

Python 3.4 บน Microsoft Visual Studio

วัตถุประสงค์

1. เรียนวิธีใช้ **Microsoft Visual Studio** 2015 เพื่อใช้ภาษา Python Version 3.4
2. เรียนภาษา Python อย่างรวดเร็วเพื่อใช้ในวิชา 01076249 Data Structures & Algorithms เมื่อมีพื้นฐาน programming ภาษา C และ Java มาแล้ว โดยให้ศึกษาจากตัวอย่างสั้นๆ และทำการรัน step ซึ่งจะได้ศึกษา:

การทดลองที่ 1 :

- | | |
|--|----|
| 1.1. ใช้ MS Visual Studio เปิดหน้าต่าง interactive window | 2 |
| 1.2. type() | 4 |
| 1.3. identifier (Name) | 4 |
| 1.4. Numbers Types & Operations | 5 |
| 1.5. Assignment Statement | 5 |
| 1.6. Import Statement | 6 |
| 1.7. Multiple Assignments | 7 |
| 1.8. String, Repetition, Concatenation, Indexing, len(), slicing | 7 |
| 1.9. List, Indexing, slicing, Repetition, Concatenation, len(), append(), Nested List, List Operations | 10 |

การทดลองที่ 2 : Open Python Project บน MS Visual Studio เบื้องต้น รันโปรแกรมแบบต่างๆ <F5> <Ctrl+F5> <F8> (<F11>) <Ctrl+F8> (<F10>) หน้าต่าง Auto หน้าต่าง Local หน้าต่าง Watch

การทดลองที่ 3 : Control Flow

- | | |
|---|----|
| 3.1. if | 22 |
| 3.2. while | 24 |
| 3.3. range() | 24 |
| 3.4. for | 25 |
| 3.5. break, continue, else clauses on loops | 25 |
| 3.6. pass | 26 |
| 3.7. Function Definition | 26 |
| 3.8. Return Value(s) จาก Function | 26 |

การทดลองที่ 4 : แบบฝึกหัด

- | | |
|--------------------------------------|----|
| 4.1. def factorial(n): | 27 |
| 4.2. def multiples_of_3_and_5(n): | 27 |
| 4.3. def integer_right_triangles(p): | 27 |
| 4.4. def gen_pattern(chars): | 27 |

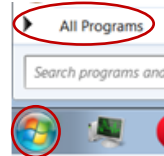
1. การทดลองที่ 1 :

1.1. ใช้ MS Visual Studio เปิดหน้าต่าง interactive window

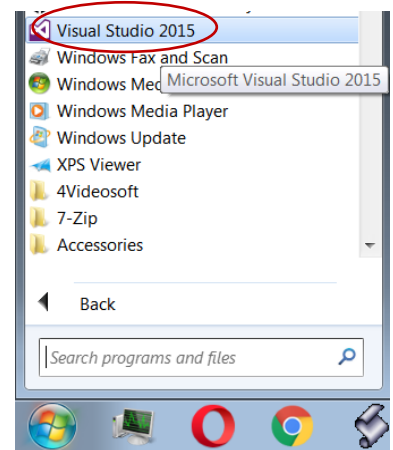


คลิก icon

หรือ

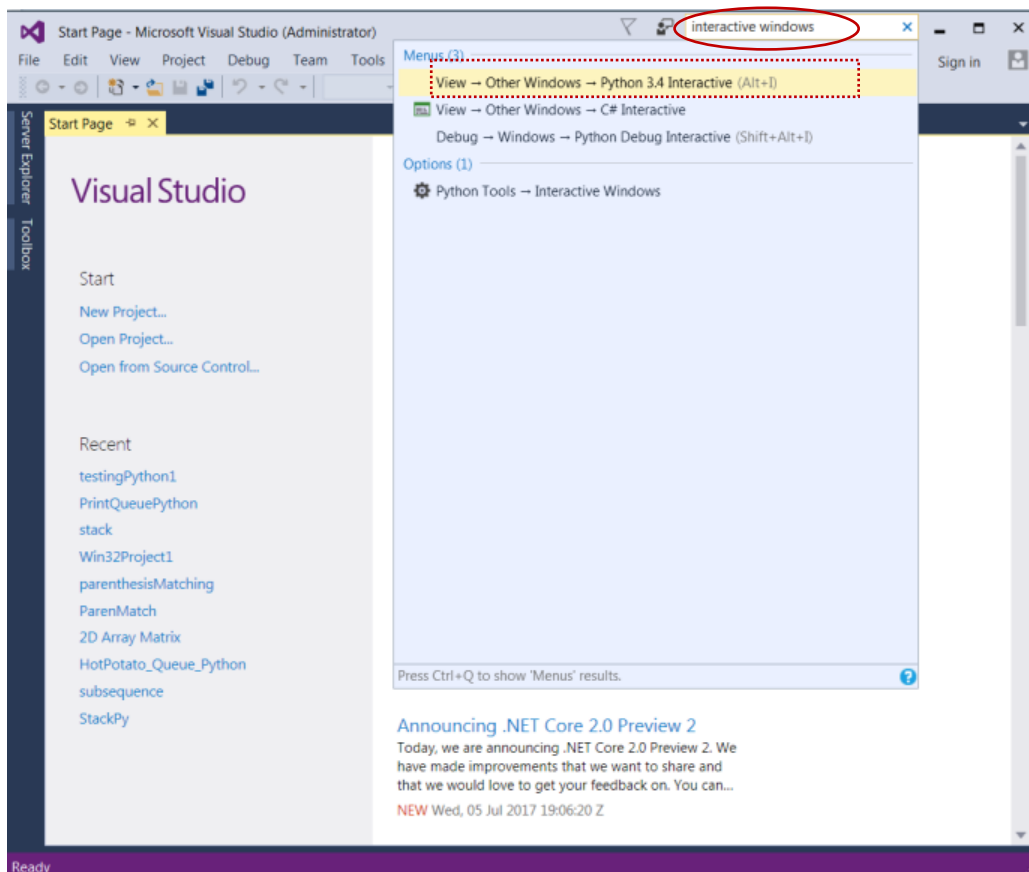


คลิก All Programs



คลิก Visual Studio 2015

Start Page

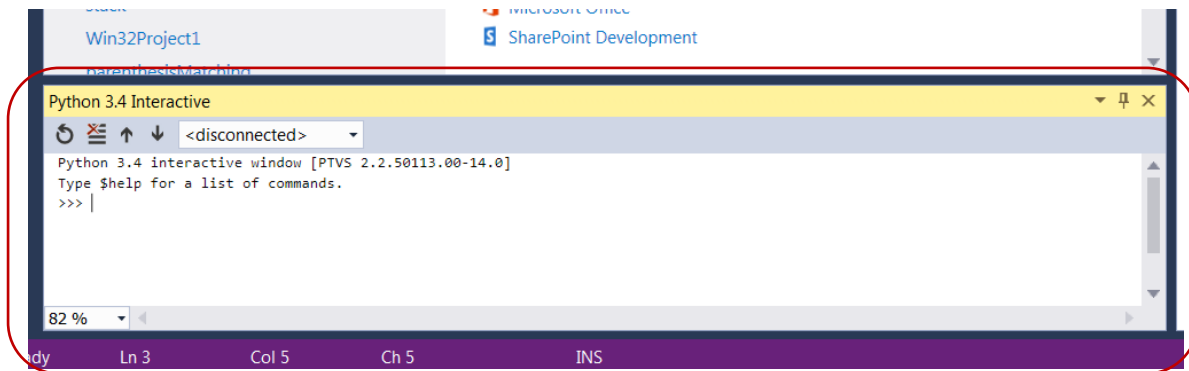


ที่ Quick Launch :

