CH08
函式 (Function)

FANTAGEEKS
大福媽瘋程式

課程重點:

8.1 Define functions

8.2 Function Examples

8.3 Variables in four scopes

8.4 Useful Built-in Methods

大福媽瘋程式
FANTAGEEKS

8.1

Define Functions

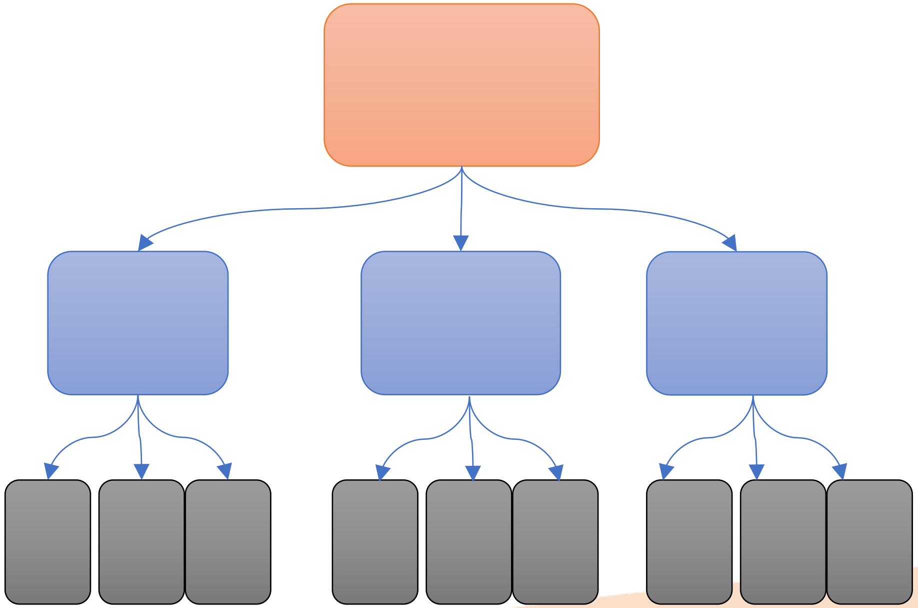
Why to have functions:

- Creating clean **repeatable code** is a key part of becoming an effective programmer.
- A function is a block of organized, reusable code that is used to **perform a single and related action**.
- Functions allow us to easily execute blocks of code many times.

Why to have functions:

- Most programs perform tasks that are large enough to be broken down into subtasks. Every subtask could be a function.
- Not to write the large set of statements. The better is to break down a program into several small functions, allowing us to *"divide and conquer"* a programming problem

Divide and Conquer (分而治之)



化繁為簡的程式美學

Two types of functions :

- Build-in Function (內建函數):
  - Python predefine the build-in functions to be called for supporting the tasks
  - >>>dir(__builtins__) 查詢有哪些內建函數使用

- Define Function (自訂函數):
  - **Programmer can define functions to have block of codes according to subtasks.**

Introduction to functions:

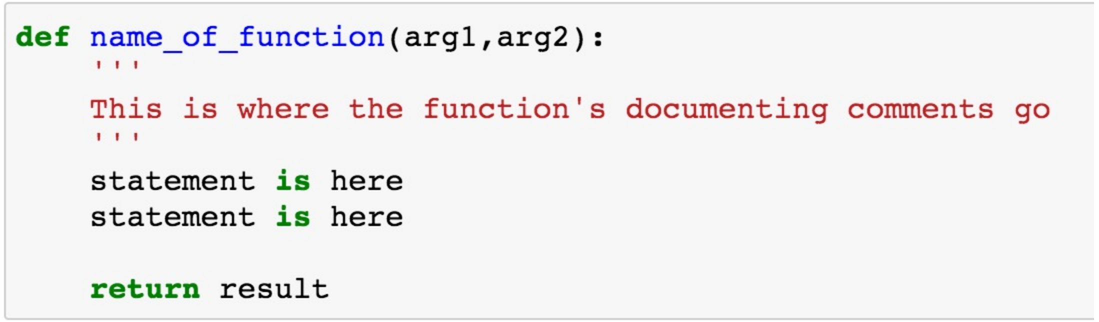
- Function is to group together a set of statements so they can be run more than once.
- Functions can also let us specify parameters that can serve as inputs to the functions.
- The reason is to help not have to repeatedly write the same code again and again.

