 วัน ยกตัวอย่างเช่น sells = [550, 700, 300, 200, 570, 304, 450] และเกณฑ์การขาดทุนคือต่ำกว่า 460 บาท พร้อมทั้งแสดงชื่อวันโดยอักษรย่อ

Input: ไม่มี output : ['Wed', 'Thu', 'Sat', 'Sun']

```
sells =[550, 700, 300, 200, 570, 304, 450]
day =['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
count =[day[i] for i in range(len(sells)) if sells[i]<460]
```

tips ใช้ list comprehension (จริง ๆ แล้วเรายังไม่ได้เรียนเรื่อง for loop) แต่เนื่องจาก list comprehension ไม่ได้เข้าใจยากในการหยิบค่านั้น ๆ ในลิสต์ออกมาทีละค่าเท่านั้นเอง

List_02 จงเขียนโปรแกรม เหมือนข้อที่ List_01 แต่เปลี่ยนรูปแบบการแสดงผลเป็น

Input: ไม่มี output : Sat Sun Thu Wed (เรียงตามพจนานุกรมแล้ว)

```
sells =[550, 700, 300, 200, 570, 304, 450]
day =['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
count =[day[i]for i in range(len(sells))if sells[i]<460]
count_sort =sorted(count)
print("".join(count_sort))
```

การเขียนโปรแกรมในข้อ List_02 นี้จะเป็นการประยุกต์ใช้งาน sorted และ .join เพื่อการแสดงผลที่ง่ายขึ้น การ sorted ข้อมูล หลังจากเรียงลำดับตามพจนานุกรมแล้วจะถูกเก็บไว้ที่ตัวแปร count_sort และนำมาแสดงผลโดยการ .join สมาชิกทั้งหมดใน count_sort

List_03 จงเขียนโปรแกรมตรวจสอบ Peak ในโปรแกรม นิยาม Peak คือสมาชิกตัวนั้นมากกว่าตัวก่อนหน้าและตัวถัดไป และแสดงผลเป็นจำนวนของ Peak โดยที่ไม่นับตัวหน้าสุด และ หลังสุด เช่น

Input	- รับค่าตัวเลขที่จะรับข้อมูลต่ออีก n บรรทัด

output	- จำนวนสมาชิกที่เข้าเงื่อนไข peak

Input	Output
1 2 3 4	0
1 5 1	1
1 5 3 5 6 3 2	2

สิ่งที่อาจเป็นประโยชน์ สามารถใช้ for loop หรือ list comprehension มาช่วย, สามารถใช้ฟังก์ชัน .split() เพื่อแยกข้อมูลเป็นรูปแบบลิสต์

```
count=0
x =input().split("")
for i in range(len(x)-2):
 if x[i]<x[i+1]and x[i+1]>x[i+2]:
        count+=1
print(count)
```

จากตัวอย่างที่ให้ไปทั้งสามข้อ สังเกตว่า List มีประโยชน์อย่างมาก

List_04 จงเขียนโปรแกรม reverse ข้อความที่ input เข้าไป

```
s =input()
print(s[::-1])
```

List_05 จงเขียนโปรแกรมหาระยะห่างของจุดสองจุด x1,y1 และ x2,y2 โดยเก็บข้อมูลในลิสต์ซ้อนลิสต์

```
import math
x1,y1 =input().split()
x2,y2 =input().split()
print(math.sqrt((int(x1)-int(x2))**2+(int(y1)-int(y2))**2 ))
```

If-else Condition

If-else เป็นโครงสร้างควบคุมการทำงานที่ใช้ในการตัดสินใจเลือกทางเลือกตามเงื่อนไขที่กำหนด โครงสร้าง if-else ช่วยให้โปรแกรมสามารถทำงานตามเงื่อนไขที่กำหนดไว้ได้

```
if เงื่อนไข:
 # Block of code to be executed if the condition is true
else:
 # Block of code to be executed if the condition is false
```