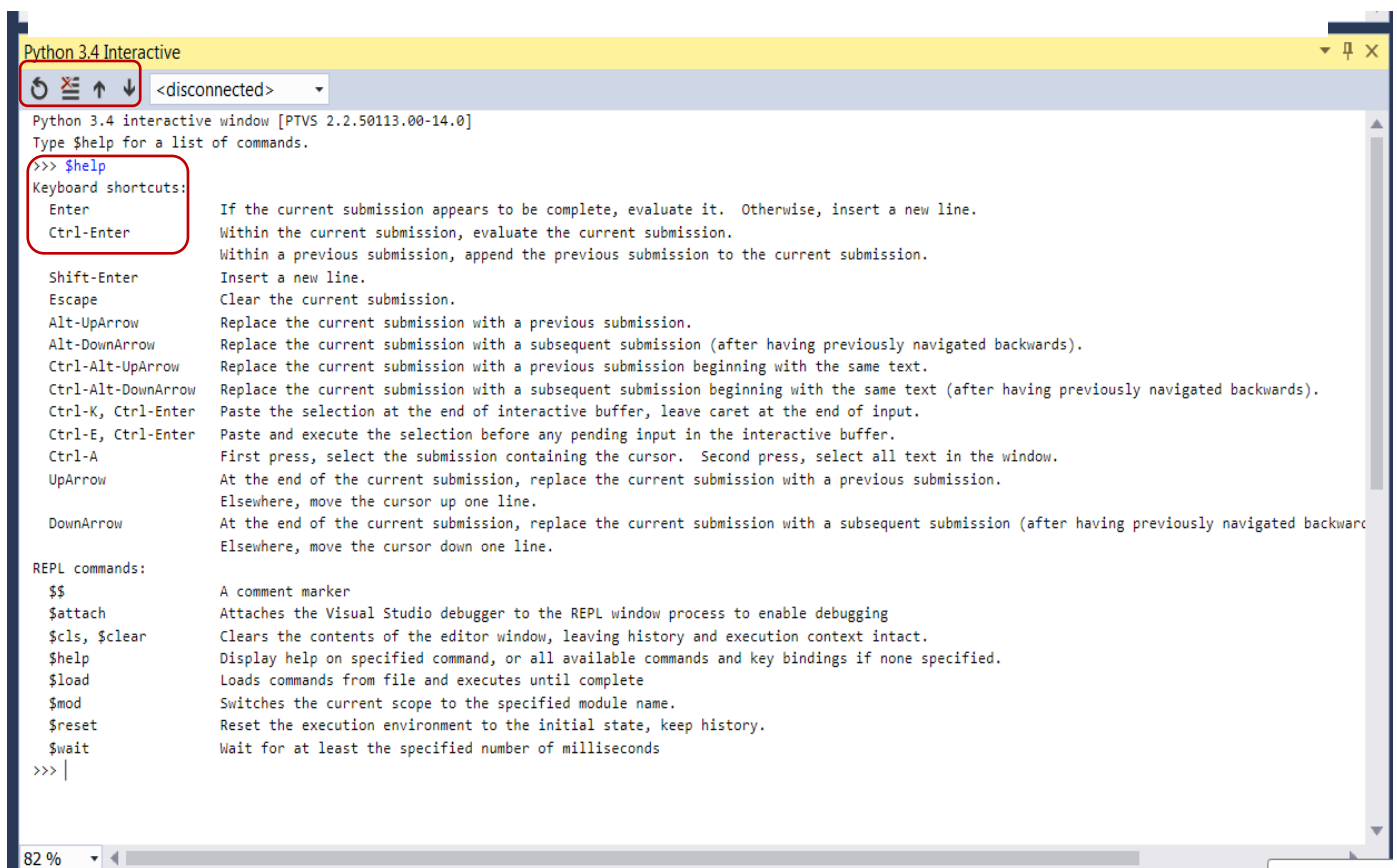
พิมพ์ interactive window
แล้วเลือก

Python 3.4 Interactive

Interactive
Window
&
prompt
>>>



- จะได้หน้าต่าง interactive แสดง version ของ Python พร้อม prompt >>> เพื่อรอรับ Python statement
- พิมพ์ \$help แล้วกด <Enter> ขยายหน้าต่าง อ่านคำสั่งต่างๆ



1.2. type() : Python เป็น OOP type เป็น object ทดลองทำคำสั่งที่เรียนใน lecture ดู output ที่ได้

```
>>> s = 'Hi'                # string in single or double quotes
>>> i = 5                    # float
>>> f = 3.2                  # int
>>> print(s, i, f, 'try')

>>> type(s)

>>> type('Hi')

>>> type(i)

>>> type(f)

>>> i                        #interactive mode ไม่ต้องใช้คำสั่ง print(i) ก็พิมพ์ค่าให้
```

1.3. identifier (Name) ชื่อ สามารถประกอบด้วย a to z, A to Z, 0 to 9, _ แต่ต้องไม่ขึ้นต้นด้วยตัวเลข และไม่เป็น keywords

Python Keywords										
False	and	break	def	else	for	if	is	not	raise	while
None	as	class	del	except	from	import	lamda	or	return	with
True	assert	continue	elif	finally	global	in	nonlocal	pass	try	yield

1.4. Numbers Types & Operations

ใช้ operators ต่างๆ ได้เหมือนในภาษาอื่น วงเล็บเพื่อทำการจัดกลุ่มที่ทำก่อน ทดลองใส่ expression ต่างๆ เอง โดยศึกษาจากตัวอย่างข้างล่าง

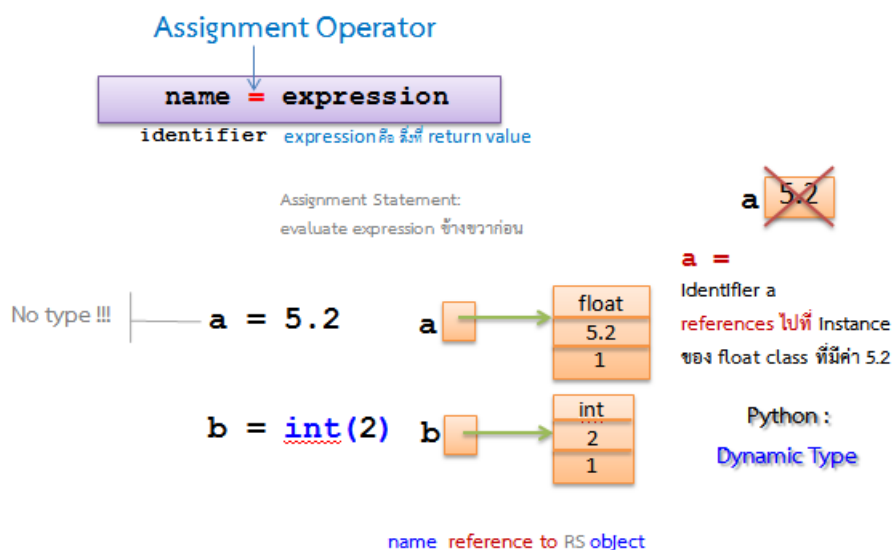
```
>>> 5/2
2.5
>>> 5//2
2
>>> 5%2
1
>>> 2**3
8
>>> 3.5 - 2
1.5
```

lowest precedence

highest precedence

	output	
5 + 3	8	
5 - 3	2	
5 * 3	15	
5 / 3	1.6666666666666667	
5 // 3	1	div
5 % 3	2	mod
-5	-5	
+5	5	
abs(-5)	5	absolute
int(5.2)	5	int conversion
float(5)	5.00	float conversion
divmod(5,3)	(1,2)	divmod pair
pow(2, 3)	8	
2 ** 3	8	

1.5. Assignment Statement



C : int a = 5.2; // a เก็บ data 5.2 ไว้
 // จึงต้อง define type ของ a เพราะขนาด memory ขึ้นกับ type

Python: a = 5.2 # a เก็บ address ของ object 5.2 ไว้
 # a reference ไปที่ 5.2

ทดลอง assignment statement และ ทดสอบว่า name reference ไปที่ object เดียวกันหรือไม่จากฟังก์ชัน id()
หรือ operators: is หรือ is not

```

>>> a = 5.2
>>> b = int(2)
>>> a is b

>>> print(a, b, id(a), id(b))

>>> a = b
>>> print(a, b, id(a), id(b))

>>> a is b

```

1.6. Import Statement

math module : file ที่รวบรวม math functions ต่างๆ สามารถใช้ได้เมื่อ import



```

>>> import math
>>> print(math)
<module 'math' (built-in)>
>>> print(math.pi)
3.141592653589793
>>> math.cos(math.pi)
-1.0
>>> math.log2(8)
3.0
>>> math.sqrt(16)
4.0
>>> math.trunc(3.67)
3

```


```

>>> from math import pi
>>> print(pi)
3.141592653589793
>>> from math import *
>>> cos(pi)
-1.0
>>> pi = 5
>>> cos(pi)
0.28366218546322625

```



1.7. Multiple Assignments ทำได้ในบรรทัดเดียว



```

>>> a, b, c = 1, 3.5, 'Hello'
>>> print(a, b, c)
1 3.5 Hello

```

```

>>> i = j = k = 'same'
>>> id(i)
37565440
>>> id(j)
37565440
>>> id(k)
37565440

```

ลำดับการ evaluate จะทำตามลำดับเลขที่อยู่ข้างหลัง expression

`expr3, expr4 = expr1, expr2`

code ข้างล่าง สลับค่า a และ b

```

>>> a = 5
>>> b = 10
>>> a, b = b, a
>>> print(a,b)
10 5

```

code ข้างล่าง ค่า `x[1]` เป็น 2 เพราะ `i = 1` ถูก evaluate ก่อน `x[i] = 2` ดังนั้นจึงเป็น `x[1] = 2`

```

>>> x = [0, 1]
>>> i = 0
>>> i, x[i] = 1, 2
>>> print(x)
[0, 2]

```

1.8. String

String คือ type ที่เก็บ character เรียงลำดับกัน Python สามารถจัดการกับ string ได้หลายแบบ

1.8.1. single quotes หรือ double quotes ใช้ คร่อม string ให้ผลเหมือนกัน

```

>>> 'string'
'string'
>>> "This is also string"
'This is also string'

```

1.8.2. Repetition (*) และ Concatenation (+) Operators

```
>>> 3*'aa' + 'bcd'
'aaaaaabcd'
>>> str = 'x'
>>> 6*str
'xxxxxx'
```

1.8.3.String Literals ขีดกัน จะทำ concatenation โดยอัตโนมัติ แต่ variable ไม่ทำ ต้องใช้ +

```
>>> '1234'+'5678'
'12345678'
>>> s2 = '1234'
>>> s2 + '5678'
'12345678'
```

มีประโยชน์มากในการแบ่ง string ยาวๆ

```
>>> longStr = 'kasdfjkas'
>>> longStr = ('long .....
...         'still .....
...         'finally.....')
>>> longStr
'long .....still .....finally.....'
```

1.8.4.String Indexing (subscript)

character ตัวแรก ของ string มี index 0 และ ตัวต่อไปเป็น 1 ...

index เป็นค่า - ได้ไล่จากตัวสุดท้ายมาตัวแรก โดย ตัวสุดท้ายมี index -1 (-0 ไม่ได้เพราะเท่ากับ 0) รองสุดท้าย -2

index เกินช่วง (range) จะเกิด error

```
>>> s = '0123456789'
>>> s[1]
'1'
>>> s[-1]
'9'
>>> s[-2]
'8'
>>> s[20]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```


1.8.5. len() return String Length