Define function:

define function→ call function → return value

```
#defining function, function name is myname
def myname ( ):
     print("this is inside the function")
     print("Just printing")

     #after print, return "OK"
     return "OK"

#call function
mystr = myname()

#variable receives return value
print(mystr)
```

Define function :



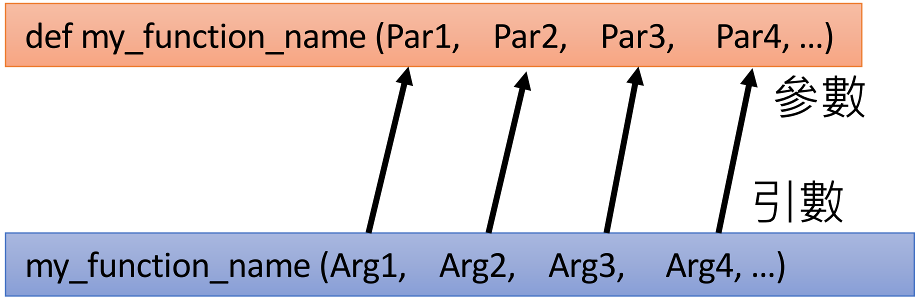
```
def name_of_function(arg1,arg2):
    '''
    This is where the function's documenting comments go
    '''
    statement is here
    statement is here

    return result
```

- Functions, like variables must be **named and created** before calling functions.
- Call function *name_of_function(arg1, arg2)* to execute the lines inside of function
- Arguments are **the inputs** for your function. Use these inputs inside the function and reference them.
- Once a function has completed, Python will *return result directly* to the line of the initial function call.
- *Return* a result that can then be stored as a variable

Example: Flow of Execution with Functions

```
# Examing flow of execution

def hello():
    print ("Hi, there")
    print ("This is a hello function inside")


#end of hello() function


print ("Good morning")
print ("Today we're talking to function !")
hello ()
print ("And now we're done.")
print ("Goodbye Goodbye !")
```

Functions with positional arguments:


def my_function_name (Par1, Par2, Par3, Par4, ...)
參數
引數
my_function_name (Arg1, Arg2, Arg3, Arg4, ...)

- Argument (引數): 呼叫函式時，要傳遞給函式的值
- Parameter (參數): 函數被呼叫參照時，需要接收到資料，讓函數成功運行
- 函數呼叫時，必須依照函數所需要的參數，**依序給引數**！

Functions with return value

- When finishing the end of the function, return the value (result) or ignore (no return statement)
- “Return something” means ending the function with return value.（碰到Return，無條件結束函式）
- Return value could be:
  - **Numeric type**
  - **String**
  - **List / tuple**
  - **None (if no return statement)**

大福媽瘋程式
FANTAGEEKS

8.2

Function Examples:

Example:

```
def myfunc (x, y, z):
    mysum = x+y+z
    myavg = sum/3

    return (myavg)


ans = myfunc (15, z=-6, y=-3)
print (ans)
```

