

01076105, 01076106

Object Oriented Programming

Object Oriented Programming Project

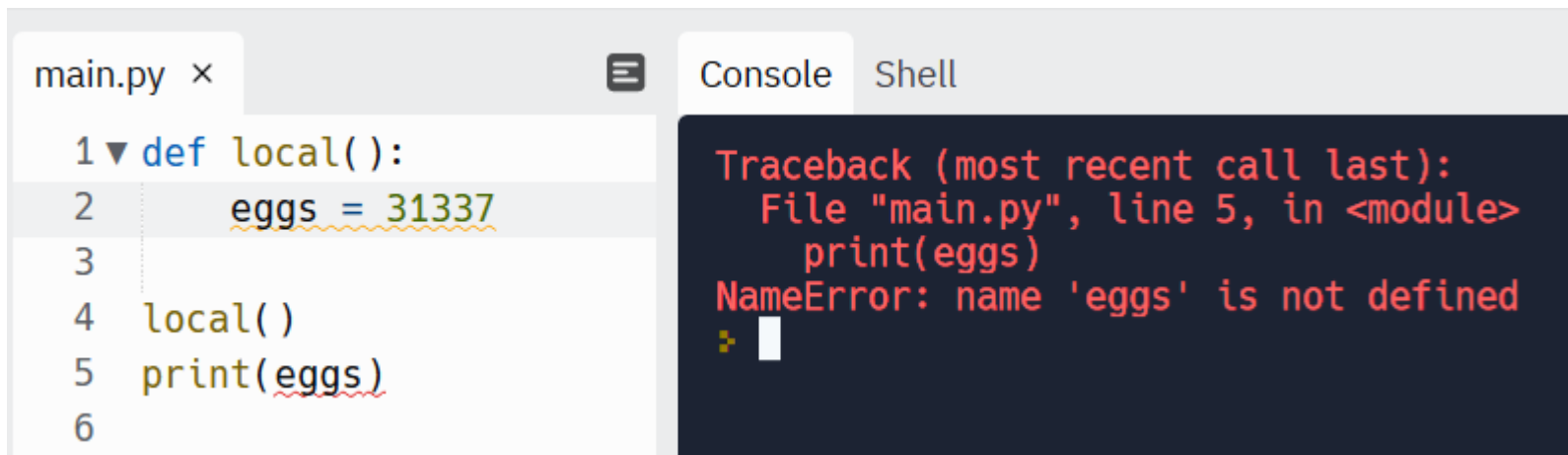
From C to Python #3

Variable Scope

- ในการใช้ฟังก์ชัน จะมีเรื่องหนึ่งที่ต้องพิจารณา คือ ขอบเขตที่ใช้งานได้ของแต่ละตัวแปร
- ตัวแปร หรือ argument ของฟังก์ชันจะเรียกว่า local scope ส่วนตัวแปรที่กำหนดไว้ นอกฟังก์ชัน จะเรียกว่า global scope
- ให้มองว่า scope ก็เหมือนกับกล่อง ตัวแปรที่อยู่ในกล่อง จะเกิดขึ้นมาได้ก็ต้องมีกล่อง เสียก่อน จากนั้นจึงสร้างตัวแปร ดังนั้นตัวแปรที่อยู่ในฟังก์ชัน ก็จะสร้างขึ้นตอนที่ ฟังก์ชันถูกเรียก และเมื่อ return ก็เหมือนกับกล่องถูกทำลาย ตัวแปรในกล่องก็就会被ทำลายไปด้วย
- คำว่า local แปลว่าใกล้ๆ ดังนั้นโปรแกรมที่อยู่ภายนอกฟังก์ชัน จะไม่สามารถเข้าถึง ข้อมูลที่อยู่ในฟังก์ชัน แต่โปรแกรมในฟังก์ชัน สามารถอ้างถึงตัวแปรที่อยู่นอกตัวได้

Variable Scope

- โปรแกรมที่อยู่ภายนอกฟังก์ชัน จะไม่สามารถเข้าถึงข้อมูลที่อยู่ในฟังก์ชัน ตามตัวอย่าง



The screenshot shows a Python IDE with a file named `main.py`. The code in the editor is as follows:

```
1 def local():  
2     eggs = 31337  
3  
4 local()  
5 print(eggs)  
6
```

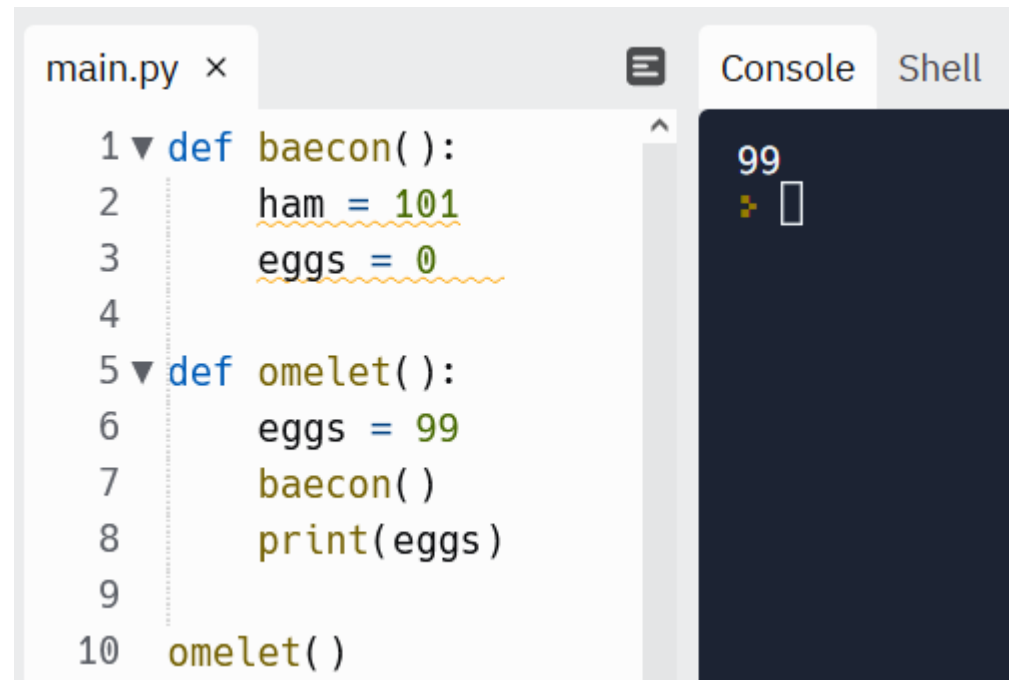
The `Console` tab is active, displaying a `Traceback` error:

```
Traceback (most recent call last):  
  File "main.py", line 5, in <module>  
    print(eggs)  
NameError: name 'eggs' is not defined
```

The error occurs because the variable `eggs` is defined inside the `local()` function but is accessed from the global scope in line 5.

Variable Scope

- โปรแกรมที่อยู่คนละ variable scope ถือเป็นคนละตัวแปรกัน แม้จะมีชื่อเดียวกัน
- ตัวแปร eggs กำหนดใน omelet ต่อมา omelet เรียก beacon ทำให้มีการสร้าง eggs ขึ้นมาอีกตัวหนึ่ง กำหนดค่า = 0
- แต่เมื่อกลับมา จะพบว่าตัวแปร eggs มีค่าเท่าเดิม
- เพราะตัวแปร eggs ทั้ง 2 ตัว ถือเป็นคนละตัวกัน เพราะ local คือ เฉพาะขอบเขตนั้น



```
main.py x
1 ▼ def baecon():
2     ham = 101
3     eggs = 0
4
5 ▼ def omelet():
6     eggs = 99
7     baecon()
8     print(eggs)
9
10 omelet()
```

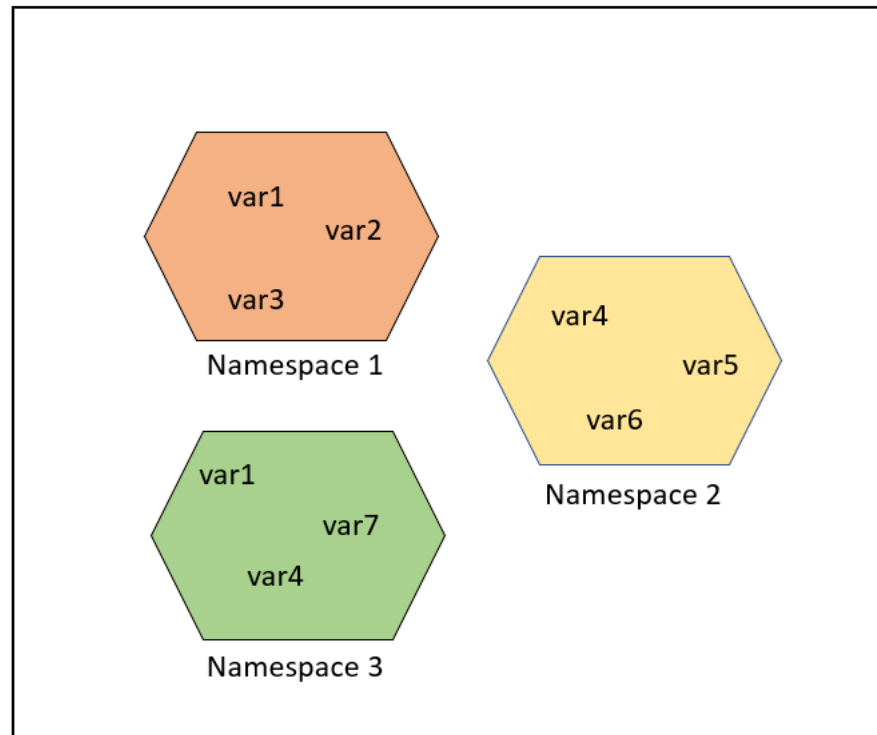
Console

99

<https://autbor.com/otherlocalscopes/>.