```
>>> s = '0123456789'
>>> len(s)
10
```

1.8.6.String Slicing

สามารถสร้าง substring จาก string ได้โดยใช้รูปแบบ a:b:c โดย

- a คือ index ตัวตั้งต้น (ไม่ใช่ หมายถึง ตัวแรก)
- b " ก่อนตัวสุดท้าย (ไม่ใช่ หมายถึง ความยาวของ string)
- c " จำนวนที่ตัดไปที่ตัว (ไม่ใช่ หมายถึง 1)

```
>>> s = '0123456789'
>>> s[1:3]
'12'
>>> s[2:9:2]
'2468'
>>> s[:3]
'012'
>>> s[2:]
'23456789'
>>> s[-7:8]
'34567'
>>> 'Hello' + s[-7:8]
'Hello34567'
```

1.8.7.String เป็น immutable type

เช่นเดียวกับ type int และ float ค่า (object) ที่เป็น string เปลี่ยนไม่ได้ เรียกว่า immutable

```
>>> s = '0123456789'
>>> s[0] = 'a'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

1.8.8. String Operations

1.9. List

List เป็น type ที่เก็บค่าได้หลายค่าที่เอนกประสงค์ที่สุดของ Python ใช้เครื่องหมาย [] ภายในเป็น element ของ list คั่นกันด้วยจุลภาค , element แต่ละตัวอาจเป็นคนละ type ก็ได้ (โดยมากเป็น type เดียวกัน)

```
>>> [1,2,3,4]
[1, 2, 3, 4]
>>> list = ['Hello', 3, 3.5]
>>> print(list)
['Hello', 3, 3.5]
```

1.9.1. Indexing (subscript) และ Slicing

เช่นเดียวกับ string และ type อื่นๆ ที่เป็น built-in sequence type

List ทำ index และ slicing ได้

```
>>> list = ['Hello', 3, 3.5]
>>> list[0]
'Hello'
>>> list[-1]
3.5
```

แต่ผิดกับ string int และ float, list เป็น mutable type

```
>>> list = ['Hello', 3, 3.5]
>>> list[0] = 1
>>> list
[1, 3, 3.5]
```

```
>>> l = [0,1,2,3,4,5,6,7,8,9]
>>> l[1:9:2]
[1, 3, 5, 7]
>>> l[-7:len(l):2]
[3, 5, 7, 9]
>>> l[:]
[0,1,2,3,4,5,6,7,8,9]
>>> l[1:9:2] = ['a','b','c','d']
>>> l
[0, 'a', 2, 'b', 4, 'c', 6, 'd', 8, 9]
>>> l[1:5] = [] # empty list
>>> l
[0, 'c', 6, 'd', 8, 9]
```

1.9.2. Repeatition (*) Concatenation (+) Operators และ len()

```
>>> li = [1,2]
>>> lis = [3,4,5]
>>> 2*li + lis
[1, 2, 1, 2, 3, 4, 5]
>>> len(li)
2
```

1.9.3. append() เป็น list method

listVariable.append(i) เพิ่ม element i ท้าย list

```
>>> li
[1, 2]
>>> li.append(3)
>>> li
[1, 2, 3]
```

1.9.4. Nested List

list สามารถซ้อนกันได้

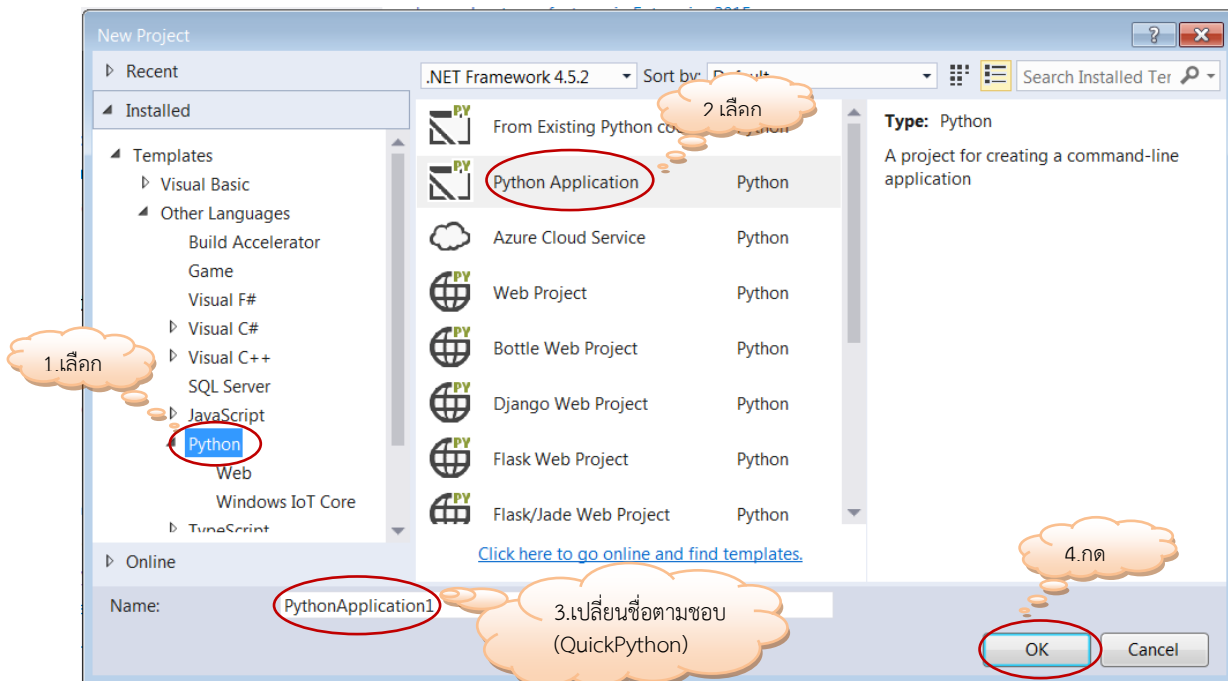
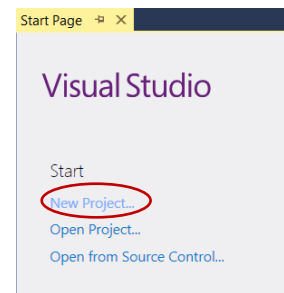
```
>>> li
[1, 2, 3]
>>> li.append([3,4])
>>> li
[1, 2, 3, [3, 4]]
>>> li[3]
[3, 4]
>>> li[3][1]
4
```

1.9.5.List Operations

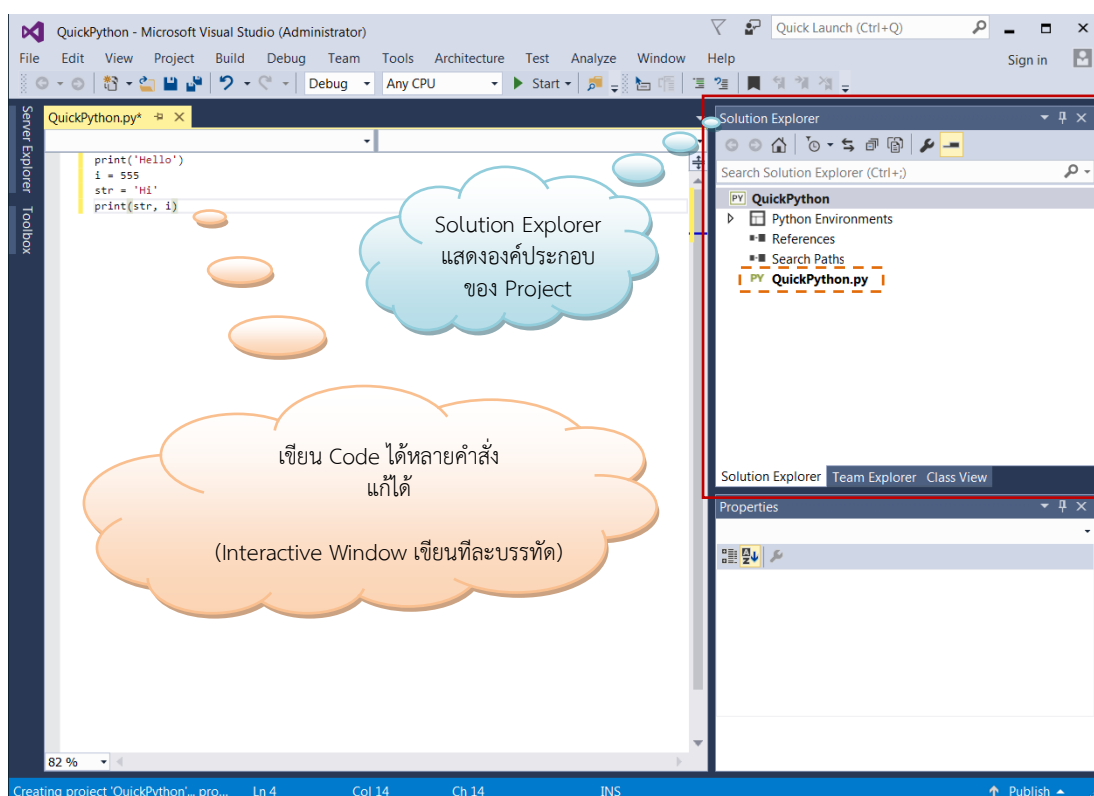
L = [1, 3, 7, 3]		
methods	ผลลัพธ์	คำอธิบาย
<code>len(L)</code>	4	จำนวนของใน list
<code>max(L)</code>	7	หา max item, ต้องเป็นไทป์เดียวกัน
<code>min(L)</code>	1	หา min item, ต้องเป็นไทป์เดียวกัน
<code>sum(L)</code>	14	หา sum ของ item, ต้องเป็น number
<code>L.count(3)</code>	2	นับจำนวน 3
<code>L.index(7)</code>	2	หา index ของ 7 ตัวแรก
<code>L.reverse()</code>	[3, 7, 3, 1]	กลับลำดับของของ
<code>L.clear()</code>	[]	ทำให้เป็น empty list
<code>L.append(5)</code>	[1, 3, 7, 3, 5]	insert object ที่ท้าย list
<code>L.extend([6,7])</code>	[1, 3, 7, 3, 6, 7]	insert list ที่ท้าย list
<code>del L[1]</code>	[1, 7, 3]	remove item index 1
<code>L.remove(3)</code>	[1, 7, 3]	remove item แรกที่มีค่า = 3
<code>L.insert(1, "Hi")</code>	[1, "Hi", 3, 7, 3]	insert new item แทรกที่ index ที่กำหนด
<code>L.pop(0)</code>	[3, 7, 3]	remove & return item index 0 , ไม่ใช่ index คือตัวล่าสุด