อย่าลืมว่าจะต้อง indent เยื้องบรรทัดเข้ามา จึงจะบ่งบอกว่าคำสั่งนี้อยู่ในเงื่อนไขที่เขียนขึ้นมา

```
x =10
if x > 5:
```

```
 print("x is greater than 5")
else:
 print("x is not greater than 5")
```

ยกตัวอย่างเงื่อนไข เช่น ถ้า x มากกว่า 5 ให้พิมพ์ “x is greater than 5” แต่ถ้าไม่ใช่ให้พิมพ์ “x is not greater than 5” ซึ่ง x = 10 จะแสดงผลเป็น “x is greater than 5”

```
name ="Alice"
if name =="Alice":
 print("Hello, Alice!")
else:
 print("Hello, stranger!")
```

นอกเหนือจาก if, else แล้วยังมี elif (else if) ด้วย หมายความว่า ถ้าเงื่อนไขด้านบนไม่เป็นจริงก็จะทำตามเงื่อนไข else-if จากบนลงล่างเสมอ

```
number =int(input("Enter a number: "))
if number %2 ==0:
 print(f"The number {number} is even.")
else:
 print(f"The number {number} is odd.")
```

ยกตัวอย่าง ด้านบนเป็นโปรแกรมตรวจสอบว่าตัวเลขที่ input เข้าไปนั้นเป็นจำนวนคู่ หรือจำนวนคี่ หากเป็นจำนวนคู่จะตรงกับเงื่อนไข **if number % 2 == 0:** ซึ่งจะแสดงผล The number ___ is even หรือ ถ้าไม่ใช่ก็ให้แสดงผลเป็น The number ___ is odd

```
score =float(input("Enter your score (0-100): "))
if score >=90:
 print("Your grade is A.")
elif score >=80:
 print("Your grade is B.")
elif score >=70:
 print("Your grade is C.")
elif score >=60:
 print("Your grade is D.")
else:
 print("Your grade is F.")
```

โปรแกรมตัดเกรด การเขียนโปรแกรมตัดเกรดนักเรียนโดยใช้คำสั่ง elif จะทำให้เราง่ายต่อการตรวจสอบเงื่อนไข เช่น หากคะแนนที่ต้องการอยู่ในช่วง 75 ถึง 80 การเขียนเงื่อนไขแบบธรรมดาอาจจะเขียนได้ว่า if score>=75 and score<80 เปลี่ยนไปเขียนแค่ elif score>=75 เนื่องจากหากคำสั่ง if **ไม่เป็นจริง** แล้วจะทำให้เงื่อนไข elif ด้านล่างจะทำงานทันที

นอกจากนี้ ยังสามารถตรวจสอบได้ว่า ข้อมูลนี้ อยู่ใน ลิสต์, dict มั๊ย ด้วยคำสั่ง

```
if value in _____:
```

ยกตัวอย่างเช่น ต้องการตรวจสอบว่า 12 อยู่ในลิสต์ data = [16,18,28,12] หรือไม่ สามารถเขียนคำสั่งได้โดย

```
data =[16,18,28,12]
if 12 in data:
 print("In")
```

```
else:
  print("out")
```

ได้ผลลัพธ์ เป็น In เนื่องจาก 12 อยู่ใน data จริง : แต่อาจจะเขียน if-else ในบรรทัดเดียวกันสั้น ๆ ได้จะประมาณนี้ จะเป็นการช่วยลดบรรทัดในการเขียนได้ (แต่อาจจะไม่เหมาะสำหรับผู้เริ่มต้นเพราะอาจจะมั่วได้)

```
data =[16,18,28,12]
if 12 in data: print("In")
else: print("out")
```

ใน dictionary, set, tuples ก็สามารถทำได้เช่นเดียวกัน

```
data =[16,18,28,12]
data_set ={16,18,28,12}
print(12 in data_set)#True
print(13 in data_set)#False
print(1 in data)#False
```

การวนลูป (Looping)

เป็นโครงสร้างควบคุมการไหลของโปรแกรม (Program Flow Control Structure) ที่ใช้ทำซ้ำชุดคำสั่งโค้ดซ้ำๆ หลายครั้ง