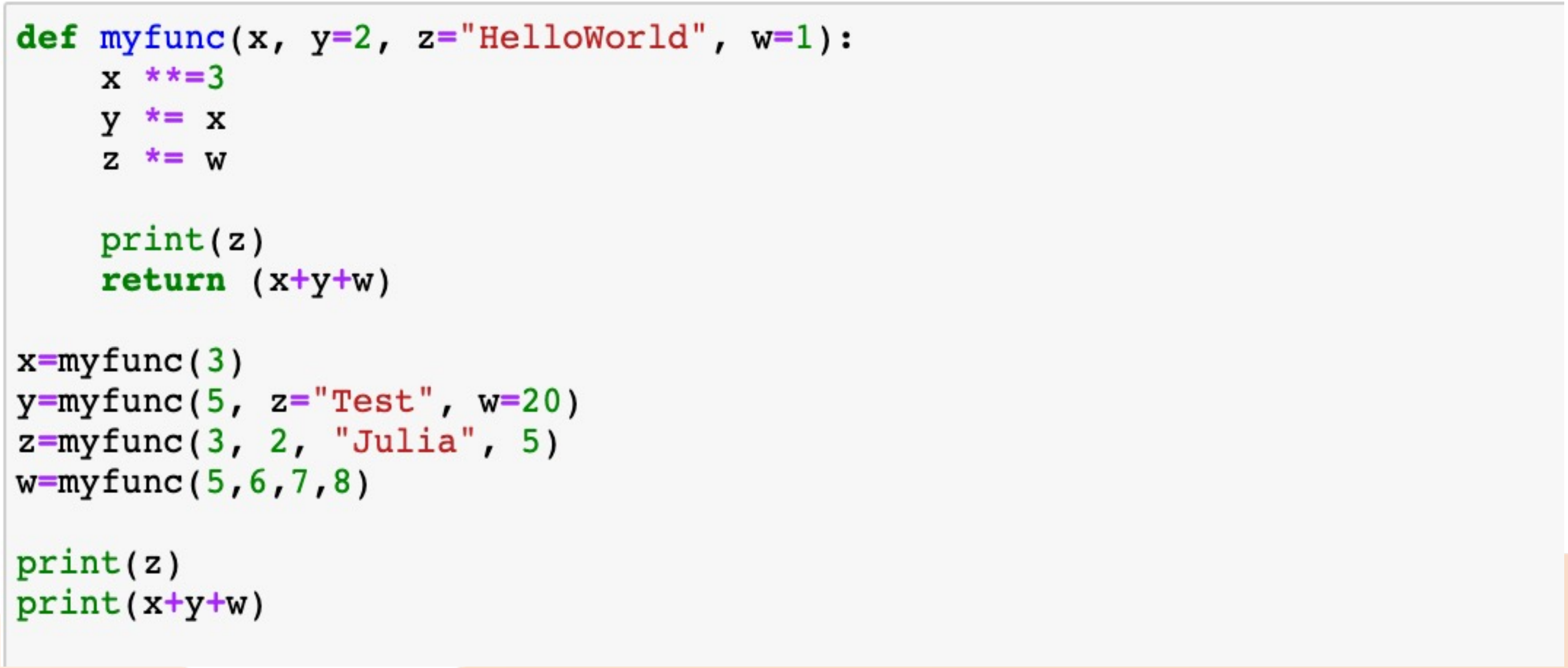
Example:

```
def myfunc (x, y, z):
    mysum = x+y+z
    myavg = sum/3

    return (2*x, 4*y, 6*z, mysum, myavg)


ans = myfunc (2, 4, 6)
print (ans)
```

Example:

```
def myfunc (x, y=3 z="Hello World"):
    print(z)
    return x+y

ans1 = myfunc (5, 6, "I want You")
ans2 = myfunc (5, 6)
ans3 = myfunc (5)

print(ans1)
print(ans2)
print(ans3)
```

Example:



```python
def myfunc(x, y=2, z="HelloWorld", w=1):
    x **=3
    y *= x
    z *= w

    print(z)
    return (x+y+w)

x=myfunc(3)
y=myfunc(5, z="Test", w=20)
z=myfunc(3, 2, "Julia", 5)
w=myfunc(5,6,7,8)

print(z)
print(x+y+w)
```

which x you're referring to in your code?

```
x = 25

def printer( ):
    x = 50
    return x

print(x)
print(printer())
```

Example:

```python
def square(x):
    return x*x

def applier (q, x):
    return q(x)

num = applier (square, 7)
print(num)
```

大福媽瘋程式
FANTAGEEKS

8.3

Variables in four scopes:

Variables in four scopes:

- The idea of scope in your code is decided in order to properly assign and call variable names.
- Variable name references in four scopes:
  - Local: Names assigned in any way within a function (def or lambda), and not declared global in that function.
  - enclosing functions: Names in the local scope of any and all enclosing functions (def or lambda), from inner to outer.
  - Global: Names assigned at the top-level of a module file, or declared global in a def within the file.
  - built-in: Names preassigned in the built-in names module : open, range, print, input. Cannot overwrite it.