2. การทดลองที่ 2 : Open Python Project บน MS Visual Studio

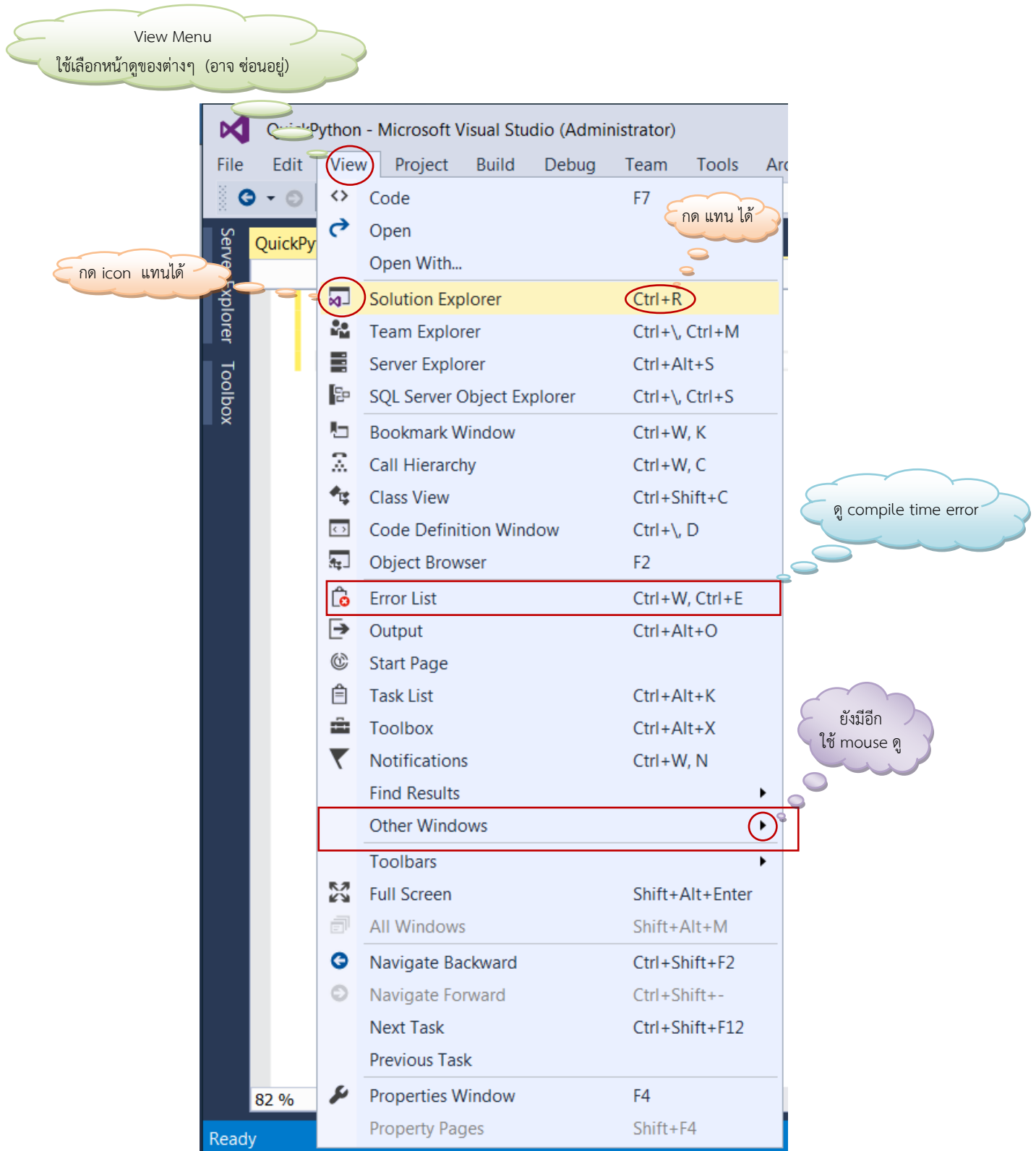
2.1. ที่ Start Page กด <New Project>



New Project Window : เลือกดังรูป เปลี่ยนชื่อ application ตามชอบ (ตัวอย่างใช้ QuickPython) กด ok