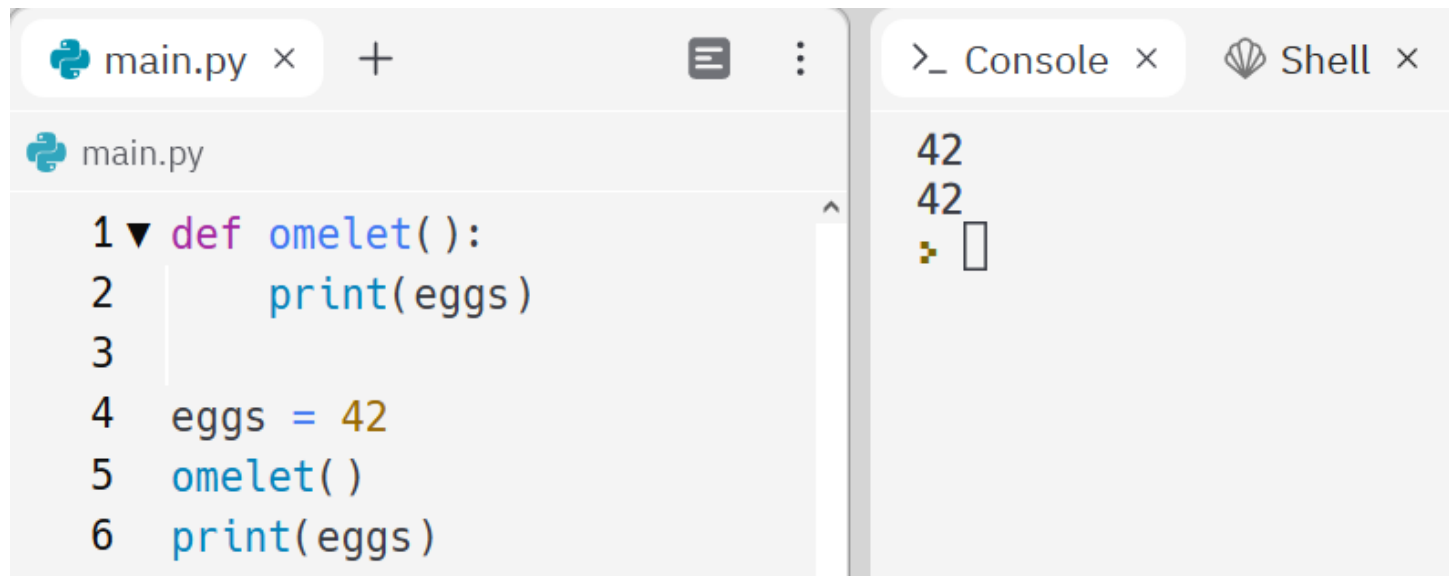
Variable Scope

- ภาพนี้แสดง โครงสร้างของตัวแปรแบบ Local Scope จะเห็นว่าแต่ละ function จะมี namespace ของตัวเอง ดังนั้นในแต่ละฟังก์ชันจึงไม่รู้จักตัวแปรที่อยู่ในฟังก์ชันอื่น ไม่สามารถอ้างอิง และ อาจตั้งชื่อซ้ำกันก็ได้



Variable Scope

- ตัวแปรที่กำหนดไว้ใน global scope สามารถอ้างถึงโดย local ได้
- จะเห็นว่าตัวแปร eggs มีการกำหนดไว้ที่ global scope
- แต่ในฟังก์ชัน omelet ซึ่งไม่มีการกำหนดตัวแปรนี้ไว้ ก็สามารถอ้างถึงได้เช่นกัน



The screenshot shows a Python IDE with a file named `main.py` and a console window. The code in `main.py` is as follows:

```
1 def omelet():  
2     print(eggs)  
3  
4 eggs = 42  
5 omelet()  
6 print(eggs)
```

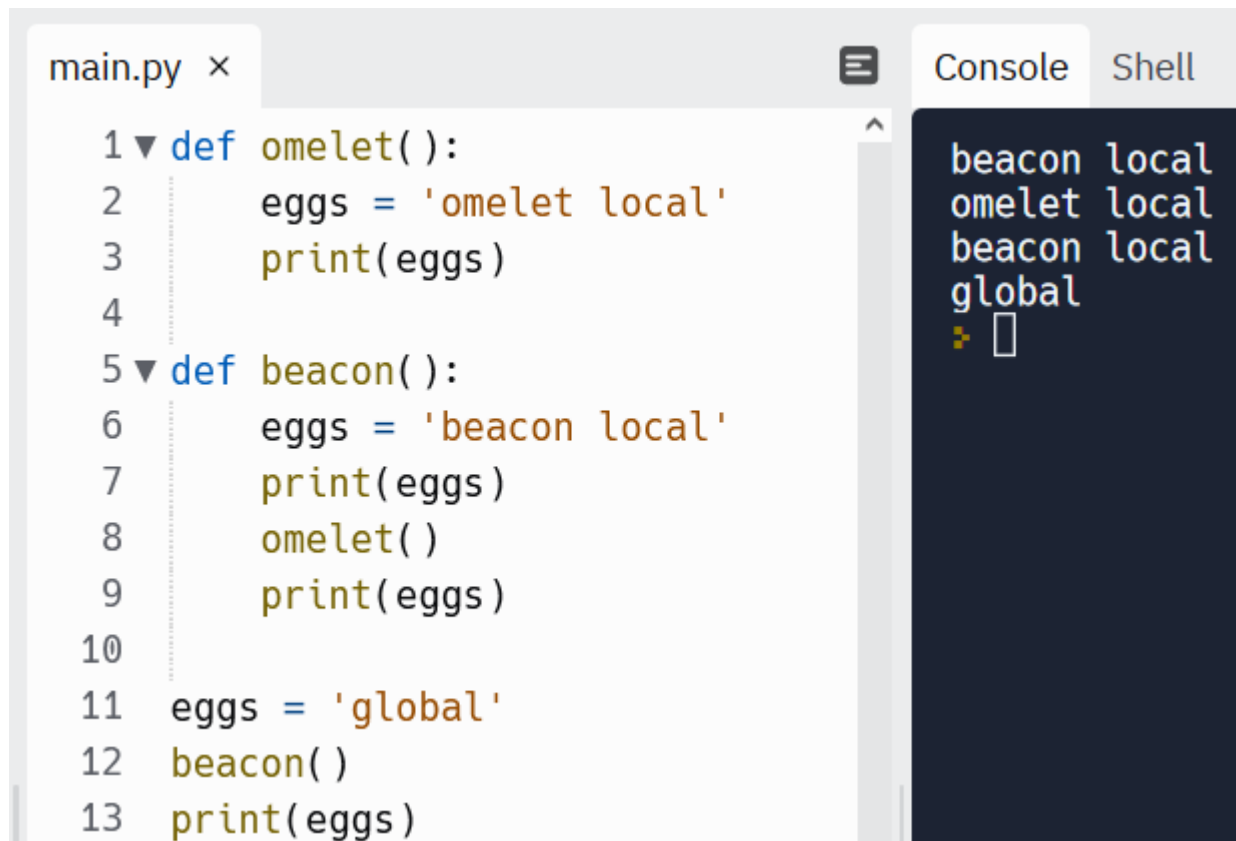
The console window shows the output of the program:

```
>_ Console x Shell x  
42  
42  
>_
```

The output consists of two lines of `42`, which corresponds to the `print(eggs)` statement inside the `omelet` function and the `print(eggs)` statement at the end of the script.

Variable Scope

- กรณีที่ local scope และ global scope มีชื่อเดียวกัน ถือเป็นตัวแปรคนละตัวกัน



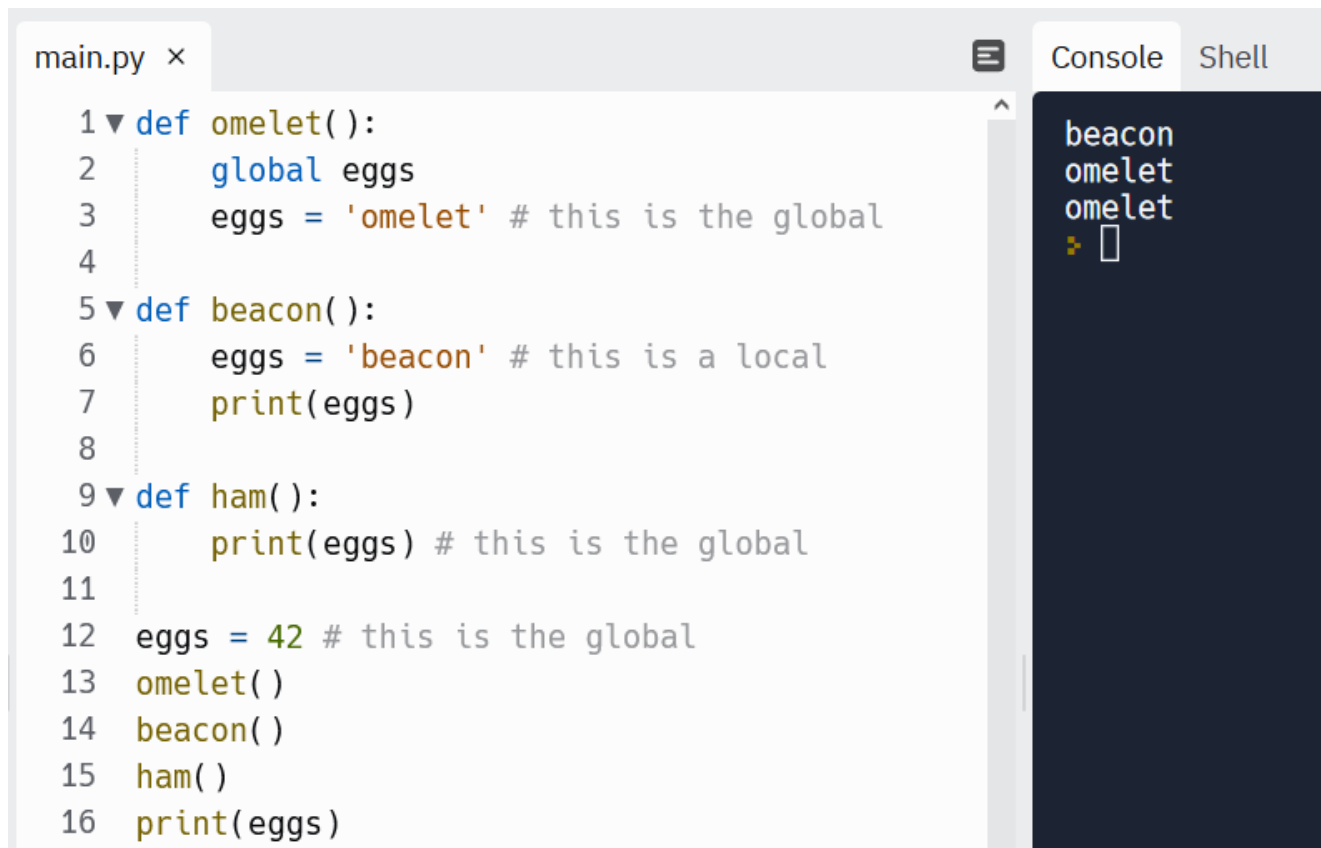
```
main.py x
1 ▼ def omelet():
2     eggs = 'omelet local'
3     print(eggs)
4
5 ▼ def beacon():
6     eggs = 'beacon local'
7     print(eggs)
8     omelet()
9     print(eggs)
10
11 eggs = 'global'
12 beacon()
13 print(eggs)
```