Looping มีสองประเภทหลัก ๆ

- For loop: ใช้สำหรับวนลูปซ้ำๆ แบบที่เราทราบจำนวนรอบที่แน่นอน
- While loop: ใช้สำหรับวนลูปซ้ำๆ ตราบใดที่เงื่อนไขยังเป็นจริง

For Loop แบบที่ 1 : for ตัวแปร in range(ตัวเลข) หมายถึง สำหรับตัวแปรนี้ให้แทนเป็นค่าที่เริ่มต้นด้วยค่าที่กำหนดไว้หลัง range ซึ่ง range ก็มีหลากหลายรูปแบบด้วย (คล้ายๆ กับ slicing ใน List start:stop:step)

```
for ตัวแปร in range(ตัวเลข):
```

```
range(start, stop, step)
```

- start: (ตัวเลือก) กำหนดค่าเริ่มต้นของรายการ ตัวเลขเริ่มต้นที่ 0
- stop: (ตัวเลือก) กำหนดค่าสิ้นสุดของรายการ ตัวเลขจะไม่รวมตัวเลข stop
- step: (ตัวเลือก) กำหนดค่าความถี่ของตัวเลขในรายการ ตัวเลขจะเพิ่มขึ้นทีละ step

```
numbers =range(10)
print(numbers) #0 1 2 3 4 5 6 7 8 9
```

```
numbers =range(1, 11, 2)
print(numbers) #1 3 5 7 9
```

```
numbers =range(10, 0, -1)
print(numbers) # 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
```

สามารถใช้คำสั่ง len(range()) ได้ด้วยเช่นเดียวกัน

```
length =len(range(10))
print(length) #10
```

```
numbers_list =list(range(10))
print(numbers_list)
```

ใช้ list() จะเป็นการเปลี่ยนข้อมูลที่ได้ให้กลายเป็นลิสต์ list(range(10))

```
for number in range(10):
 print(number *number) #0 1 4 9 16 … 81
```

For Loop แบบที่ 2 : for values in list,dict,set,tuples,string: สำหรับทุกค่าในลิสต์, dictionary, tuples ก็จะหยิบค่านั้น ๆ ออกมาทีละตัว เช่น data = [1,2,3,4,5], for x in data: จะไล่ข้อมูลตั้งแต่ x เป็น 1,2,3...,5 เพราะใน data มีข้อมูลสุดแค่ 5 อีกตัวอย่าง vote = {'cartoon':1, 'formula1':1, 'animation':1}, for x in vote: จะไล่ข้อมูลของ keys ทั้งหมดตั้งแต่ x เป็น cartoon, formula1 และ animation, ตัวอย่างสุดท้ายคือ string เช่น str_name = 'Pixelmath' เขียนคำสั่ง for x in str_name: จะได้ไล่ค่า x เป็นตัวอักษรทั้งหมดของ str_name จะได้ P, I, x, e, l, m, a, t, h และการเขียนแบบนี้จะเป็นประโยชน์มากเนื่องจากจะง่ายกว่าการเขียนแบบ index เยอะมาก

```
#ผลบวกทั้งหมดในลิสต์
data =[16,18,28,12]
count =0
for x in data:
    count+=x
print(count)#74
```

```
#ผลบวกทั้งหมดในลิสต์
data =[16,18,28,12]
print(sum(data)) #74
```

```
#ผลบวกทั้งหมด 1-n
n =int(input())
count =0
for x in range(1,n+1):
    count+=x
print(count)
```

วิธีการเขียน รับข้อมูล n เข้ามาเพื่อวนลูปตามจำนวน n ลูปบวกเลขตั้งแต่ 1 ถึง n, สร้างตัวแปร count=0 เพื่อรอการบวกของการวนซ้ำ และนำค่ามาบวกใน count , จบโปรแกรม แสดงผล count

```
data =[]
n =int(input())
for i in range(n):
    x =int(input())
    data.append(x)
print(data)
```

```
3
1
2
3
[1, 2, 3]
```

ตัวอย่างด้านบน แสดงโปรแกรมรับข้อมูล n ตัวเข้า List data

```
even_count =0
odd_count =0
num =input().split()
for numbers in num:
  if int(numbers)%2==0: even_count+=1
  else: odd_count+=1
print(even_count, odd_count)
```

```
1 2 3 4 5
2 3
```