8.3.1 Local Variable:

- Functions are like mini programs. You can create variables inside functions.
- Variables that are defined inside of a function are called “local” variables
- Local variables that they only exist within the function. Outside of SCOPE of the function will not be able to access the local variables

```
def power1(x):
    x=x**2
    return x
```

Local Variable:

- Different functions can have their own local variables that use the same variable name.

- These local variables will not overwrite one another because they exist in different functions and different scopes

Local Variable:

```
def power1(x):
    x=x**2
    return x

def power2(x):
    x=x**3
    return x

ans1 = power1(4)
ans2 = power2(4)

print(ans1)
print(ans2)
```

8.3.2 Enclosing function locals:

- Names in the local scope of any and all enclosing functions (nested functions), from inner to outer.

Enclosing function locals

```
name = 'This is a global name'

def greet():
    name = 'Sammy`
    print('Hello 1'+name)

    def hello():
        print('Hello 2'+name)

    hello()

greet()
print(name)
```

8.3.3 Global Variable

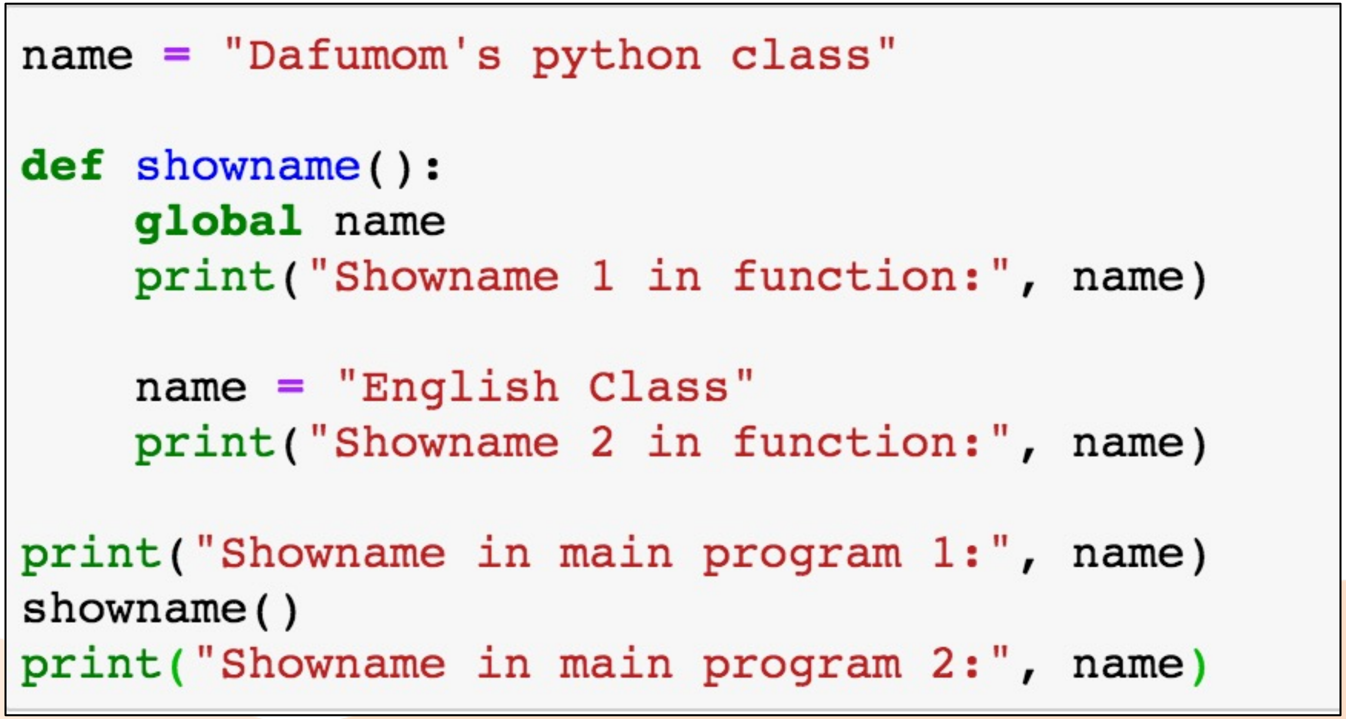
- When a variable is created outside all of your functions, it is considered a “global variable”.
- Global variables can be accessed by any statement in your program file.
- You also can access global variables in any function, but cannot change

```
name = ”python course”

def showname():
    print(“show name in function:”, name)

print(“show name in main:”, name)
showname()
```