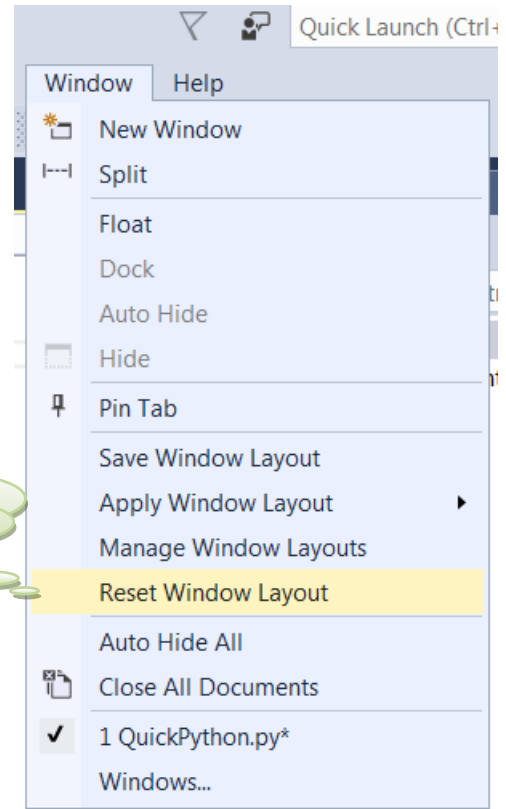


หน้าต่างต่างๆ : สามารถ เลื่อน ย่อ/ขยาย และ ปิด (ซ่อน) ได้



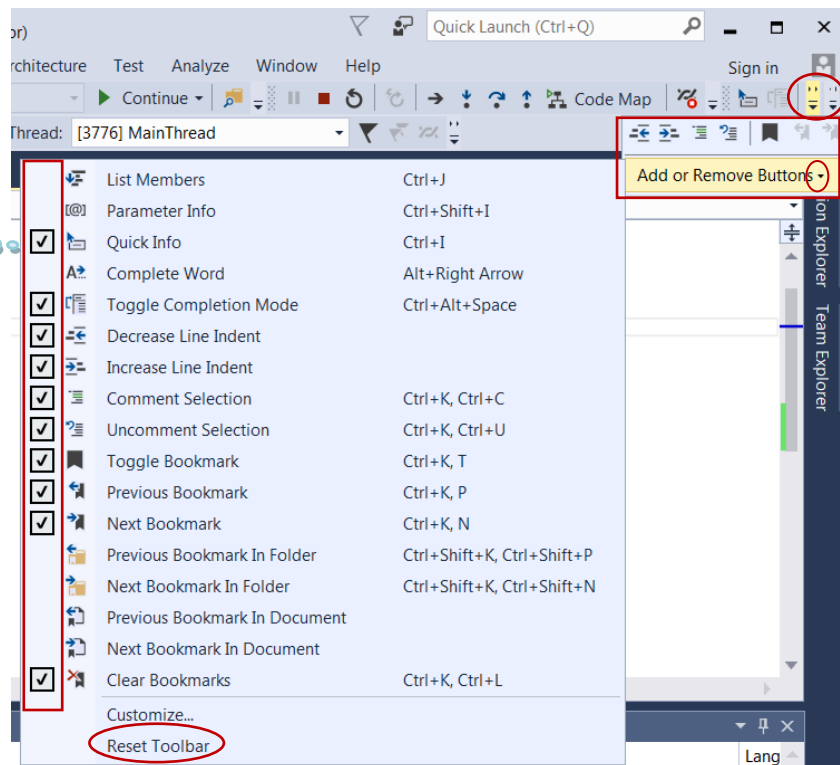
การ reset window layout : Menu Window

จัดรูปแบบหน้าต่างให้เป็นแบบดั้งเดิม
ได้หน้าต่าง Solution Explorer ด้วย



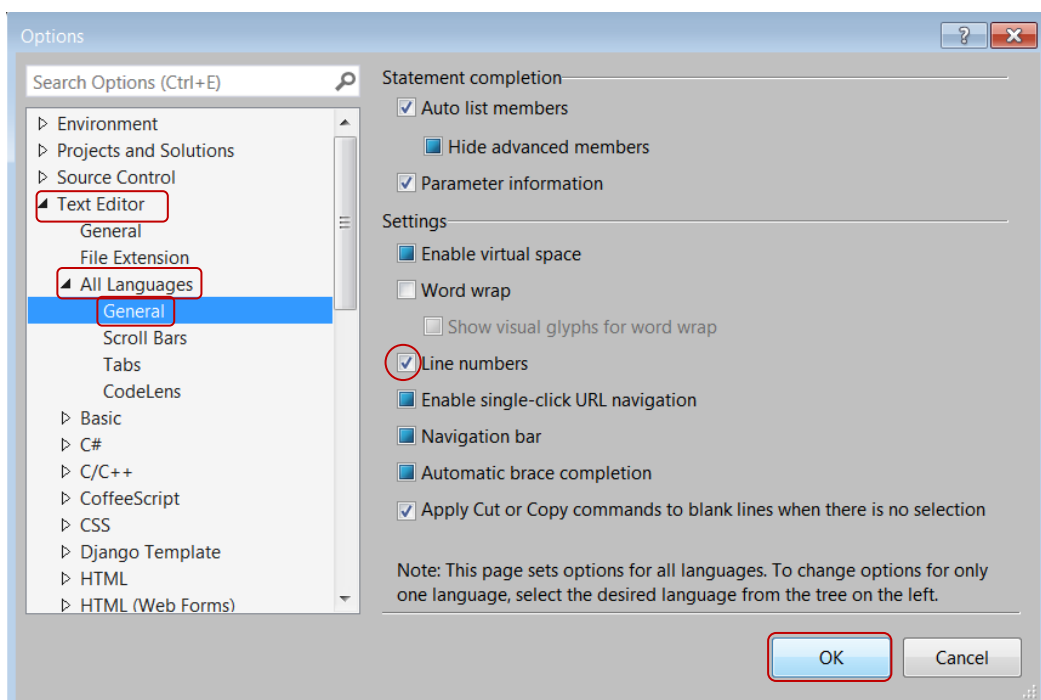
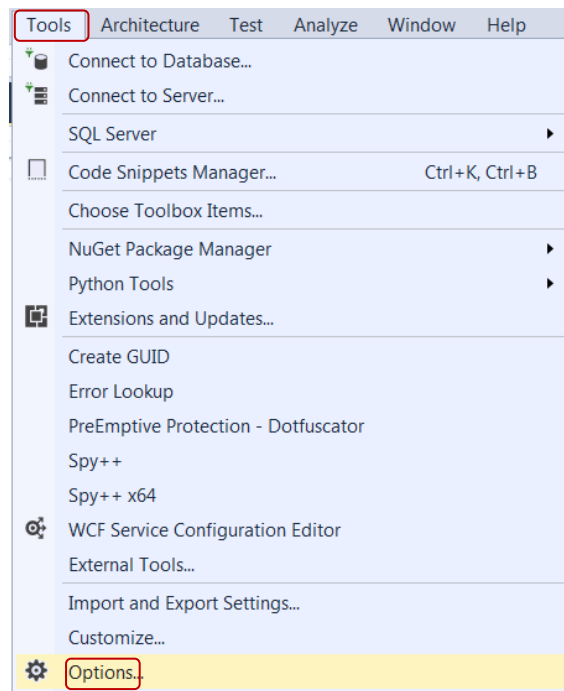
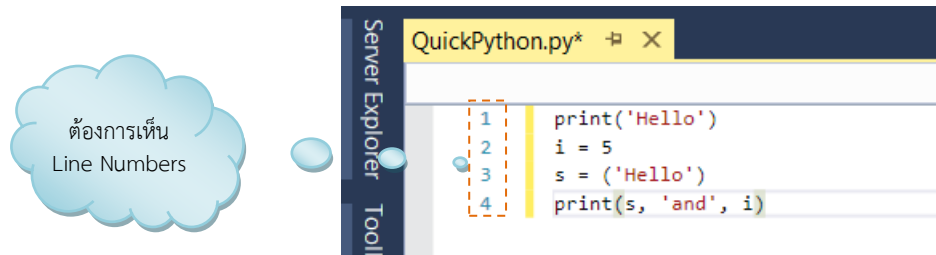
การเพิ่ม / ลด ปุ่ม ใน Menu Bar

เลือกที่
ต้องการ



ยังมีอีก
ใช้ mouse ดู

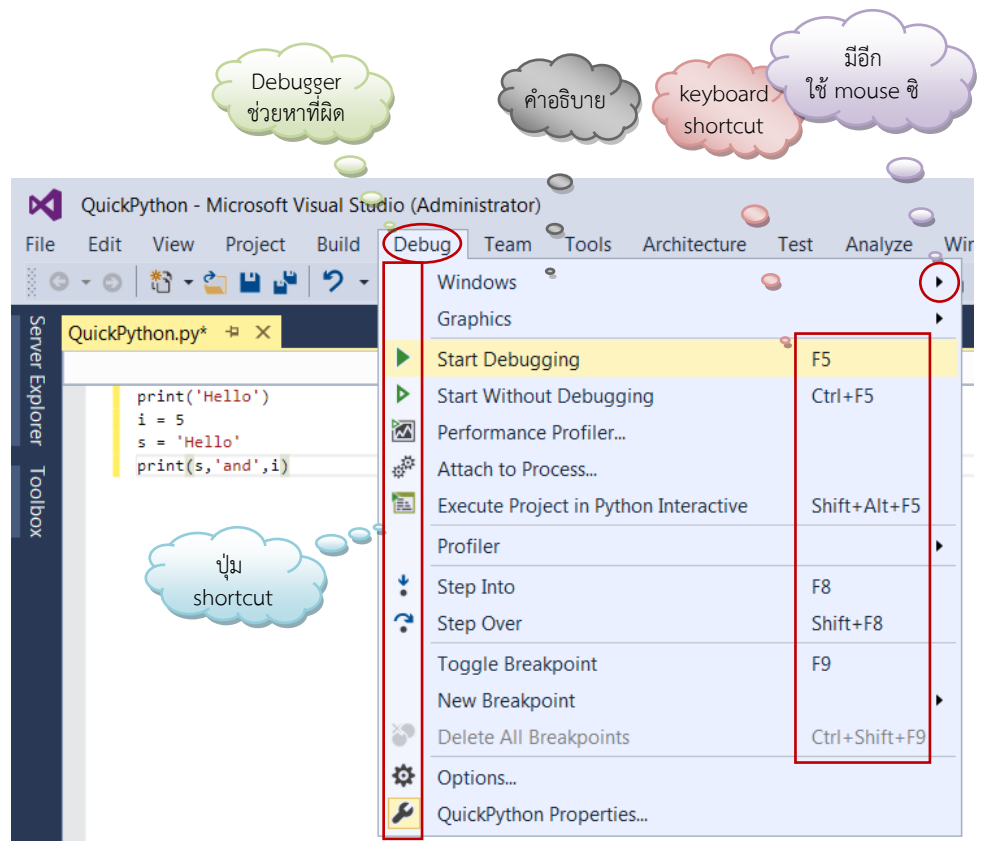
2.2. Line Numbers



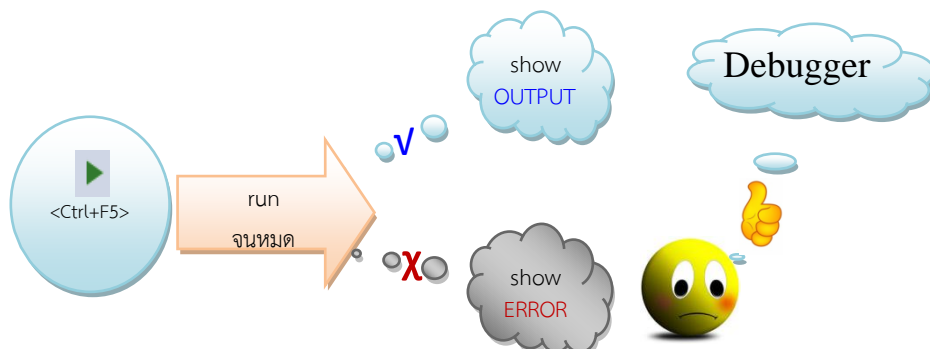
2.3. การ run

MS Visual Studio มี debugger ช่วยในการ debug หาที่ผิดพลาดเวลา run

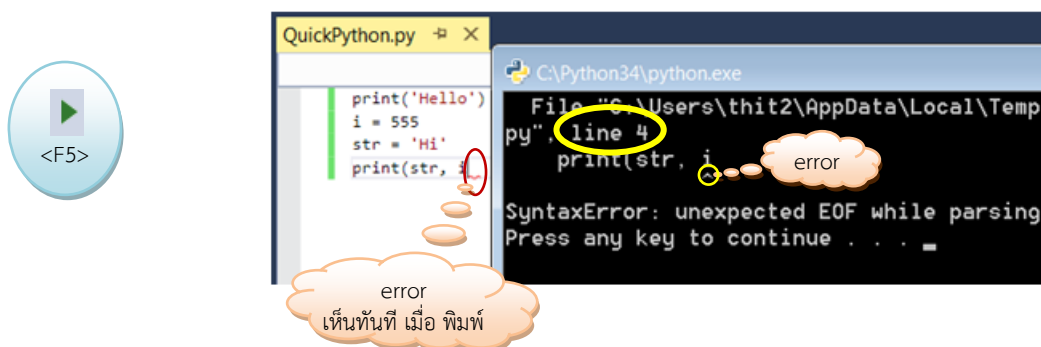
มีปุ่ม short cut ให้เลือก แต่หากจำไม่ได้ เข้าไปเลือกใน Debug Menu



2.3.1.  <Ctrl+F5> ไม่ใช้ debugger run ทั้งหมด แล้วค่อย แสดง output / ผิด แสดง error