การสร้างตัวแปรยาว ๆ บางครั้งก็อาจจะใช้เวลาเขียนนาน หากชำนาญแล้วสามารถตั้งตัวแปรสั้น ๆ ได้ ยกตัวอย่างเช่น จาก odd_count อาจจะเป็น oc, even_count อาจจะเป็น ec เป็นต้น แต่การตั้งตัวแปรยาว ๆ อาจจะจำง่ายกว่าว่าตัวแปรนี้เก็บค่าอะไร เป็นต้น

While loop

ในภาษา Python การวนลูปเป็นเครื่องมือสำคัญที่ช่วยให้สามารถทำงานซ้ำๆ ได้ โดยไม่ต้องเขียนโค้ดซ้ำๆ หลายๆ ครั้ง หนึ่งในลูปที่นิยมใช้คือ while loop ลูปนี้จะทำงานซ้ำๆ จนกว่าจะมีเงื่อนไขบางอย่างเป็นจริง

```
while เงื่อนไข:
  ชุดคำสั่ง
```

หากเงื่อนไขด้านบนเป็นจริงแล้ว จะทำให้การวนลูปรันต่อไปเรื่อย ๆ จนกว่าจะเป็นเท็จ

```
i = 1
while i <= 5:
    print(i)
    i += 1
```

ตัวอย่าง โปรแกรมแสดงตัวเลข 1,2,3,4 และ 5 : สังเกตได้ว่า เมื่อด้านหลัง Loop เป็นจริง จะทำให้แสดงคำสั่ง print แต่หลังจากจบ Loop ครั้งหนึ่งจะต้องเพิ่มค่า I ไปเรื่อย ๆ ดังนั้นหากค่า i>5 โปรแกรมจะหยุดการวนซ้ำทันที

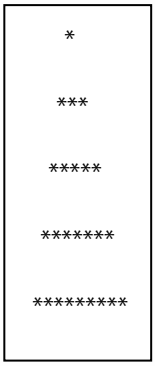
```
sum = 0
number = 1

while number <= 100:
    sum += number
    number += 1

print(f"Summation from 1 to 100 is {sum}")
```

ตัวอย่าง โปรแกรมแสดงผลลัพธ์จากการบวกเลข 1-100 (จริง ๆ แล้วแอบใช้ for loop จะง่ายกว่า)

```
while True:
    user_input = input("กรุณาใส่ข้อความ (พิมพ์ 'exit' เพื่อออก): ")
    if user_input.lower() == 'exit':
       break
    print(f"คุณพิมพ์ว่า: {user_input}")
```

ตัวอย่าง หากเราเขียน while True หมายความว่าโปรแกรมของเราจะวนซ้ำจนกว่าจะมีคำสั่งให้หยุดทำงาน ในที่นี้จะเป็นคำสั่งที่ชื่อว่า break **เพิ่มเติม** การ break loop ใน python นั้นสามารถใช้คำสั่ง “break” เพียว ๆ แบบนี้ได้เลย แต่การ break loop นั้นจะ break เฉพาะลูปนั้นเท่านั้น หากเราเขียนโปรแกรมซ้อนกันหลาย ๆ ชั้นพบว่าโปรแกรมลูปด้านนอกก็ยังทำงานตามปกติ

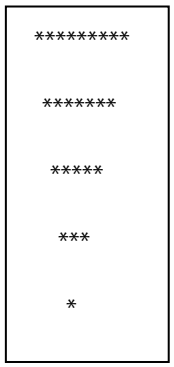
```
rows = 5
i = 1

while i <= rows:
    spaces = ' ' * (rows - i)

    stars = '*' * (2 * i - 1)

    print(spaces + stars)
    i += 1
```