Console

```
beacon local
omelet local
beacon local
global
```

Variable Scope

- แต่เราสามารถประกาศตัวแปรจาก local scope ให้เป็นตัวแปรในระดับ global ได้โดยใช้คำสั่ง `global` จากรูปจะถือว่า `eggs` ใน `omelet` ตัวเดียวกับใน `global`



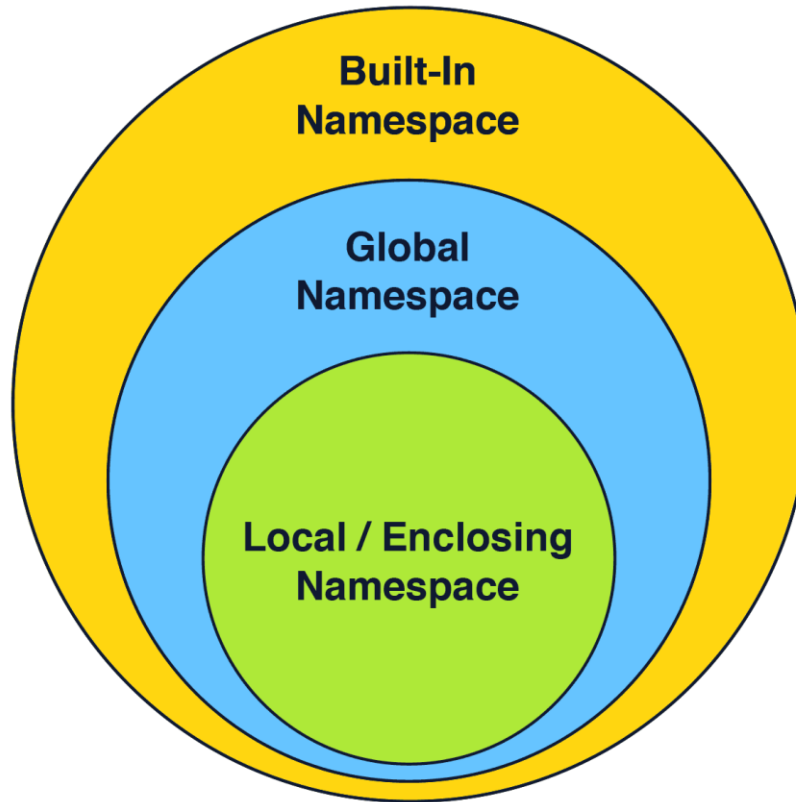
```
main.py x
1 ▼ def omelet():
2     global eggs
3     eggs = 'omelet' # this is the global
4
5 ▼ def beacon():
6     eggs = 'beacon' # this is a local
7     print(eggs)
8
9 ▼ def ham():
10    print(eggs) # this is the global
11
12    eggs = 42 # this is the global
13    omelet()
14    beacon()
15    ham()
16    print(eggs)
```

Console

```
beacon
omelet
omelet
>
```

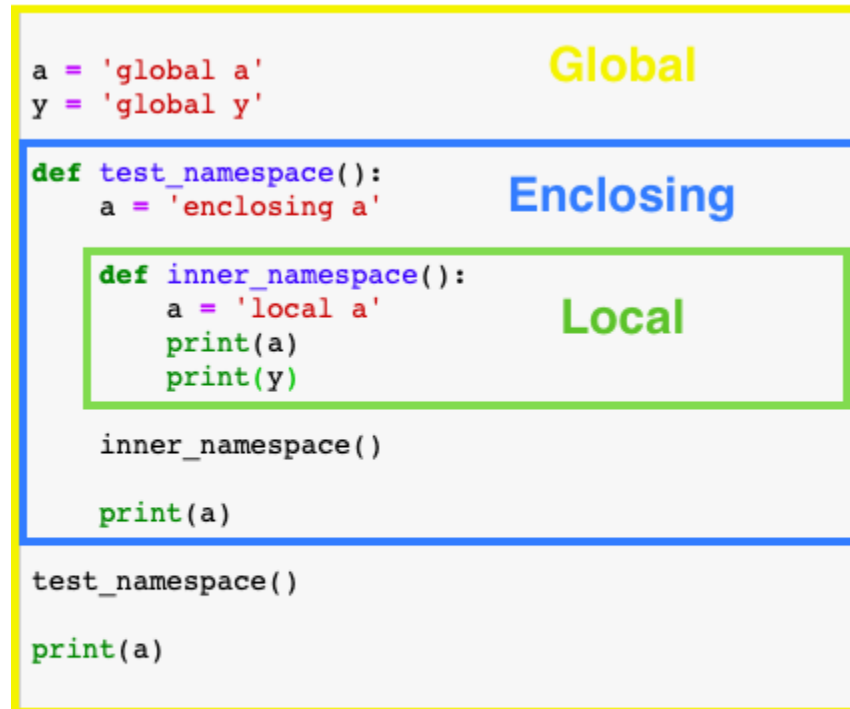

Variable Scope

- รูปแสดงขอบเขต Namespace ของ Python
- ระดับ Enclosing คือเป็นฟังก์ชันระดับที่อยู่เหนือขึ้นไป



Variable Scope

- แสดงขอบเขต Namespace ของ Python



```
local a
global y
enclosing a
global a
```

Variable Scope

- สรุปหลักการในการค้นหาตัวแปรใน Namespace ต่างๆ
- เขียนเป็นตัวย่อ คือ LEGB
 - Local: ถ้าตัวแปร x อยู่ภายใน function โปรแกรม python จะใช้ตัวแปรใน function (local)
 - Enclosing: ถ้าตัวแปร x ไม่อยู่ใน local scope แต่พบใน function ที่อยู่ใน function ด้านนอก, โปรแกรม python จะใช้ในตัวแปรใน enclosing function's scope.
 - Global: ถ้าตัวแปร x ไม่อยู่ทั้งใน local scope และ enclosing function's scope โปรแกรม python จะค้นหาใน global เป็นลำดับต่อไป
 - Built-in: ถ้าไม่พบตัวแปร x ในที่ใดๆ โปรแกรม python จะพยายามหาใน built-in scope

Variable Scope

- **Exercise** จงหาคำตอบของโปรแกรมต่อไปนี้

main.py ×

```
1  #| Example 1: Single Definition
2
3  x = 'global'
4  def f():
5      def g():
6          print(x)
7          g()
8
9  f()
```

global



Variable Scope

- Exercise จงหาคำตอบของโปรแกรมต่อไปนี้

main.py ×

```
1  #| Example 2: Double Definition
2
3  x = 'global'
4 ▼ def f():
5      x = 'enclosing'
6
7 ▼     def g():
8         print(x)
9
10        g()
11
12    f()
```

enclosing



Variable Scope

- Exercise จงหาคำตอบของโปรแกรมต่อไปนี้

```
main.py ×  
1  #| Example 3: Triple Definition  
2  
3  x = 'global'  
4 ▼ def f():  
5      x = 'enclosing'  
6 ▼      def g():  
7          x = 'local'  
8          print(x)  
9      g()  
10  
11 f()  
12
```

local



Variable Scope

- Exercise จงหาคำตอบของโปรแกรมต่อไปนี้

main.py ×

```
1  #| Example 5: Local และ Enclosing Name Space
2
3  ▼ def f():
4      print('Start f()')
5  ▼  def g():
6      print('Start g()')
7      print('End g()')
8      return
9      g()
10     print('End f()')
11     return
12
13  f()
```

```
Start f()
Start g()
End g()
End f()
> █
```

Variable Scope

- ใน local scope แม้ว่าจะอ้างถึง global scope ได้ แต่ไม่สามารถ assignment ได้
- โปรแกรมนี้หาก run จะเกิด Error ตามภาพ เพราะโปรแกรมเข้าใจว่า var เป็น local

main.py ×

```
1  #| Example : หากแก้ไขตัวแปรแบบ Global ใน function จะเกิดอะไรขึ้น
2  var = 100 # A global variable
3  ▼ def increment():
4      var = var + 1 # Try to update a global variable
5
6  increment()
```

UnboundLocalError: local variable 'var' referenced
before assignment

Variable Scope

- หรือการอ้างถึง ก่อนจะ assignment ก็เช่นกัน โปรแกรมจะมองว่าเป็น local scope เนื่องจากมีการ assignment ภายใน function

main.py ×

```
1  #| Example : กรณีกำหนดตัวแปร Local แล้วอ้างถึง
2  var = 100 # A global variable
3  ▼ def func():
4      print(var) # Reference the global variable, var
5      var = 200 # Define a new local variable using the same name, var
6
7  func()
8
```

Variable Scope

- สมมติว่าต้องการเขียน function `update_counter` ซึ่งทำหน้าที่เพิ่มค่า `counter` จะทำอย่างไร เช่น โปรแกรมตามรูปจะ error

main.py ×

```
1  #| Example 10: การอ้างถึงตัวแปร Global ภายใน function
2  counter = 0 # A global name
3  ▼ def update_counter():
4      counter = counter + 1 # Fail trying to update counter
5
6  update_counter()
7
```

Variable Scope

- ให้แก้ไขโปรแกรมตามนี้ โดยกำหนดให้เป็น global scope แล้วจึงสามารถ assignment ได้

main.py ×

```
1 counter = 0 # A global name
2
3 ▼ def update_counter():
4     global counter # Declare counter as global
5     counter = counter + 1 # Successfully update the counter
6
7     update_counter()
8     print (counter)
9     update_counter()
10    print (counter)
11    update_counter()
12    print (counter)
```

Variable Scope

- หรือจะใช้วิธีนี้ก็ได้อีก คือ ใช้วิธี return แล้วให้ assignment ไปอยู่ที่ global scope

main.py ×

```
1 global_counter = 0 # A global name
2
3 ▼ def update_counter(counter):
4     return counter + 1 # Rely on a local name
5
6 global_counter = update_counter(global_counter)
7 print (global_counter)
8 global_counter = update_counter(global_counter)
9 print (global_counter)
10 global_counter = update_counter(global_counter)
11 print (global_counter)
```