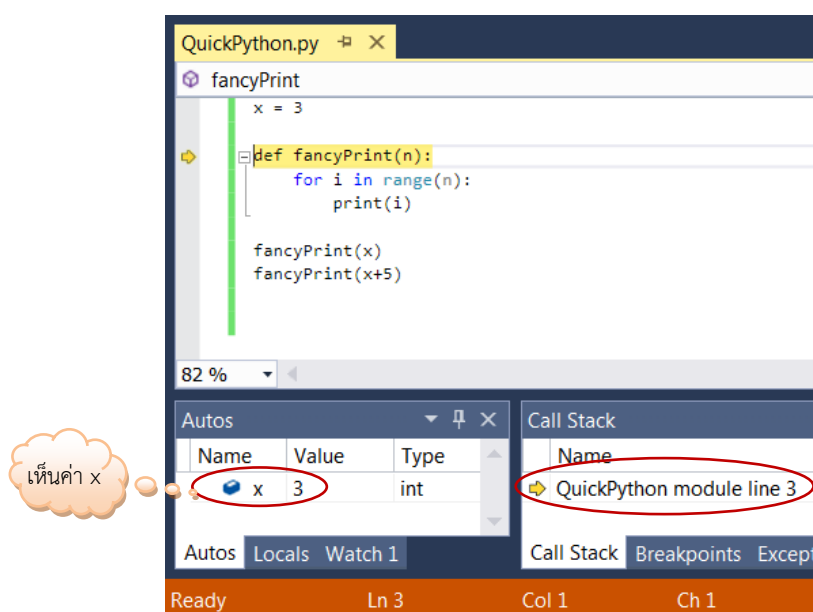
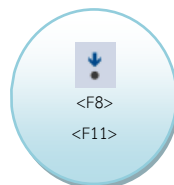
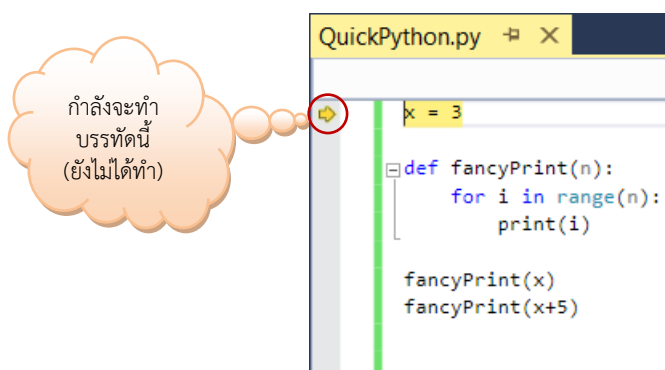
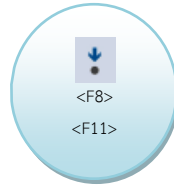
2.3.2.  <F5> ใช้ debugger แล้วค่อย แสดง output ทั้งหมด / ผิด แสดง error

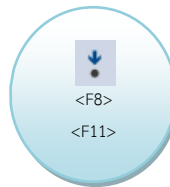


2.3.3.  <Shift+F8> หรือ <F10> Step Over ใช้ debugger และ ไม่เข้าไปในฟังก์ชัน ข้ามไป

2.3.4.  <F8> หรือ <F11> Step Into ใช้ debugger และ เข้าไป run ในฟังก์ชันให้เห็น

2.3.5.  <Ctrl+Shift+F8> Step Out ใช้ debugger และ หลุดออกจากฟังก์ชัน





อ่านข้าม fⁿ definition ไป (ไม่ได้ถูก call)

```
QuickPython.py
```

```
x = 3
```

```
def fancyPrint(n):
```

```
    for i in range(n):
```

```
        print(i)
```

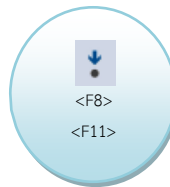
```
fancyPrint(x)
```

```
fancyPrint(x+5)
```

Name	Value	Type
x	3	int

Name
QuickPython module line 7

Ready Ln 7 Col 1 Ch 1



step into เข้าไปทำใน fⁿ

```
QuickPython.py
```

```
fancyPrint
```

```
x = 3
```

```
def fancyPrint(n):
```

```
    for i in range(n):
```

```
        print(i)
```

```
fancyPrint(x)
```

```
fancyPrint(x+5)
```

Name	Value	Type
n	3	int

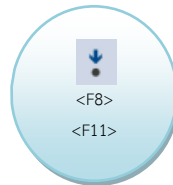
Name
fancyPrint in QuickPython line 4
QuickPython module line 7

Ready Ln 4 Col 1 Ch 1

เห็นค่า n

ดู stack จากล่างขึ้นบน

1. ทำ line 7 ใน QuickPython แล้ว
2. jump ไปทำ fancyPrint()



$i = 0 < 3$ เข้าใน for

QuickPython.py

```
fancyPrint
x = 3
def fancyPrint(n):
    for i in range(n):
        print(i)
fancyPrint(x)
fancyPrint(x+5)
```

82 %

Name	Value	Type
i	0	int
n	3	int

Autos Locals Watch 1

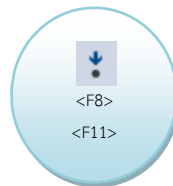
Ready Ln 5 Col 1 Ch 1

Call Stack

Name
fancyPrint in QuickPython line 5
QuickPython module line 7

Call Stack Breakpoints Exception S

เห็นค่า n, i



print output i

QuickPython.py

```
fancyPrint
x = 3
def fancyPrint(n):
    for i in range(n):
        print(i)
fancyPrint(x)
fancyPrint(x+5)
```

82 %

Name	Value	Type
i	0	int
n	3	int

Autos Locals Watch 1

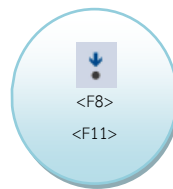
Ready Ln 4 Col 1 Ch 1

Call Stack

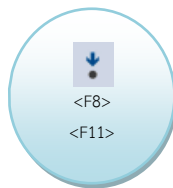
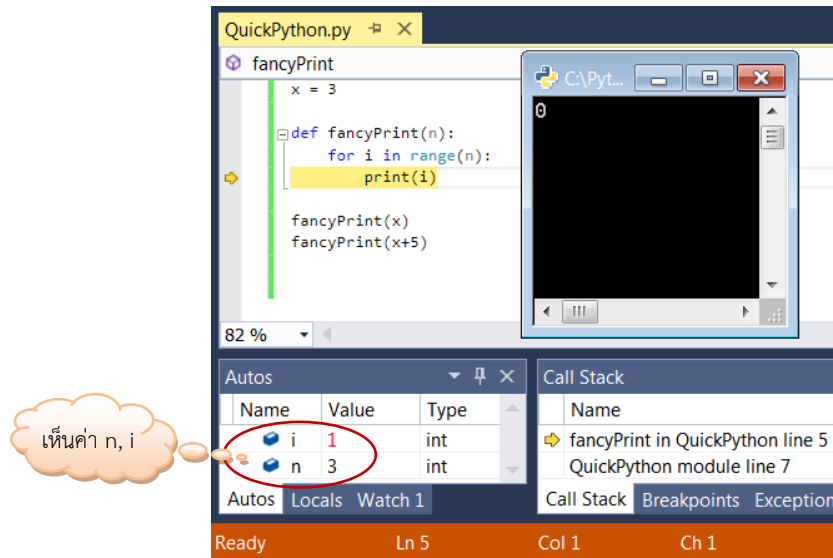
Name
fancyPrint in QuickPython line 4
QuickPython module line 7

Call Stack Breakpoints Exception

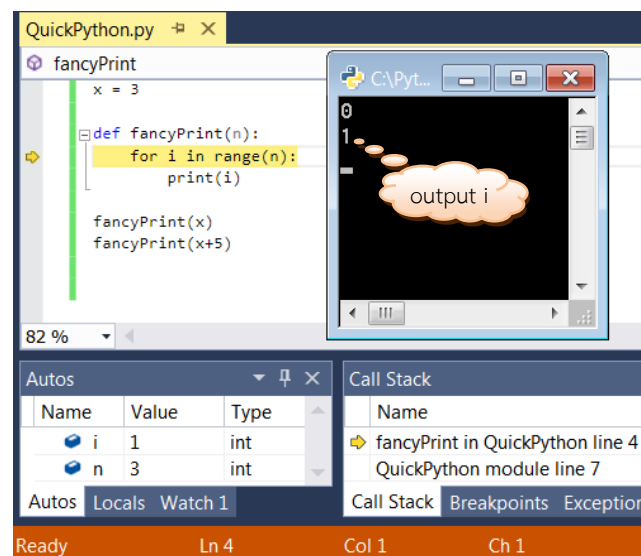
output i

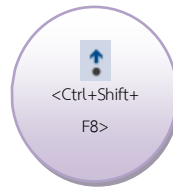


$i = 1 < 3 (n)$ เข้าใน for

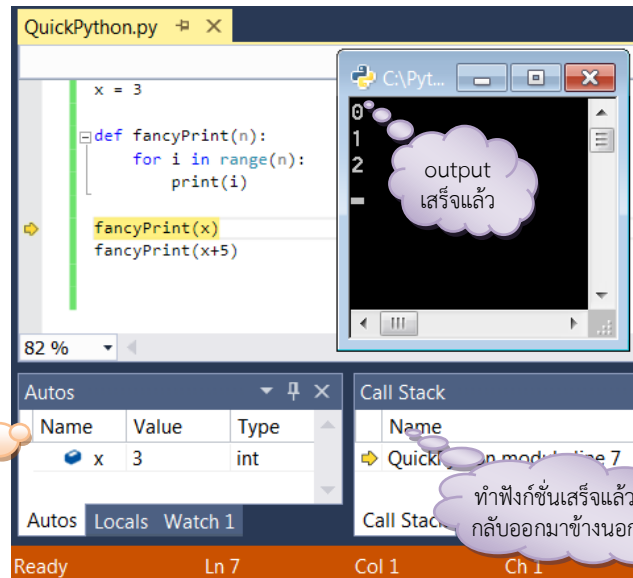


print output i





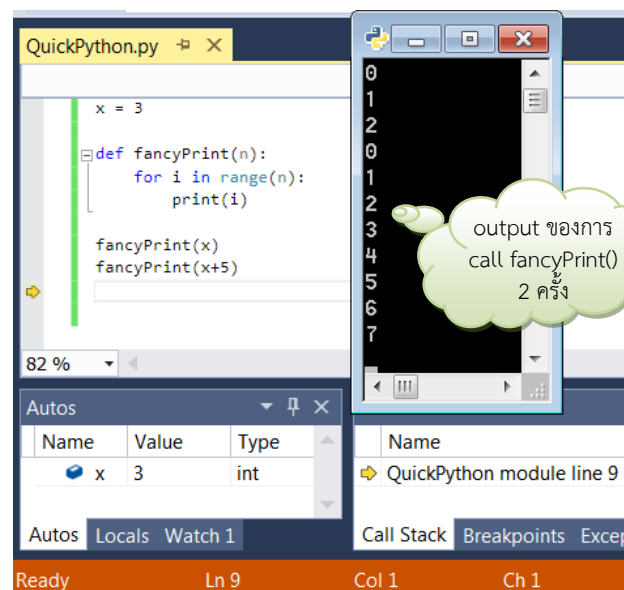
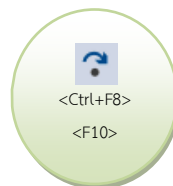
step out :
control หลุดจากฟังก์ชัน (ไม่แสดงการรันต่อ)
ได้ output เรียบร้อยแล้ว