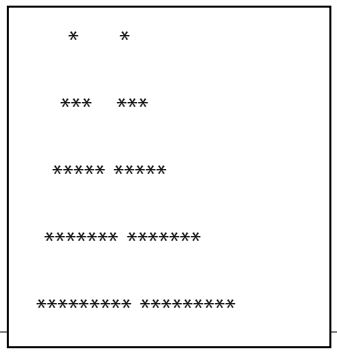
ตัวอย่าง การเขียนโปรแกรมสร้าง pyramid ด้วยภาษา python

สังเกตว่า ข้อดีของภาษา python อีกอย่างหนึ่งคือใช้การเขียน spaces = ' ' * (rows - i) เป็นการเขียนว่าให้ซ้ำตามจำนวนที่เราคูณเข้าไป ซึ่งจะง่ายกว่าภาษาอื่น ๆ อย่าง C, C++ เป็นต้น

```
rows = 5
i = rows

while i >= 1:
    spaces = ' ' * (rows - i)

    stars = '*' * (2 * i - 1)
    print(spaces + stars)

    i -= 1
```




```
rows = 5
i = 1

while i <= rows:
    spaces = ' ' * (rows - i)
    stars = '*' * (2 * i - 1)
    print(spaces + stars + ' ' + stars)
    i += 1
```



```
rows =5
i =1

while i <=rows:
   spaces =' '*(rows -i)
   numbers =''.join(str(x)for x in range(1, 2*i))
    print(spaces +numbers)
   i +=1
```

Dictionary

Dictionary เป็นข้อมูลที่เก็บ key, value ยกตัวอย่างเช่น

```
#Dictionary
student = {'Somsak':12, 'Somying' : 19, "Somchai":21}
print(student['Somsak'])
```

```
12
```

โดยที่หากเราต้องการ value ของ key นั้น ๆ เราสามารถ access ได้โดยการเขียน dict_name[key] จะได้ value ออกมานั่นเอง ยกตัวอย่างเช่น student[“Somsak”] จะได้ 12

```
translations = {
  "hello": "hola",
  "goodbye": "adiós",
  "thank you": "gracias"}

# Get a word from the user
user_word = input()
print(translations[user_word])
```

```
 hello
hola
```

```
translations = {
  "hello": "hola",
  "goodbye": "adiós",
  "thank you": "gracias"}

# Get a word from the user
user_word = input()
print(translations[user_word])
```

```
 goodbye
adiós
```

โปรแกรมแปลภาษา โดยการเก็บ dict ; key เป็นภาษาอังกฤษ, value เป็นภาษาสเปน ดังนั้นหมายความว่า รับข้อมูลเข้ามาเป็นภาษาอังกฤษก็จะแปลเป็นภาษาสเปน โดยใช้ translations[user_word] เพื่อให้แสดงผล value ของ user_word ที่รับเข้ามานั่นเอง นอกจากนี้เรายังสามารถที่จะเพิ่มข้อมูลใน dict ได้เช่นเดียวกัน ด้วยคำสั่ง

```
student = {'Somsak':12, 'Somying' : 19, "Somchai":21}
student["Sommai"] = 67
print(student)
```

```
{'Somsak': 12, 'Somying': 19, 'Somchai': 21, 'Sommai': 67}
```

หากเขียนคำสั่ง student["Sommai"] = 67 ซึ่งชื่อ “Sommai” ไม่มีอยู่ใน dict ดังนั้นโปรแกรมจะ add ข้อมูลนี้เพิ่มเข้าไปใน student ทันที แต่หากมีข้อมูลนั้น ๆ อยู่ใน dict แล้วจะเป็นการอัพเดตค่าของ dict key, value ตัวนั้น ๆ แทน เช่น

```
student = {'Somsak':12, 'Somying' : 19, "Somchai":21}
student["Sommai"] = 67
student["Somsak"] = 18
print(student)
```

```
{'Somsak': 18, 'Somying': 19, 'Somchai': 21, 'Sommai': 67}
```