Global Variables

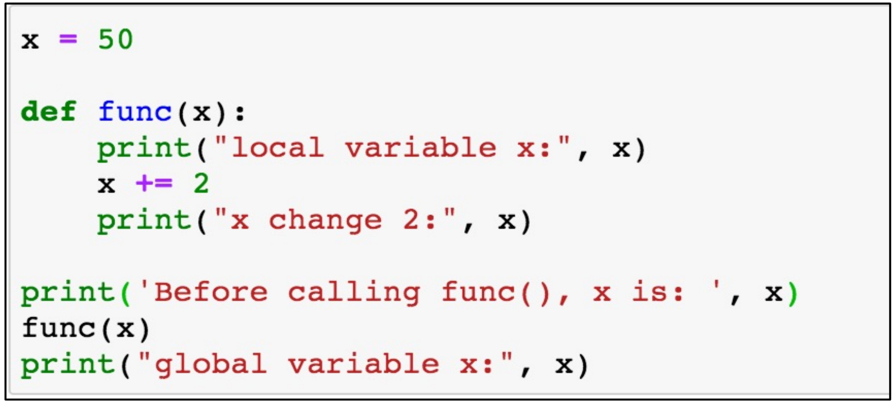
- If you want to be able to change a global variable inside of a function, you must first tell Python using the “global” keyword inside the function

Global Variables

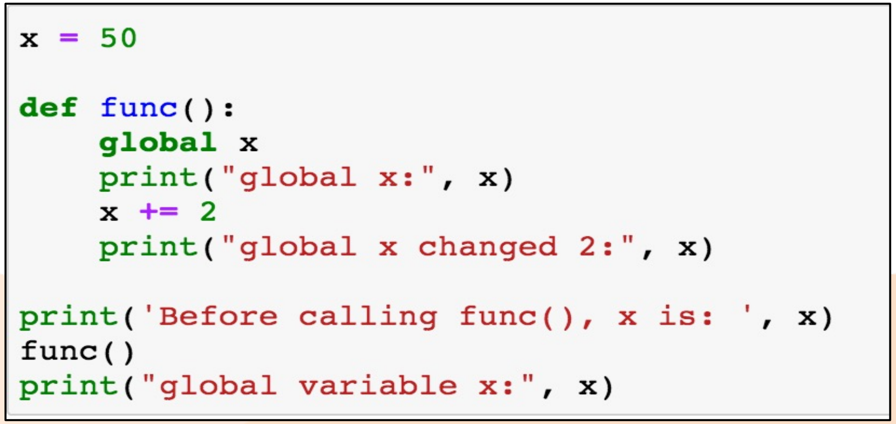


```python
name = "Dafumom's python class"

def showname():
    global name
    print("Showname 1 in function:", name)

    name = "English Class"
    print("Showname 2 in function:", name)

print("Showname in main program 1:", name)
showname()
print("Showname in main program 2:", name)
```