ค่า local variables : n, i
ของ fancyPrint() หายไป
เพราะอยู่นอก scope

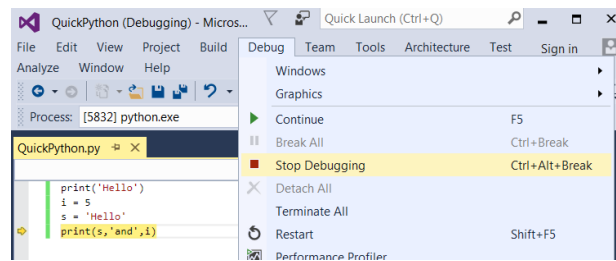
เห็นค่า x ซึ่งเป็น local
ของ QuickPython แทน

step over :
run fancyPrint(x+5) โดยไม่แสดง
ได้ output เรียบร้อยแล้ว



output ของการ
call fancyPrint()
2 ครั้ง

2.4. Stop Debugging ใช้หยุดการ debug



3. การทดลองที่ 3 : Control Flow

3.1. If มี 3 แบบ

1. if

```
if test expression:
    statement(s)
```

: แสดงว่า
ต่อไปเป็น body block ของบรรทัดนี้
ใช้การเยื้องเท่ากัน เป็นตัวกำหนด

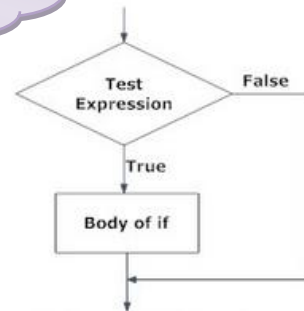


Fig: Operation of if statement

2. if ... else

```
if test expression:
    Body of if
else:
    Body of else
```

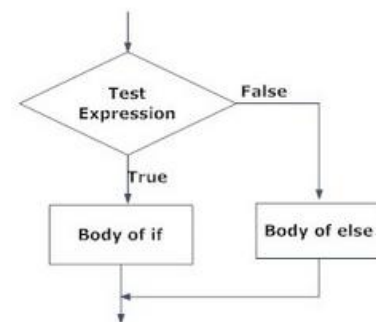


Fig: Operation of if...else statement

3. if ... elif ...else # if ... else if ... else

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

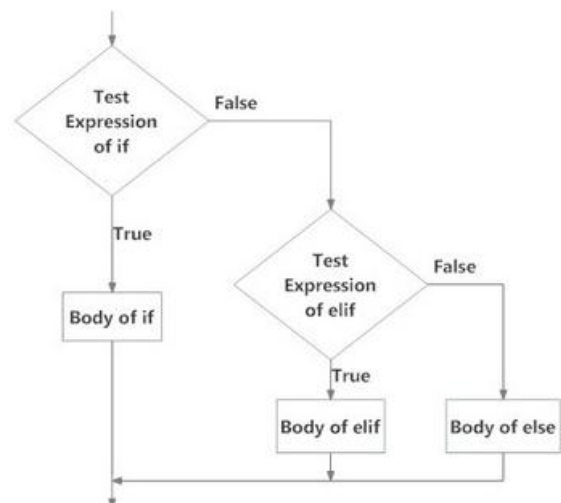
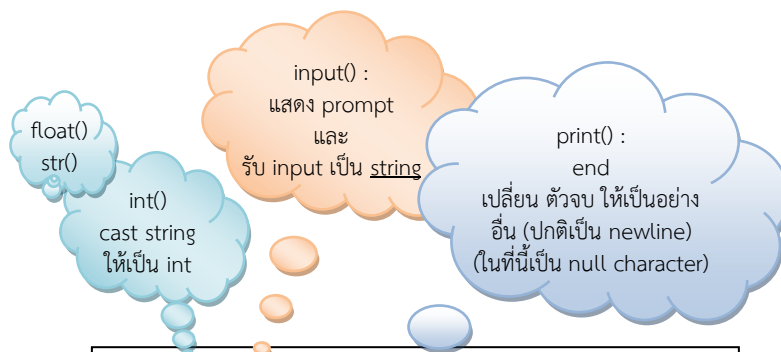
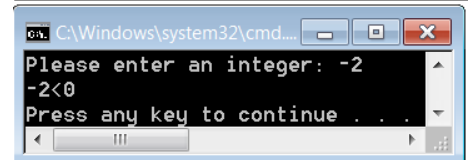
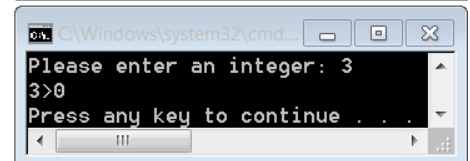
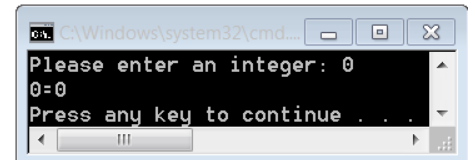


Fig: Operation of if...elif...else statement

- level ของ nesting ใช้การเยื้อง indentation
- ไม่เหมือน C condition ไม่จำเป็นต้องอยู่ใน ()
- : จำเป็นต้องมี แสดงว่าข้างล่างมี block ของ statements ในบรรทัดนี้



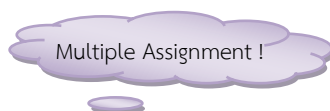
```
x = int(input("Please enter an integer: "))
print(x, end = '\n')
if x >= 0:
    if x == 0:
        print('=0')
    else:
        print('>0')
else:
    print('<0')
```



```
score = int(input('input your score : '))
if score < 50:
    print('fail')
elif score == 50:
    print('Ohh!! almost fail')
else: print('pass')
```

แบบฝึกหัด : เขียน Python Code ต่อเพื่อพิมพ์ค่าสูงสุดของค่า 3 ค่า และรันพิสูจน์โดยเปลี่ยนค่า test case

เฉลย :



```
x, y, z = 5,7,3
if x > y and x > z :
    print('max = ',x)
elif y > x and y > z :
    print('max = ',y)
else: print('max = ',z)
```

```
x, y, z = 15,7,20
if x > y > z:
    print('max = ',x)
elif y > x > z :
    print('max = ',y)
else: print('max = ',z)
```

สะดวกมาก !

3.2. while

```
while test_expression:
    Body of while
```

```
# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

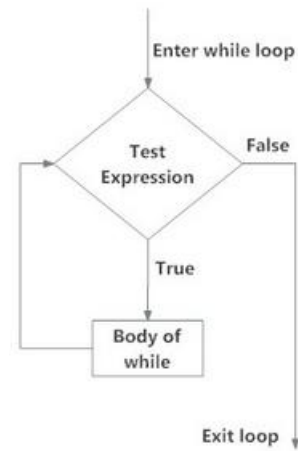
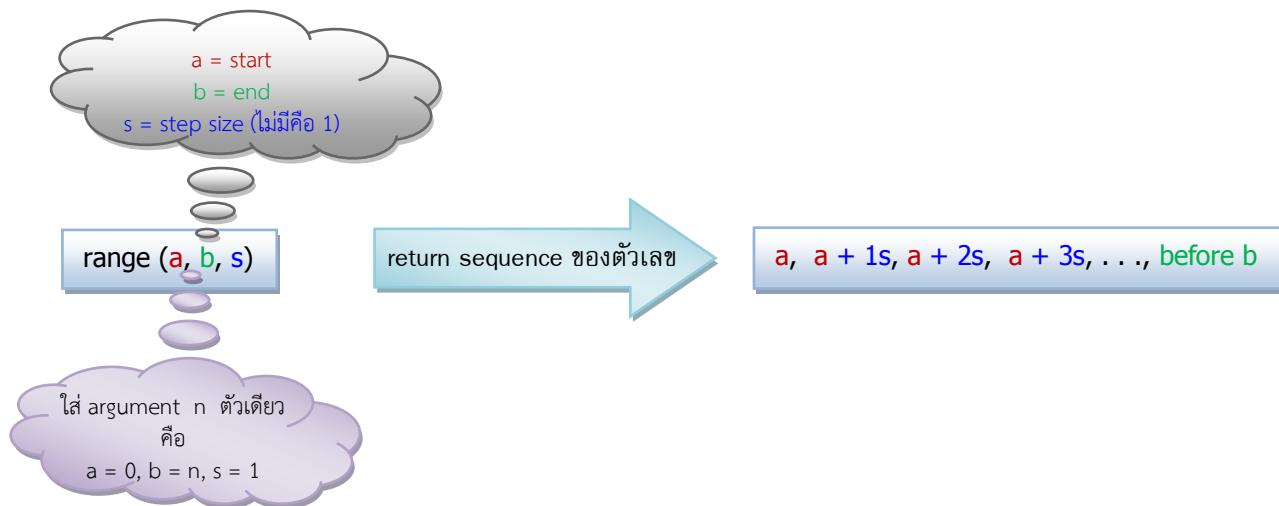


Fig: operation of while loop

3.3. range()



```
print(list(range(5)))
print(list(range(0,5)))
print(list(range(-10,-50,-20)))
```

```
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4]
[-10, -30]
```

3.4. for

variable ซึ่งเอาค่า
จาก sequence มา
ในแต่ละ iteration

for val in sequence:
Body of for

```
# Find sum of elements in list
list = [2, 1, 3, 4]
sum = 0

for ele in list:      # iterate over the list
    sum = sum + ele

print("Sum =", sum) → Sum = 10
```

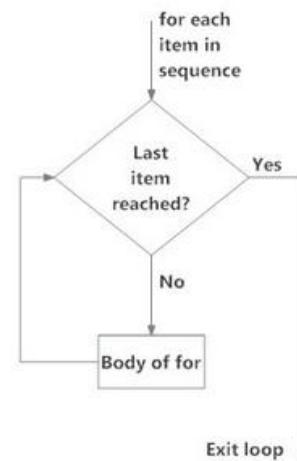


Fig: operation of for loop

```
for i in range(5):
    print(i, end = ' ')
```

0 1 2 3 4

?