หรือเราต้องการตรวจสอบว่า ข้อมูลนี้มีอยู่ใน dictionary นี้หรือไม่ ถ้ามีให้ += 1 ถ้าไม่มีให้เพิ่มเป็นข้อมูลใหม่แล้วให้เป็น 1 เช่นโปรแกรมผลโหวตของนักแสดง

```
cartoon = {'Tom and jerry':0, 'Dexter':0, 'Cars':0, "Scooby-doo":0}
while True:
    vote = input()
    if vote=='q': break
    if vote in cartoon: cartoon[vote]+=1
    else: cartoon[vote]=1
print(cartoon)
```

```
 Dexter
 Dexter
 Cars
 Peter-Pan
 Cars
 q
{'Tom and jerry': 0, 'Dexter': 2, 'Cars': 2, 'Scooby-doo': 0, 'Peter-Pan': 1}
```

สังเกตว่า หากเราใส่ Peter-Pan ซึ่งไม่มีอยู่ใน cartoon การเขียนคำสั่ง cartoon[vote]=1 เป็นการบอกว่าให้นำ Key คือ Peter-Pan และ value เท่ากับ 1 ไปใส่ใน dict ชื่อว่า cartoon นั่นเอง สามารถเขียนโค้ดรูปแบบอื่น ๆ ได้ เช่น ให้แสดงผลลัพธ์แบบจัดเรียงสวย ๆ อีกสักนิด

```
cartoon = {'Tom and jerry':0, 'Dexter':0, 'Cars':0, "Scooby-doo":0}
while True:
    vote = input()
    if vote=='q': break
    if vote in cartoon: cartoon[vote]+=1
    else: cartoon[vote]=1
for cartoon_name in cartoon:
    print(f"Movie name {cartoon_name} has {cartoon[cartoon_name]} vote(s)")
```

```
Dexter
Dexter
Peter-Pan
Tom and jerry
Cars
q
Movie name Tom and jerry has 1 vote(s)
Movie name Dexter has 2 vote(s)
Movie name Cars has 1 vote(s)
Movie name Scooby-doo has 0 vote(s)
Movie name Peter-Pan has 1 vote(s)
```

หรืออาจจะเพิ่มความยากขึ้นมาอีกสักนิดนึง ด้วยการแสดงผลลัพธ์ของการ์ตูนที่มีคะแนนโหวตมากที่สุด ในที่นี้หากมีคะแนนซ้ำกันหลายเรื่องก็ให้แสดงผลทุกเรื่อง ยกตัวอย่างเช่น ใน dict มี {‘Tom and jerry’ :2, ‘Dexter’: 1} จะแสดงผลลัพธ์ว่า Tom and jerry เนื่องจากมีคะแนนโหวตสูงที่สุด **คำแนะนำ** อาจจะใช้ลิสต์มาช่วยในการหาค่า max, min ใน value ของ key ได้

```
cartoon = {'Tom and jerry':0, 'Dexter':0, 'Cars':0, "Scooby-doo":0}
max = 0
while True:
    vote = input()
    if vote=='q': break
    if vote in cartoon: cartoon[vote]+=1
    else: cartoon[vote]=1
for cartoon_name in cartoon:
    if cartoon[cartoon_name] > max: max = cartoon[cartoon_name]
for cartoon_name in cartoon:
    if cartoon[cartoon_name] == max:
        print(cartoon_name)
```

```
Dexter
Dexter
Cars
q
Dexter
```

ภาพด้านบนเป็นการเขียนแบบที่ 1 : ไม่ใช้ลิสต์ในการช่วยหา Max

```
cartoon = {'Tom and jerry': 0, 'Dexter': 0, 'Cars': 0, "Scooby-doo": 0}

while True:
    vote = input()
    if vote == 'q':
        break
    if vote in cartoon:
        cartoon[vote] += 1
    else:
        cartoon[vote] = 1

max_vote = max(cartoon.values())
winners = [cartoon_name for cartoon_name, vote_count in cartoon.items() if vote_count == max_vote]

if winners:
    print("The winner(s) is/are:", ", ".join(winners))
else:
    print("There are no winners (all cartoons received 0 votes).")
```

```
Dexter
Dexter
Cars
Cars
q
The winner(s) is/are: Dexter, Cars
```