Variable Scope

- ตัวแปรที่กำหนด global ในฟังก์ชัน จะถือว่าเป็น global scope จึงสามารถอ้างใน global ได้

main.py ×

```
1  #| Example 12: การสร้างตัวแปร Global จากภายใน function
2  ▼ def create_lazy_name():
3      global lazy # Create a global name, lazy
4      lazy = 100
5      return lazy
6
7  create_lazy_name()
8  print(lazy) # The name is now available in the global scope
9
```

Variable Scope

- ในการอ้างถึงตัวแปรที่อยู่ใน enclosing scope จะใช้คำว่า nonlocal โดยจะหมายถึงตัวแปรที่อยู่ใน scope ถัดขึ้นไป

main.py ×

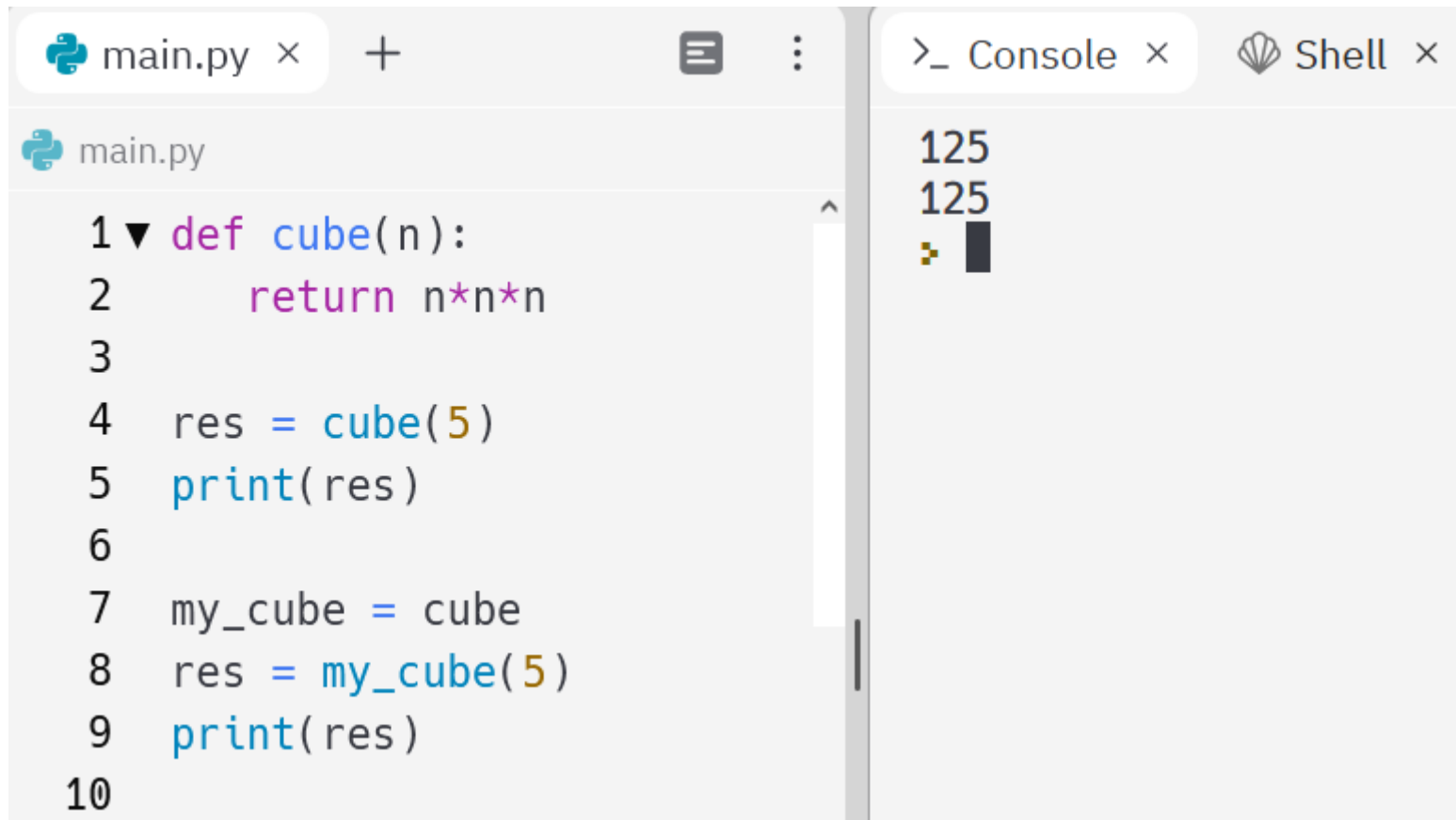
```
1 ▼ def func():
2     var = 100 # A nonlocal variable
3 ▼     def nested():
4         nonlocal var # Declare var as nonlocal
5         var += 100
6     nested()
7     print(var)
8 func()
```

First Class Function

- ฟังก์ชันในภาษา Python เอง ก็เป็น Object ดังนั้นจึงมีความสามารถเช่นเดียวกับ Object ชนิดอื่นๆ เช่น List โดยมีคุณสมบัติดังนี้
 - สามารถ Assign ให้ตัวแปร หรือ เก็บในตัวแปรได้
 - สามารถใช้เป็น Argument ในการผ่านค่าระหว่าง Function ได้
 - สามารถ Return จาก Function ได้
 - สามารถเก็บค่าใน List, Tuple หรือโครงสร้างข้อมูลอื่นๆ

First Class Function

- สามารถ Assign ให้ตัวแปร หรือ เก็บในตัวแปรได้



```
main.py x + [Icons] >_ Console x Shell x

main.py

1 ▼ def cube(n):
2     return n*n*n
3
4 res = cube(5)
5 print(res)
6
7 my_cube = cube
8 res = my_cube(5)
9 print(res)
10
```

The screenshot shows a Python IDE with a file named 'main.py' and a 'Console' window. The code in 'main.py' defines a function 'cube(n)' that returns 'n*n*n'. It then calls 'cube(5)' and assigns the result to 'res', prints 'res', and also assigns the function 'cube' to 'my_cube', calls 'my_cube(5)', and prints the result. The console shows the output '125' twice, corresponding to the two print statements.

First Class Function

— สามารถใช้เป็น Argument ในการผ่านค่าระหว่าง Function ได้

main.py

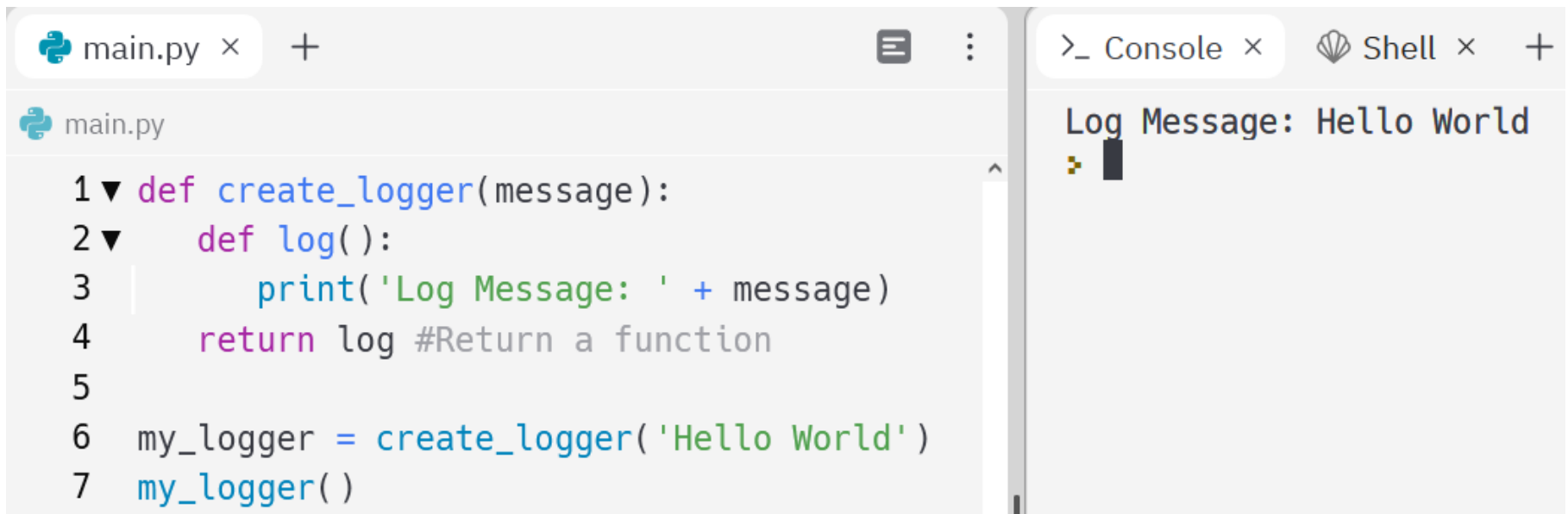
```
1 ▼ def cube(n):  
2     return n*n*n  
3  
4 ▼ def my_map(method, argument_list):  
5     result = list()  
6 ▼     for item in argument_list:  
7         result.append(method(item))  
8     return result  
9  
10 my_list = my_map(cube, [1, 2, 3, 4, 5])  
11 print(my_list)
```

[1, 8, 27, 64, 125]