Global Variables



```python
x = 50

def func(x):
    print("local variable x:", x)
    x += 2
    print("x change 2:", x)

print('Before calling func(), x is: ', x)
func(x)
print("global variable x:", x)
```



```python
x = 50

def func():
    global x
    print("global x:", x)
    x += 2
    print("global x changed 2:", x)

print('Before calling func(), x is: ', x)
func()
print("global variable x:", x)
```

IPO Notation:

- As you start writing more advanced functions, please document the functions based on their Input, Processing and Output (IPO)

```
# function: add_ages
# input: age1 (integer), age2 (integer)
# processing:  combines the two integers
# output: returns the combined value
def add_ages(age1, age2):
     sum = age1+age2
     return sum
```

大福媽瘋程式
FANTAGEEKS

8.4

Useful Built-in Methods

Built-in Methods

- Methods are essentially functions built into objects.
- Methods perform specific actions on an object and can also take arguments, just like a function.

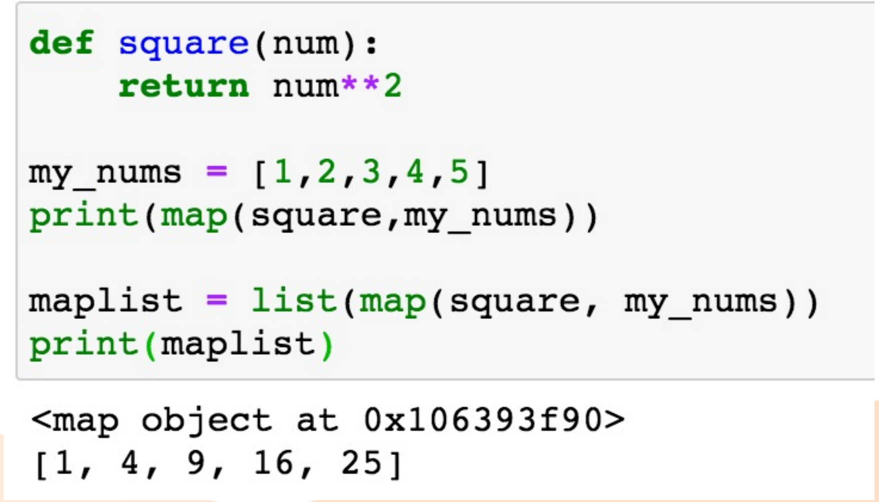
```
object.method(arg1,arg2, ..etc)
```

Useful function:

- **Enumerate():** enumerate was created to enum ***index*** and ***object*** enumerate(sequence, [start=0])
- **Zip()**: quickly create a list of tuples by "zipping" up together two lists.
- **Min() and max() :** Quickly check the minimum or maximum of a list

map function

- The map function allows you to "map" a function to an iterable object.
- Quickly call the same function to every item in an iterable



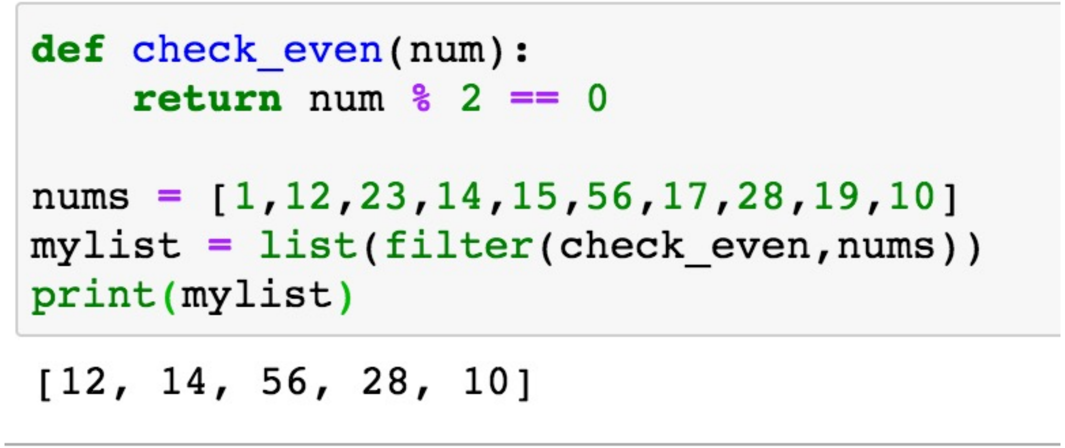
```python
def square(num):
    return num**2

my_nums = [1,2,3,4,5]
print(map(square,my_nums))

maplist = list(map(square, my_nums))
print(maplist)
```

```
<map object at 0x106393f90>
[1, 4, 9, 16, 25]
```