ภาพด้านบนเป็นการเขียนแบบที่ 2 : มีการใช้ list comprehension ของ winners และมีการใช้คำสั่ง .join(variables) จะเป็นการนำค่าในลิสต์มาต่อกันในรูปแบบสตริง นั่นเอง ซึ่งการเขียนแบบนี้จะเขียนได้รวดเร็ว และง่ายต่อการใช้คำสั่ง แต่ที่สำคัญผู้เริ่มต้นอาจจะเขียนวิธีที่ง่ายกว่านี้ก่อนก็ได้ ตามที่ได้แจ้งไปว่าหากใช้ลิสต์มาช่วยในการหา max ก็ทำได้เช่นเดียวกัน อาจจะเขียนในรูปแบบของ

```
cartoon = {'Tom and jerry': 3, 'Dexter': 2, 'Cars': 1, "Scooby-doo": 0}
print(max(cartoon.values()))
print(min(cartoon.values()))
```

```
3
0
```

หรือสามารถเขียนแบบนี้ก็ได้เช่นเดียวกัน

```
cartoon = {'Tom and jerry': 3, 'Dexter': 2, 'Cars': 1, "Scooby-doo": 0}
l = []
for cartoon_name in cartoon:
    l.append(cartoon[cartoon_name])
print(max(l))
print(min(l))
```

```
3
0
```

การ access ข้อมูลใน dictionary เพิ่มเติม

- Keys() นำ keys มาดำเนินการต่อ
- Values() นำ values มาดำเนินการต่อ
- Items() นำ ทั้ง key และ value มาดำเนินการต่อ

```python
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}

# Print keys
print("Keys:", my_dict.keys())

# Print values
print("Values:", my_dict.values())

# Print items (key-value pairs)
print("Items:", my_dict.items())
```

```
Keys: dict_keys(['name', 'age', 'city'])
Values: dict_values(['Alice', 30, 'New York'])
Items: dict_items([('name', 'Alice'), ('age', 30), ('city', 'New York')])
```

Function (ฟังก์ชัน)

ฟังก์ชันในภาษา python คือส่วนของโปรแกรมที่เรียกใช้ง่าย ๆ เพื่อจัดระเบียบของโปรแกรม โดยรูปแบบเบื้องต้นของ function ใน python หน้าตาเป็นแบบนี้

```
def Hello():
 print("Hello world") #คำสั่งด้านในฟังก์ชัน

Hello() #เรียกใช้งานฟังก์ชัน
```

โดย Hello เป็นชื่อของฟังก์ชันเวลาเราเรียกใช้งาน และในวงเล็บหากเราไม่ใส่อะไรเลยจะเป็นฟังก์ชันที่ไม่ได้ส่งผ่านค่าเข้าไปในฟังก์ชัน ดูในตัวอย่าง ด้านล่าง จะมีการส่งค่าเข้าไปในฟังก์ชันชื่อว่า add และผ่านค่า a และ b เข้าไปในฟังก์ชัน, และค่าที่ออกมาจะเป็น return

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

sum = add(3, 5)
print(sum) # Output: 8
```

Recursive Function (ฟังก์ชันการวนซ้ำ)

```
def factorial(n):
 if n == 0:
  return 1
 else:
  return n * factorial(n - 1) #เรียกใช้ฟังก์ชันก่อนหน้า

print(factorial(5))
```

เนื่องจาก factorial คือ n(n-1)...3.2.1 ดังนั้น ก็หมายถึงเราจะเรียกฟังก์ชันแล้วคูณลงไปเรื่อย ๆ จนถึง 1

```
def apply_function_to_list(func, num_list):
 return [func(num) for num in num_list]

def square(x):
 return x * x

numbers = [1, 2, 3, 4, 5]
squared_numbers = apply_function_to_list(square, numbers)
print(squared_numbers) # Output: [1, 4, 9, 16, 25]
```

เรียกใช้ ฟังก์ชัน ซ้อน ฟังก์ชันได้ **ดังนั้น** สามารถเรียก `apply_function_to_list(square, numbers)` ได้ เพราะฟังก์ชัน square ถูกดึงเข้ามาใส่ใน apply_function_to_list

Func_01 จงเขียนฟังก์ชัน ตรวจสอบจำนวนเฉพาะ