✚ □

First Class Function

— สามารถ Return จาก Function ได้

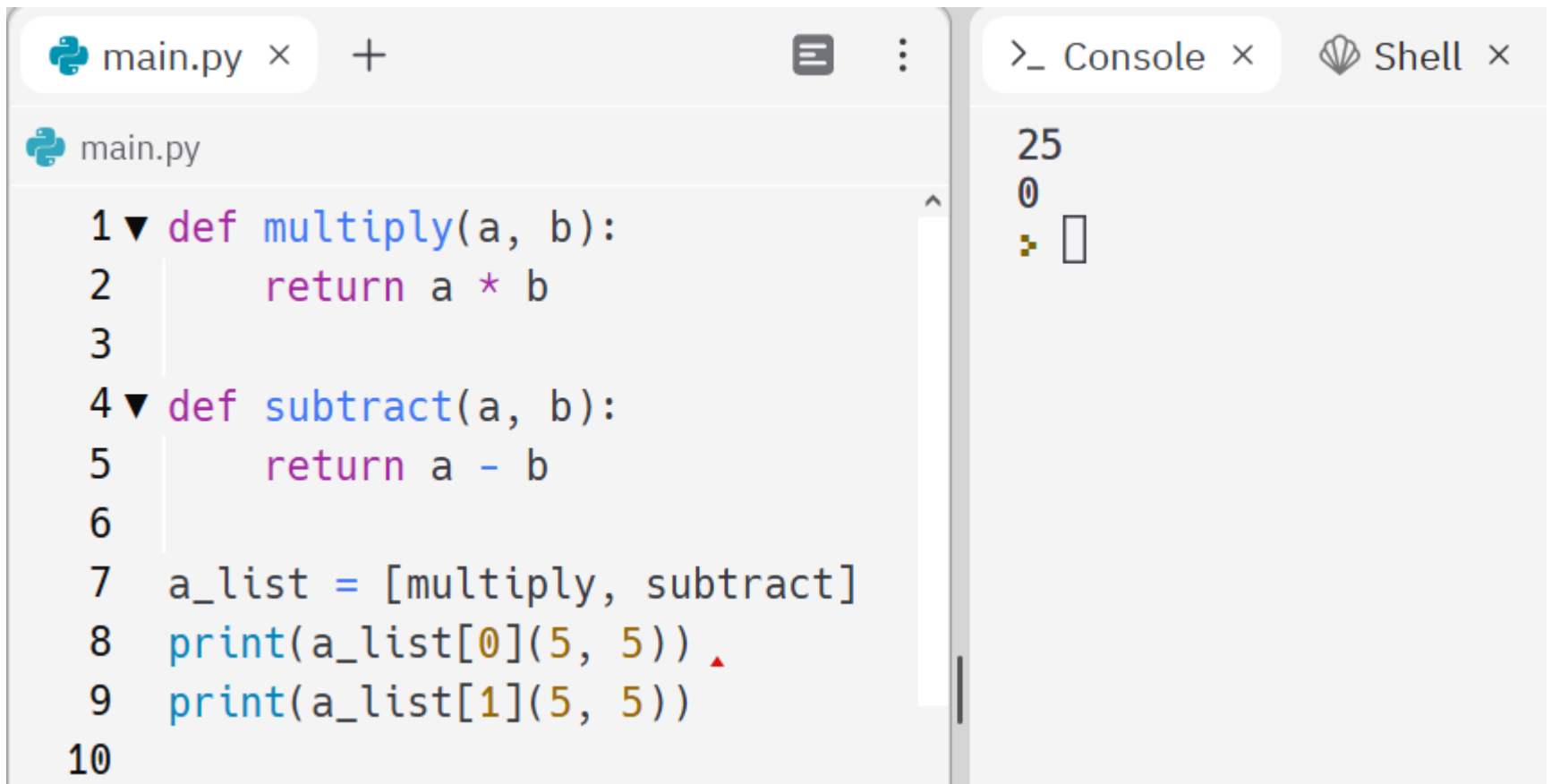


```
main.py x +
main.py
1 ▼ def create_logger(message):
2 ▼     def log():
3       print('Log Message: ' + message)
4       return log #Return a function
5
6 my_logger = create_logger('Hello World')
7 my_logger()
```

```
>_ Console x Shell x +
Log Message: Hello World
>
```

First Class Function

- สามารถเก็บค่าใน List, Tuple หรือโครงสร้างข้อมูลอื่นๆ



```
main.py x +
main.py
1 ▼ def multiply(a, b):
2     return a * b
3
4 ▼ def subtract(a, b):
5     return a - b
6
7 a_list = [multiply, subtract]
8 print(a_list[0](5, 5))
9 print(a_list[1](5, 5))
10
```

The screenshot shows a Python IDE with a file named 'main.py' open. The code defines two functions, 'multiply' and 'subtract', and stores them in a list 'a_list'. The list is then used to call both functions with arguments (5, 5). The output in the console shows '25' for the first call and '0' for the second call.

>_ Console x Shell x

25
0
+ []

Lambda Function

- จะใช้ในกรณีที่เรต้องการสร้างฟังก์ชันแบบไม่ต้องการตั้งชื่อ
- โครงสร้างของ Lambda Function มีดังนี้

```
lambda argument(s) : expression
```

- ต้องขึ้นต้นด้วยคำว่า lambda จากนั้นตามด้วย argument (ไม่ต้องมีวงเล็บ) จากนั้นจึงเป็นส่วนของคำสั่งที่ต้องการ ดังตัวอย่าง

```
main.py
1 x = lambda a : a + 10
2 print(x(5))
```

15

Lambda Function

- ตัวอย่างการใช้ Lambda Function

```
main.py
1 ▼ def myfunc(n):
2     return lambda a : a * n
3
4 doubler = myfunc(2)
5 tripler = myfunc(3)
6
7 print(doubler(11))
8 print(tripler(11))
9
```

22
33
✎

Lambda Function

- ตัวอย่างการใช้ Lambda Function กับ List Comprehension



The screenshot shows a Python IDE with a file named `main.py`. The code defines a list `is_even_list` using a list comprehension with a lambda function. The lambda function takes an argument `arg` and returns `arg * 10`. The list comprehension iterates over `range(1, 5)`. A `for` loop then iterates over `is_even_list` and prints each item. The console on the right shows the output: 10, 20, 30, 40.

```
1 is_even_list = [lambda arg=x: arg * 10 for x in range(1, 5)]
2
3 for item in is_even_list:
4     print(item())
5
```

Console Output:

```
10
20
30
40
```

Dictionary

- Dictionary เป็นโครงสร้างข้อมูลอีกชนิดหนึ่ง ใช้ { } ในการกำหนด
- Dictionary มีลักษณะของการจับคู่ ซึ่งอ้างอิงโดย key : value การใช้งานจะเป็นรูปแบบ { key : value, key : value }
- Dictionary เป็นโครงสร้างแบบไม่มีลำดับ ดังนั้นจะใช้ index ในการระบุตำแหน่งที่มีลักษณะเป็นตัวเลขเหมือนกับ list ไม่ได้ ในการอ้างอิงข้อมูลจะใช้ค่า key ในการอ้าง
- Dictionary เป็นข้อมูลแบบ Mutable

Dictionary

- ตัวอย่างการสร้าง dictionary

main.py ×

```
1  # empty dictionary
2  my_dict = {}
3
4  # dictionary with integer keys
5  my_dict = {1: 'apple', 2: 'ball'}
6
7  # dictionary with mixed keys
8  my_dict = {'name': 'John', 1: [2, 4, 3]}
9
10 # using dict()
11 my_dict = dict({1:'apple', 2:'ball'})
12
13 # from sequence having each item as a pair
14 my_dict = dict([(1,'apple'), (2,'ball')])
```


Dictionary

- การอ้างอิงสมาชิกใน dictionary จะใช้ key เป็นหลัก
- สามารถอ้างอิงโดยใช้ Index [] (ถ้าไม่พบจะ Error) , หรือใช้ method get (ถ้าไม่พบ จะ return None)

main.py x

```
1 # get vs [] for retrieving elements
2 my_dict = {'name': 'Jack', 'age': 26}
3
4 # Output: Jack
5 print(my_dict['name'])
6
7 # Output: 26
8 print(my_dict.get('age'))
9
10 # Trying to access keys which doesn't exist throws error
11 # Output None
12 print(my_dict.get('address'))
13
14 # KeyError
15 print(my_dict['address'])
```

Console Shell

```
Jack
26
None
Traceback (most recent call last):
  File "main.py", line 15, in <module>
    print(my_dict['address'])
KeyError: 'address'
❏
```

Dictionary

- การเปลี่ยน หรือ เพิ่มข้อมูล
- การเปลี่ยนให้ใช้ในรูปแบบ `dict[key] = value`
- การเพิ่มก็เช่นเดียวกัน โดย dictionary จะตรวจสอบ key ที่มีอยู่เดิม ถ้ามีอยู่เดิม ก็จะเป็นการเปลี่ยน แต่ถ้าไม่มี key นั้น ก็จะเป็นการเพิ่ม

main.py ×

```
1 # Changing and adding Dictionary Elements
2 my_dict = {'name': 'Jack', 'age': 26}
3
4 # update value
5 my_dict['age'] = 27
6 print(my_dict)
7
8 # add item
9 my_dict['address'] = 'Downtown'
10 print(my_dict)
```

Console

Shell

```
{'name': 'Jack', 'age': 27}
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
>
```

Dictionary

- การลบข้อมูล 1) ทำโดยใช้ method pop() จะคืนค่าเป็น value และลบ 2) ทำโดยใช้ method popitem() จะคืนค่า (key, value) 3) method clear() เป็นการลบทั้งหมด

```
main.py x
1 squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
2
3 # remove a particular item, returns its value
4 print(squares.pop(4))
5 print(squares)
6
7 # remove an arbitrary item, return (key,value)
8 print(squares.popitem())
9 print(squares)
10
11 # remove all items
12 squares.clear()
13 print(squares)
14
15 # delete the dictionary itself
16 del squares
17 print(squares)
```

Console Shell

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
(5, 25)
{1: 1, 2: 4, 3: 9}
{}
Traceback (most recent call last):
  File "main.py", line 17, in <module>
    print(squares)
NameError: name 'squares' is not defined
>
```

Dictionary

- `fromkeys()` เป็น method ที่เพิ่มข้อมูลเข้าไปใน dictionary โดยใช้กรณีที่มีหลาย key แต่มี value เดียวกัน เช่น การเพิ่มวิชา โดยมีคะแนนเป็น 0 ทุกวิชา
- `items()` เป็น method ที่ส่งคืน key, value ของแต่ละสมาชิกใน dictionary กลับคืนมา เราสามารถแยก key กับ value ได้ โดยเอาตัวแปร 2 ตัวมารับ เช่น
`for k, v in marks.items()`
- เราสามารถจะดึงเฉพาะค่า key หรือ value ได้โดยใช้ method `keys()` หรือ `values()`

```
main.py x
1 # Dictionary Methods
2 marks = {}.fromkeys(['Math', 'English', 'Science'], 0)
3 print(marks)
4
5 ▼ for item in marks.items():
6     print(item)
7
8 print(list(sorted(marks.keys())))
```

```
Console Shell
{'Math': 0, 'English': 0, 'Science': 0}
('Math', 0)
('English', 0)
('Science', 0)
['English', 'Math', 'Science']
>
```

Dictionary

- ตัวอย่างการใช้งาน dictionary โดยการสร้างเครื่องคิดเลขอย่างง่าย
- จะเริ่มจากสร้าง ฟังก์ชัน บวก ลบ คูณ หาร
- และสร้าง dictionary เพื่อเก็บ
 - key : เครื่องหมาย
 - value : function

```
13 ▼ operations = {'+' : add,  
14                  '-' : subtract,  
15                  '*' : multiply,  
16                  '/' : devide}  
17
```

main.py ×