```
list[0] = 2
list[1] = 1
list[2] = 3
list[3] = 4
```

3.5. break , continue, else clauses on loops

break

ออกจาก loop ใกล้เคียงที่ล๊อมนั้น
ไม่ทำ else ของ loop ด้วย

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, '=', x, '*', n//x)
            break
    else:
        # loop fell through without finding a factor
        print(n, 'is prime')
```

else

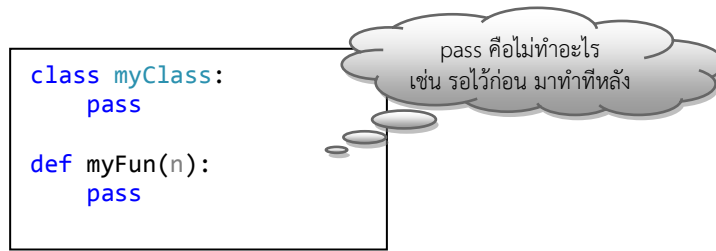
ทำเมื่อทำงานหมด for
หรือ
while condition เป็น false

continue

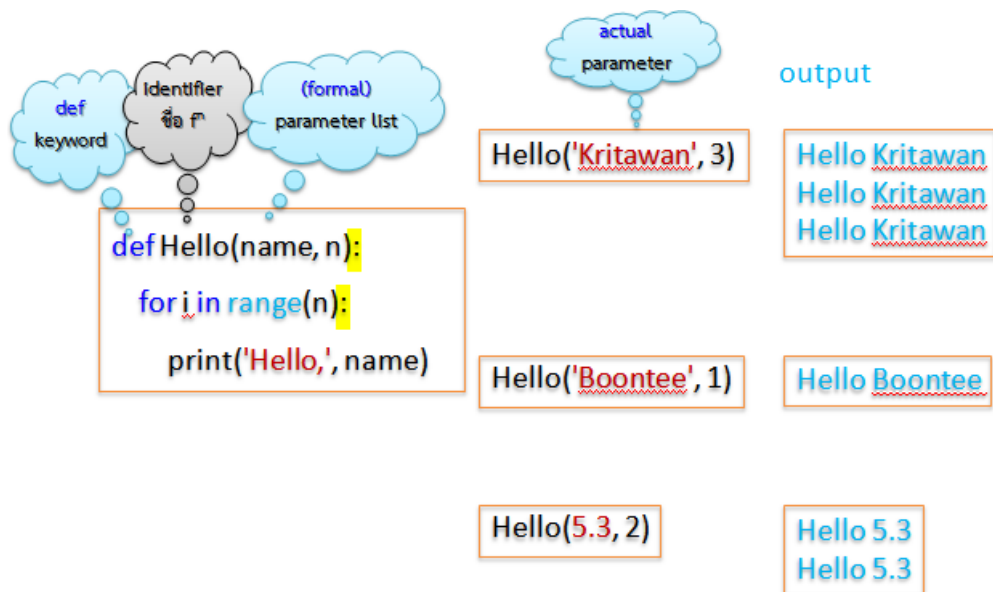
ออกจาก iteration นั้นทันที
ไปทำ iteration ถัดไป

```
2 is prime
3 is prime
4 = 2 * 2
5 is prime
6 = 2 * 3
7 is prime
8 = 2 * 4
9 = 3 * 3
```

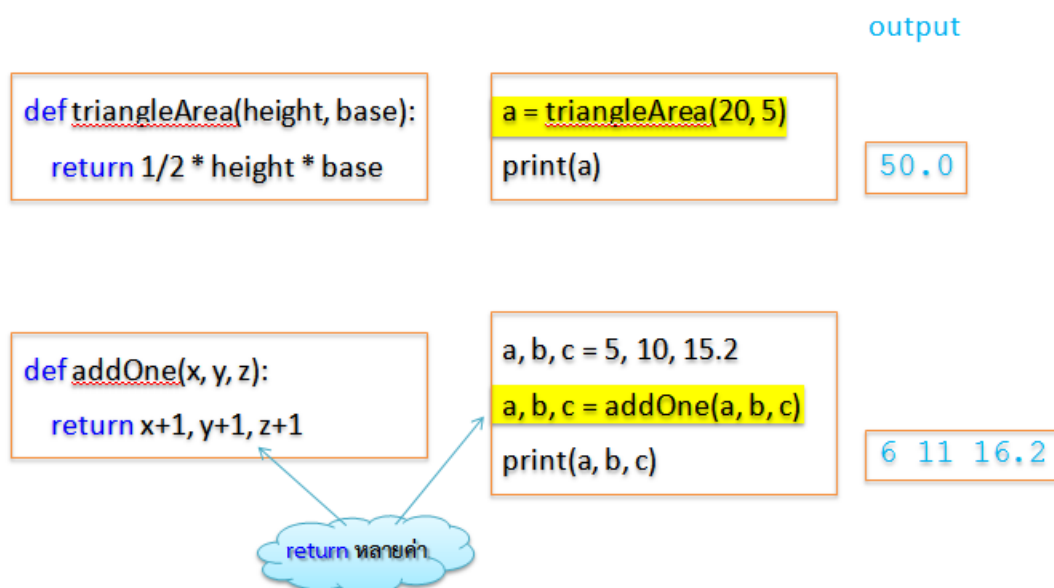
3.6. pass



3.7. Function Definition



3.8. Return Value(s) จาก Function



4. การทดลองที่ 4 : แบบฝึกหัด

เขียนฟังก์ชันต่อไปนี้ โดยลบ comment และ raise statement ออก

4.1. factorial

```
def factorial(n):  
    # YOUR CODE HERE  
    raise NotImplementedError()
```

4.2. หาผลบวกของจำนวนนับที่น้อยกว่า n ซึ่งเป็น multiples ของ 3 หรือ 5 เช่น multiples ของ 3 หรือ 5 ที่น้อยกว่า 10 คือ 3 5 6 และ 9 ซึ่งมีผลบวกคือ 23

```
def multiples_of_3_and_5(n):  
    # YOUR CODE HERE  
    raise NotImplementedError()
```

4.3. หากให้เส้นรอบรูปคือ 60 เราสามารถหาสามเหลี่ยมมุมฉาก (right triangle) ได้ 2 รูป ซึ่งมีด้านดังนี้ [(10, 24, 26), (15, 20, 25)] เติม code ข้างล่าง กำหนด integer p และ returns list ของ tuples (a, b, c) ซึ่งเป็นด้านทั้งสามของสามเหลี่ยมมุมฉาก แต่ละ tuple ควร $a \leq b < c$ list ไม่ควรมี solution ที่ซ้ำกัน และ ควรเรียงลำดับจากน้อยไปมากของด้าน a

```
def integer_right_triangles(p):  
    # YOUR CODE HERE  
    raise NotImplementedError()
```

4.4. เขียนฟังก์ชัน gen_pattern() เพื่อทำ pattern ดังแสดงตัวอย่าง เมื่อเรียกฟังก์ชันนี้ด้วย string ที่มีความยาว ≥ 1 จะสร้าง ASCII art pattern ของชั้นของรูปข้าวหลามตัดโดยใช้ character เหล่านั้น return กลับจากฟังก์ชัน

```
def gen_pattern(chars):  
    # YOUR CODE HERE  
    raise NotImplementedError()
```

```

> print(gen_pattern('X'))

X

> print(gen_pattern('XY'))

..Y..
Y.X.Y
..Y..

> print(gen_pattern('WXYZ'))

.....Z.....
....Z.Y.Z....
..Z.Y.X.Y.Z..
Z.Y.X.W.X.Y.Z
..Z.Y.X.Y.Z..
....Z.Y.Z....
.....Z.....

```

ฟังก์ชันจะ return pattern เป็น string (ไม่ใช่ print ออกมา) ดังนั้นแต่ละบรรทัดต้องคั่นด้วย newline ซึ่งใน Python ใช้ '\n' ตัวอย่าง pattern ที่สองจะ return str '..Y..\nY.X.\n..Y..'

โจทย์ข้างต้นจะต้องใช้ ฟังก์ชัน 2 ฟังก์ชัน ของ type str คือ join และ center ดังแสดงในตัวอย่าง :

```

> '*' .join(['one', 'two', 'three'])

'one*two*three'

> '*' .join('abcde')

'a*b*c*d*e'

> 'hello'.center(11, '*')

'***hello***'

```