```
def is_prime(n):
 def check_prime(num):
  if num <= 1:
   return False
  for i in range(2, num):
   if num % i == 0:
    return False
  return True

 return check_prime(n)

print(is_prime(7))
print(is_prime(10))
```

ผลลัพธ์ของ is_prime(7) เนื่องจาก 7 เป็นจำนวนเฉพาะ return True -> True

ผลลัพธ์ของ is_prime(10) เนื่องจาก 10 ไม่ใช่จำนวนเฉพาะ return False -> False

Func_02 เขียนฟังก์ชัน add ชื่อภาพยนตร์ และเพิ่มชื่อนักแสดงหากเป็นภาพยนตร์เรื่องเดียวกัน ข้อนี้ค่อนข้างยาก

```
movies = {}

def add_actor(movie_name, actor_name):
 if movie_name in movies:
    movies[movie_name].append(actor_name) # Add to existing list of actors
 else:
    movies[movie_name] = [actor_name] # Create a new list for the movie

add_actor("The Shawshank Redemption", "Tim Robbins")
add_actor("The Shawshank Redemption", "Morgan Freeman")
add_actor("The Godfather", "Marlon Brando")
add_actor("The Godfather", "Al Pacino")
add_actor("The Dark Knight", "Christian Bale")
add_actor("The Dark Knight", "Heath Ledger")
print(movies)
```

```
{'The Shawshank Redemption': ['Tim Robbins', 'Morgan Freeman'], 'The Godfather':
['Marlon Brando', 'Al Pacino'], 'The Dark Knight': ['Christian Bale', 'Heath
Ledger']}
```

โปรแกรมเดียวกัน แต่เก็บ values ในรูปแบบ set แทน List

```
movies = {}

def add_actor(movie_name, actor_name):
 if movie_name in movies:
    movies[movie_name].add(actor_name)
 else:
```

```
    movies[movie_name] = set([actor_name])

add_actor("The Shawshank Redemption", "Tim Robbins")
add_actor("The Shawshank Redemption", "Morgan Freeman")
add_actor("The Godfather", "Marlon Brando")
add_actor("The Godfather", "Al Pacino")
add_actor("The Dark Knight", "Christian Bale")
add_actor("The Dark Knight", "Heath Ledger")
add_actor("The Dark Knight", "Heath Ledger")
print(movies)
```

```
{'The Shawshank Redemption': {'Tim Robbins', 'Morgan Freeman'}, 'The Godfather': {'Marlon Brando', 'Al Pacino'}, 'The Dark Knight': {'Christian Bale', 'Heath Ledger'}}
```

ข้อสังเกต การเพิ่มข้อมูลเข้าไปใน set จะใช้คำสั่ง .add() ซึ่งต่างกับลิสต์ที่ใช้คำสั่ง append() การเขียนโปรแกรมในรูปแบบนี้สามารถเขียนได้หลากหลายวิธี สามารถสร้างลิสต์ว่างเปล่าขึ้นมาก่อน -> append เข้าลิสต์ว่าง -> append เข้า values ของ keys ก็สามารถทำได้เช่นเดียวกัน

```
movies = {}

def add_actor(movie_name, actor_name):
 if actor_name in movies:
        movies[actor_name].add(movie_name)
 else:
        movies[actor_name] = set([movie_name])
add_actor("The Shawshank Redemption", "Tim Robbins")
add_actor("The Shawshank Redemption", "Morgan Freeman")
add_actor("The Godfather", "Marlon Brando")
add_actor("The Godfather", "Al Pacino")
add_actor("The Dark Knight", "Christian Bale")
add_actor("The Dark Knight", "Heath Ledger")
print(movies)
```

```
{'Tim Robbins': {'The Shawshank Redemption'}, 'Morgan Freeman': {'The Shawshank Redemption'}, 'Marlon Brando': {'The Godfather'}, 'Al Pacino': {'The Godfather'}, 'Christian Bale': {'The Dark Knight'}, 'Heath Ledger': {'The Dark Knight'}}
```