```
1 ▼ def add(n1, n2):  
2     return n1+n2  
3  
4 ▼ def subtract(n1, n2):  
5     return n1-n2  
6  
7 ▼ def multiply(n1, n2):  
8     return n1*n2  
9  
10 ▼ def devide(n1, n2):  
11     return n1/n2  
12
```

Dictionary

- จากนั้นก็เขียนโปรแกรมรับตัวเลขมาคำนวณ จะเห็นว่าในบรรทัดที่ 27 เราสามารถ assign ตัวแปรให้มาชี้ฟังก์ชันที่ตรงกับเครื่องหมายที่เลือก และ สั่งให้ทำงานในบรรทัดที่ 26

```
main.py x
18 num1 = float(input("What's the first number?: "))
19 ▼ for symbol in operations:
20     print(symbol)
21
22 should_continue = True
23
24 ▼ while should_continue:
25     operation_symbol = input("Pick an operation: ")
26     num2 = float(input("What's the next number?: "))
27     calculation_function = operations[operation_symbol]
28     answer = calculation_function(num1, num2)
29     print(f"{num1} {operation_symbol} {num2} = {answer}")
30
31 ▼ if input(f"Type 'y' to continue calculating with {answer}, or
    type 'n' to start a new calculation: ") == 'y':
32     num1 = answer
33 ▼ else:
34     should_continue = False
```

Console Shell

```
What's the first number?: 25
+
-
*
/
Pick an operation: *
What's the next number?: 25
25.0 * 25.0 = 625.0
Type 'y' to continue calculating with 625.0,
art a new calculation: 
```

Dictionary

- การสร้าง Dictionary 2 มิติ
- ใน Dictionary แบบ 1 มิติ เช่น dictionary สำหรับเก็บคะแนนของนักศึกษา 1 คน

```
sdict1 = {'python': 40, 'calculus': 45}
```

จะมี 2 Element คือ `sdict[0] = 'python': 50` และ `sdict[1] = 'calculus': 55`
- หากจะเพิ่มนักศึกษาคนที่ 2 ก็ต้องเขียนเป็น `sdict2 = {'python': 50, 'calculus': 55}`
- หากต้องการโครงสร้างข้อมูลสำหรับเก็บคะแนนของนักศึกษาหลายๆ คนอาจเขียนเป็น

```
st_score = {'65015001': sdict1, '65015002': sdict2 }
```
- ก็จะกลายเป็น Dictionary 2 มิติ หรือ Nested Dictionary ทันที
- ในการอ้างถึง ถ้าอ้างถึง `st_score['65015001']` จะหมายถึง `sdict1` หรือ `{'python': 40, 'calculus': 45}` ถ้าอ้าง `st_score['65015001']['python']` ก็จะหมายถึง 40

Dictionary

- สำหรับการเพิ่มข้อมูลใน dictionary 2 มิติ อาจทำได้ดังนี้
 - `st_score['65015003'] = {}`
 - `st_score['65015003']['python'] = 30`
 - `st_score['65015003']['calculus'] = 35`
- หรือ
`st_score['65015004'] = {'python': 42, 'calculus': 25}`
- ในการเขียนโปรแกรมจะมีความซับซ้อนขึ้น เนื่องจากต้องแยกให้ดีกว่า Key ใดเป็น Key ของ Dictionary ใด และจะได้ Value ออกมาเป็นอะไร

Dictionary

- ตัวอย่าง สมมติว่านักศึกษาจะจัด Party โดยให้เพื่อนแต่ละคนคิดสูตรขนมเค้กมาคนละ 1 สูตร จากนั้นรวมไว้ใน dictionary เช่น

```
recipes = {  
    'dusty cake' : {'cup of sugar':4, 'cup of flour':4},  
    'eggy cake' : {'egg':1 },  
    'eggier cake' : {'egg':10 },  
    'mega egg' : {'egg':100 },  
    'chocolate cake' : {'egg':2, 'cup of sugar':2, 'chocolate':1}  
}
```

- สมาชิกแรก (key) ของ dictionary คือ ชื่อของเค้ก สมาชิกที่ 2 (value) คือ สูตรที่ใช้ในการทำ เช่น dusty cake จะประกอบด้วย น้ำตาลทราย (cup of sugar) จำนวน 4 ถ้วย และแป้งสาลี (cup of flour) จำนวน 4 ถ้วย

Dictionary

- ข้อมูลอีกข้อมูลหนึ่ง คือ ข้อมูลของวัตถุดิบในครัว จะเก็บใน dictionary เช่นกัน เช่น

```
alice_pantry = {'egg':12, 'cup of sugar':4, 'cup of flour':4}  
bob_pantry = {'egg':12, 'cup of sugar':4, 'chocolate':5}
```

- หมายถึงในครัวของ alice มีไข่ 12 ฟอง มีน้ำตาลทราย 4 ถ้วย และแป้งสาลี 4 ถ้วย แต่ในครัวของ bob มีไข่ 12 ฟอง มีน้ำตาลทราย 4 ถ้วย และช็อกโกแลต 5 แท่ง
- กลุ่มเพื่อนกำลังตัดสินใจว่าจะไป party ที่บ้านใคร โดยจะพิจารณาจากไปบ้านใครแล้วทำขนมเค้กได้หลายแบบที่สุด เพื่อนจึงมอบหมายให้นักศึกษาเขียนโปรแกรมเพื่อหาว่าในครัวของบ้าน สามารถทำขนมได้หรือไม่ โดยรับข้อมูลเป็น pantry และ recipes

Dictionary

```
def find_bakeable(pantry, recipes):
    bakeable_list = []
    for cake_name in recipes.keys():
        cake_dict = recipes[cake_name]
        bakeable = True
        for ingredient in cake_dict.keys():
            if ingredient not in pantry.keys() or \
                pantry[ingredient] < cake_dict[ingredient]:
                bakeable = False
        if bakeable:
            bakeable_list.append(cake_name)
    return bakeable_list
```

Dictionary comprehension

- กรณีของ dictionary comprehension ก็เช่นเดียวกัน แต่เป็นการสร้างใส่ dictionary แทนที่จะเป็น list แต่เนื่องจาก dictionary มีความซับซ้อนมากกว่า ดังนั้นจึงทำให้อ่านยากขึ้นไปอีก
- รูปแบบการทำงานของ dictionary comprehension มีดังนี้

```
dictionary = {key: value for vars in iterable}
```

```
{ key: value for vars in iterable }
```

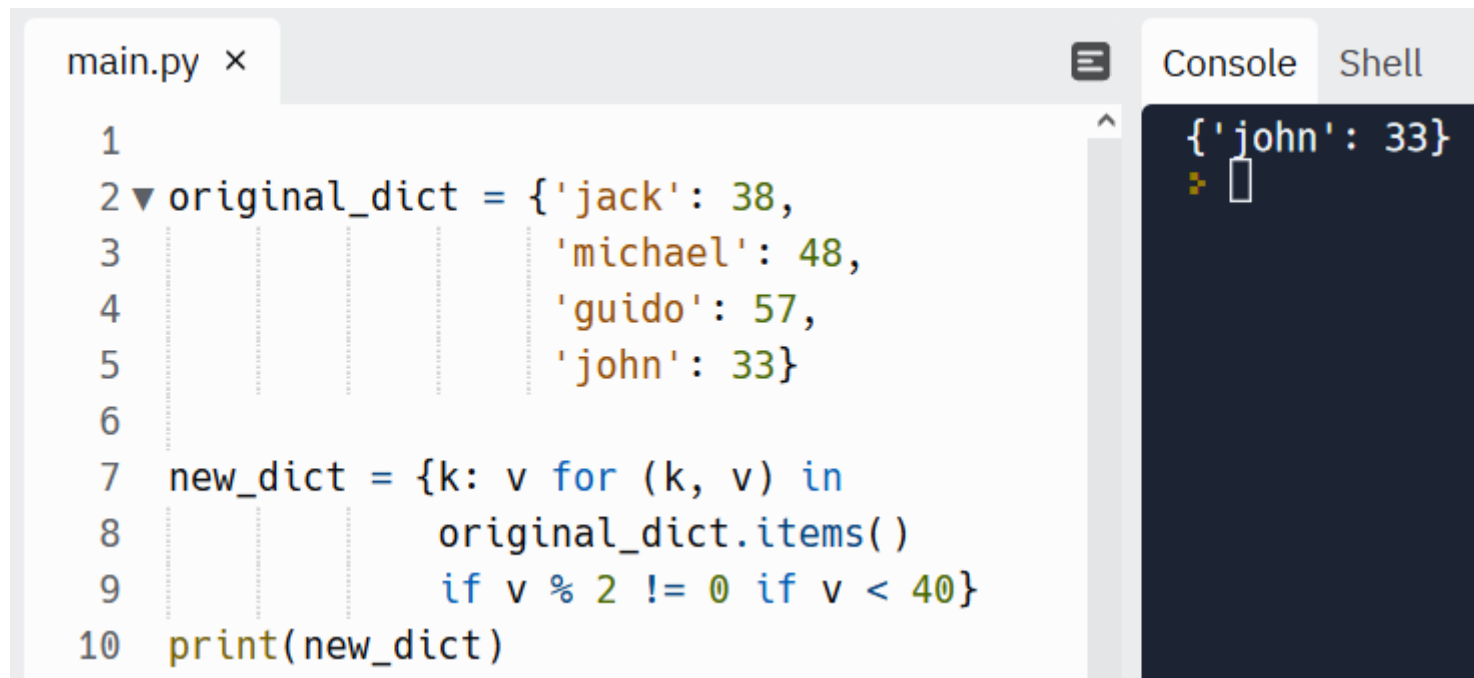
Diagram illustrating the components of a dictionary comprehension:

```
{ num: num*num for num in range(1, 11) }
```

- `key`: The expression used to generate the keys of the dictionary.
- `value`: The expression used to generate the values of the dictionary.
- `vars`: The variable(s) used in the key and value expressions.
- `iterable`: The iterable used to generate the values of the dictionary.

Dictionary comprehension

- ตัวอย่าง กรณีมี 2 เงื่อนไข คือ ต้องเป็นเลขคี่ และ ต้องมี value น้อยกว่า 40



The screenshot shows a Python IDE with a file named `main.py`. The code defines an original dictionary and a new dictionary using a comprehension with two conditions. The console output shows the resulting dictionary.

```
1
2 ▼ original_dict = {'jack': 38,
3                     'michael': 48,
4                     'guido': 57,
5                     'john': 33}
6
7 new_dict = {k: v for (k, v) in
8              original_dict.items()
9              if v % 2 != 0 if v < 40}
10 print(new_dict)
```

Console output:

```
{'john': 33}
```

Dictionary comprehension

- ตัวอย่างกรณี ใช้ if else

main.py x

```
1 ▼ original_dict = {'jack': 38,  
2                     'michael': 48,  
3                     'guido': 57,  
4                     'john': 33}  
5  
6 new_dict_1 = {k: ('old' if v > 40  
7                  else 'young')  
8               for (k, v) in  
9                   original_dict.items()  
10  
11 print(new_dict_1)
```



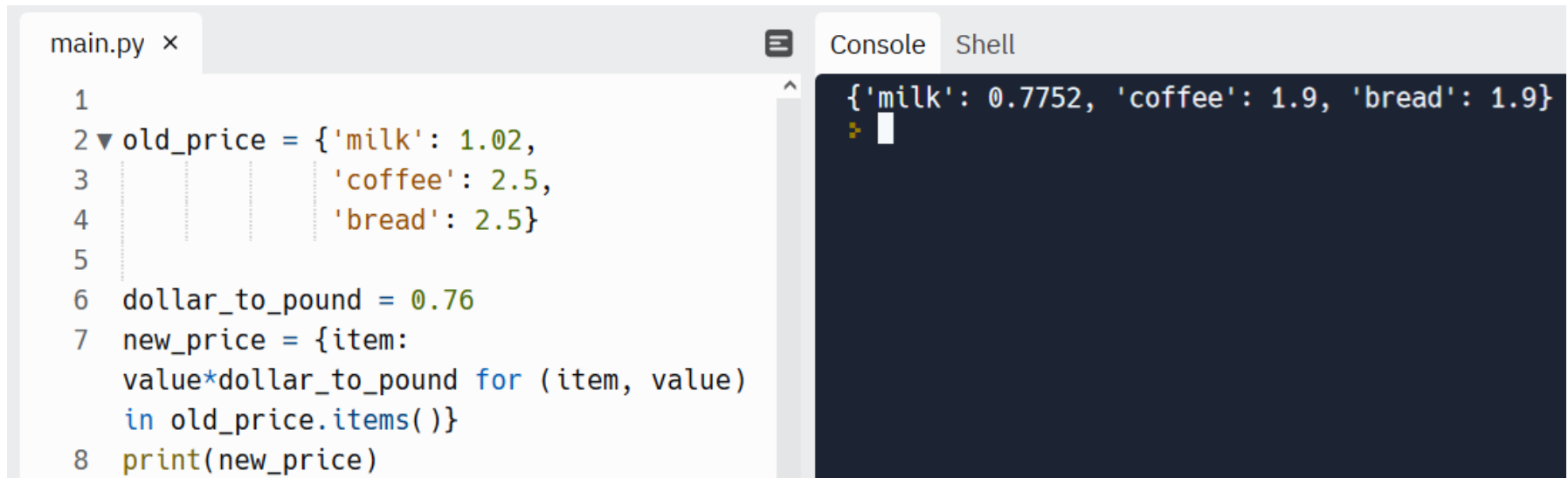
Console

Shell

```
{'jack': 'young', 'michael': 'old', 'guido': 'old', 'john': 'young'}
```

Dictionary comprehension

- ตัวอย่าง เป็นการสร้าง dictionary ใหม่ ที่เก็บราคาเครื่องดื่ม จากเดิมเป็น dollar เป็น ปอนด์



The screenshot shows a Python IDE with a file named `main.py` and a console window. The code in `main.py` defines a dictionary `old_price` with items 'milk', 'coffee', and 'bread' in dollars. It then uses a dictionary comprehension to create `new_price` by multiplying each value by the conversion rate `dollar_to_pound = 0.76`. The console output shows the resulting dictionary: `{'milk': 0.7752, 'coffee': 1.9, 'bread': 1.9}`.

```
main.py x
1
2 ▼ old_price = {'milk': 1.02,
3               'coffee': 2.5,
4               'bread': 2.5}
5
6 dollar_to_pound = 0.76
7 new_price = {item:
8             value*dollar_to_pound for (item, value)
9             in old_price.items()}
10 print(new_price)

Console Shell
{'milk': 0.7752, 'coffee': 1.9, 'bread': 1.9}
```

Exception

- โปรแกรมที่ดีควรจะให้การทำงานที่ถูกต้องเสมอ แต่บางครั้งก็อาจเกิด Error เช่น Error จาก Input ที่ป้อนเข้ามาไม่ตรงตามกำหนด หรือ access ข้อมูลเกินขอบเขต
- Interpreter ของ Python จะจัดการ Error โดยการ raise Exception ขึ้นมา พร้อมกับ Error Message
- Exception ของ Python มีประเภทมากมาย โดยสามารถดูทั้งหมดได้จาก https://www.w3schools.com/python/python_ref_exceptions.asp

Exception

- Exception **SyntaxError** จากการเขียนโปรแกรมผิด Syntax

```
print "hello word"
```

```
File "C:\Users\khtha\AppData\Local\Temp\ipykernel_12568/345270128.py",  
  print "hello word"  
    ^
```

SyntaxError: Missing parentheses in call to 'print'. Did you mean print("hello word")?

- Exception **ZeroDivisionError** จากการหารด้วย 0

```
x = 5 / 0
```

```
-----  
ZeroDivisionError                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_12568/1806623527.py in <module>  
----> 1 x = 5 / 0
```

ZeroDivisionError: division by zero

Exception

- Exception **IndexError** จากการอ้าง Index ที่อยู่นอกขอบเขต

```
lst = [1,2,3]  
print(lst[3])
```

IndexError

Traceback (most recent call last)

~\AppData\Local\Temp\ipykernel_12568\1052905902.py in <module>

1 lst = [1,2,3]

----> 2 print(lst[3])

IndexError: list index out of range

Exception

- Exception **TypeError** จากการใช้ข้อมูลผิดประเภท

```
lst = [1,2,3]  
lst + 2
```

```
-----  
TypeError                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_12568\3418216332.py in <module>  
      1 lst = [1,2,3]  
----> 2 lst + 2
```

TypeError: can only concatenate list (not "int") to list

Exception

- Exception **AttributeError** จากการอ้างถึงตัวแปรหรือ Attribute ที่ไม่มี

```
lst = [1,2,3]  
lst.add
```

AttributeError

Traceback (most recent call last)

~\AppData\Local\Temp\ipykernel_12568\453125606.py in <module>

1 lst = [1,2,3]

----> 2 lst.add

AttributeError: 'list' object has no attribute 'add'

Exception

- Exception **KeyError** จากการใช้คีย์ที่ไม่พบใน Dictionary

```
d = {'a': 'hello'}  
d['b']
```

```
-----  
KeyError                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_12568\3661029466.py in <module>  
      1 d = {'a': 'hello'}  
----> 2 d['b']
```

KeyError: 'b'

- Exception **NameError** จากการใช้ตัวแปรที่ไม่ได้กำหนดขึ้น

```
print(this_is_not_a_var)
```

```
-----  
NameError                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_12568\2762618246.py in <module>  
----> 1 print(this_is_not_a_var)
```

NameError: name 'this_is_not_a_var' is not defined

Exception

- เราสามารถสร้าง Exception ได้เองด้วย โดยใช้คำสั่ง raise

Use raise to force an exception:



- เช่น

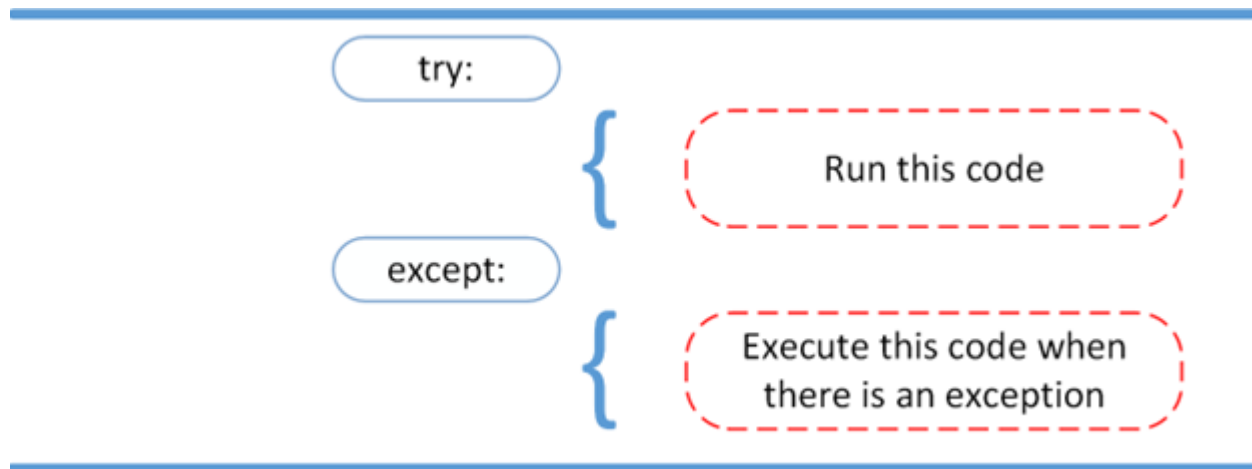
```
x = -1

if x < 0:
    raise Exception("Sorry, no numbers below zero")
```

```
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    raise Exception("Sorry, no numbers below zero")
Exception: Sorry, no numbers below zero
```

Exception

- ในการเขียนโปรแกรม อาจใช้ประโยชน์จาก Exception ในการแจ้งให้ทราบว่าโปรแกรมมีที่ผิดพลาดตรงไหน
- แต่การเขียน Software เราจะปล่อยให้เกิด Exception ไม่ได้ เพราะทำให้โปรแกรมหยุดทำงาน ดังนั้นเราจะต้องดัก Exception เอาไว้ โดยใช้ Try
- คำสั่ง Try จะรัน code block ส่วน try และจะดัก Exception เอาไว้ หากไม่มี Exception ก็จะทำตามปกติ แต่หากมี Exception จะไปทำใน block except:



Exception

- จากตัวอย่าง จะเห็นว่าแม้จะเกิดความผิดพลาดขึ้น แต่โปรแกรมจะไม่แสดงออกมาเป็น Error แต่จะ print error message ที่ตั้งเอาไว้ออกมาแทน
- การทำแบบนี้ทำให้โปรแกรมไม่หยุด และสามารถควบคุมการทำงานต่อได้



```
main.py
1 x = 5
2 y = 0
3
4 ▼ try:
5     x/y
6 ▼ except:
7     print("An exception occurred")
```

An exception occurred

Exception

- การดัก Exception สามารถระบุได้ว่าจะดัก Exception ชนิดใด

main.py

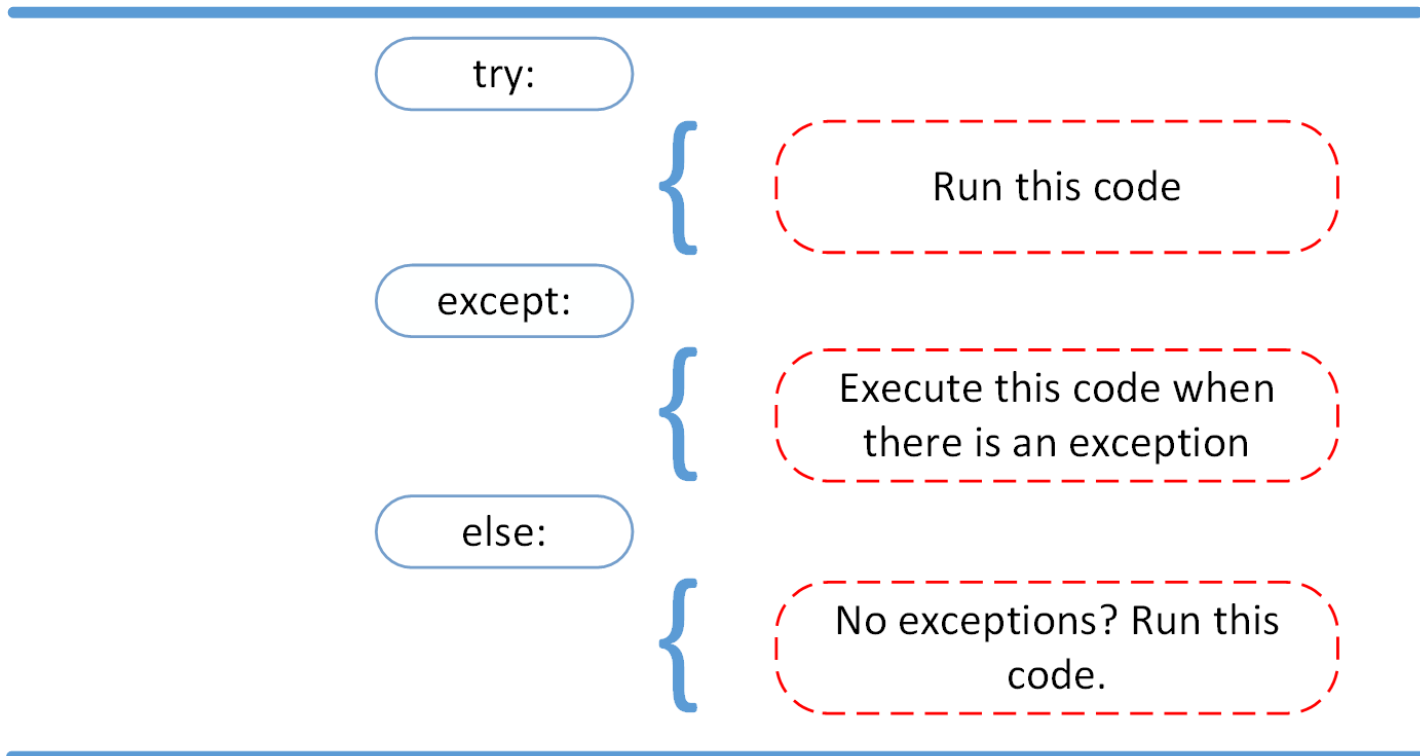
```
1 y = 0
2
3 ▼ try:
4     x/y
5 ▼ except NameError:
6     print("Variable x is not defined")
7 ▼ except ZeroDivisionError:
8     print("Zero is not a good idea.")
9 ▼ except:
10    print("Something else went wrong")
```

Variable x is not defined



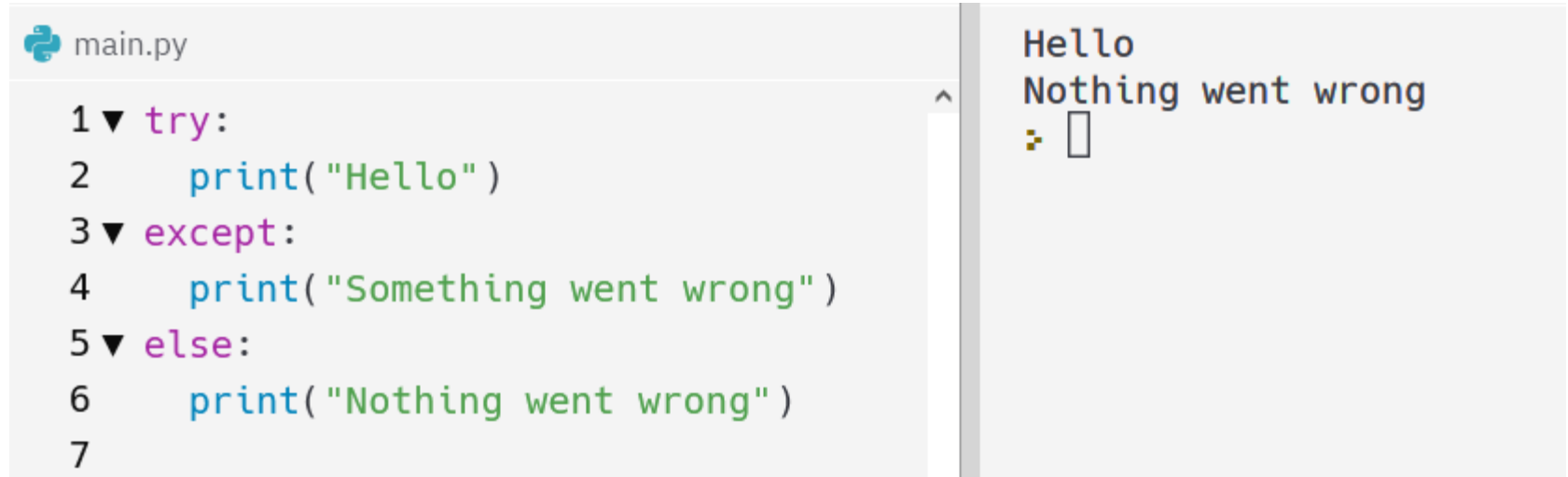
Exception

- นอกเหนือจาก try except แล้ว ยังมี else clause อีกด้วย โดยมีความหมาย คือ หากไม่มี Exception เกิดขึ้นให้รันใน else clause แทน



Exception

- ตัวอย่าง

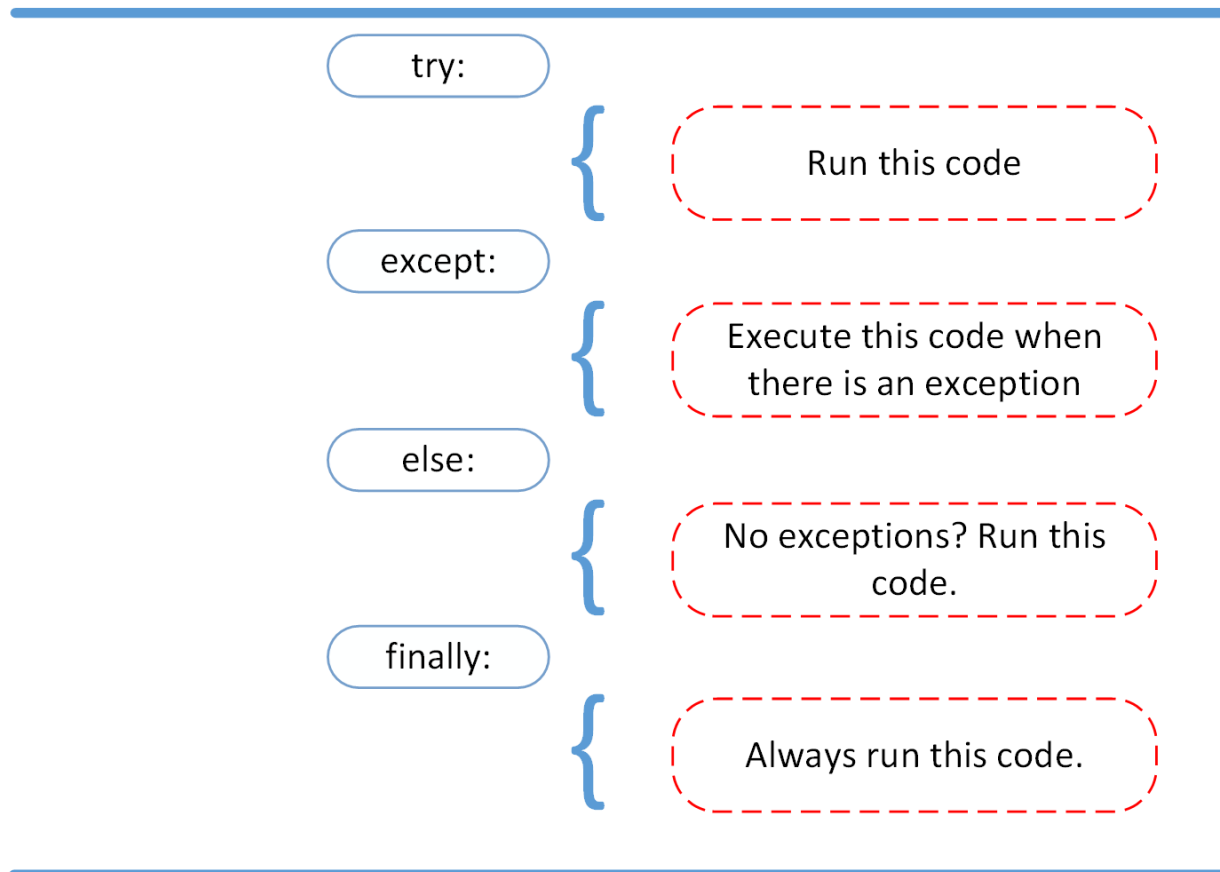


```
main.py
1 ▼ try:
2     print("Hello")
3 ▼ except:
4     print("Something went wrong")
5 ▼ else:
6     print("Nothing went wrong")
7
```

Hello
Nothing went wrong

Exception

- ท้ายสุด คือ finally course จะทำงานในทุกกรณี



Exception

- finally มักใช้ในการจัดการกับงานส่วนปิดท้าย เช่น
 - จัดการ database connection ที่ค้างไว้
 - ปิดไฟล์
 - ส่งข้อมูลในระบบเครือข่ายเพื่อแจ้งข้อมูล



For your attention