Filter function

- The filter function **returns an iterator** yielding those items of iterable for which function(item) is true.
- First filter by a function that returns either True or False. Then passing that into filter (along with your iterable) .
- Get back only the results that would return True then passed to the function.

Filter function



```
def check_even(num):
    return num % 2 == 0

nums = [1,12,23,14,15,56,17,28,19,10]
mylist = list(filter(check_even,nums))
print(mylist)
```

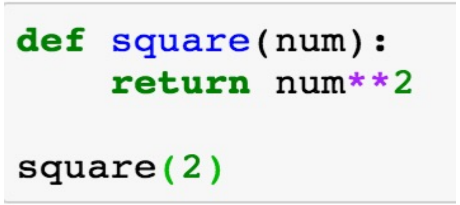
```
[12, 14, 56, 28, 10]
```

Lambda Expression:

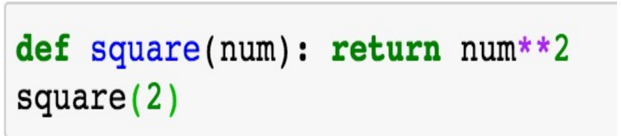
- lambda expressions allow us to create ”no-name " functions. We can quickly make ad-hoc functions without needing to properly define a function using def.

- **lambda's body is a single expression, not a block of statements.**

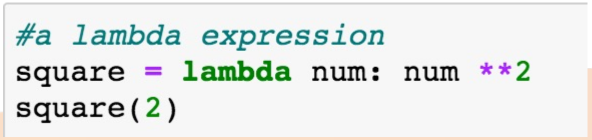
lambda expression : How it creates !



```python
def square(num):
    return num**2

square(2)
```

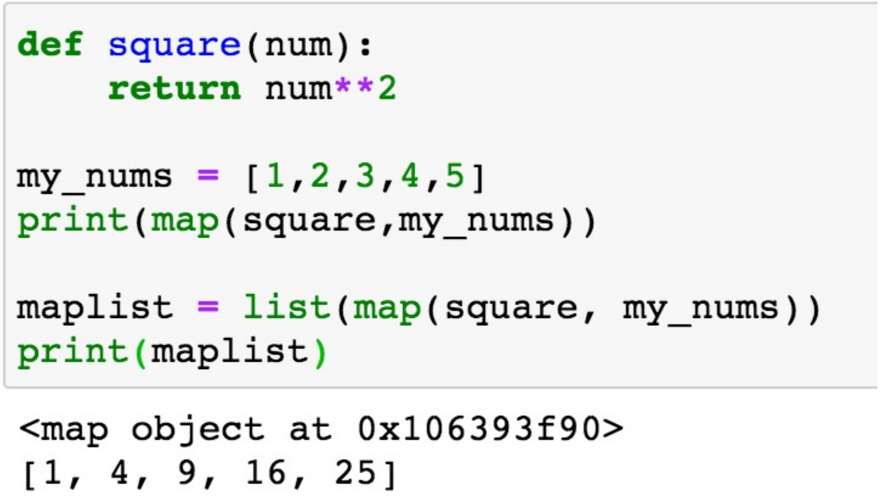


```python
def square(num): return num**2
square(2)
```



```python
#a lambda expression
square = lambda num: num **2
square(2)
```

Lambda expression:

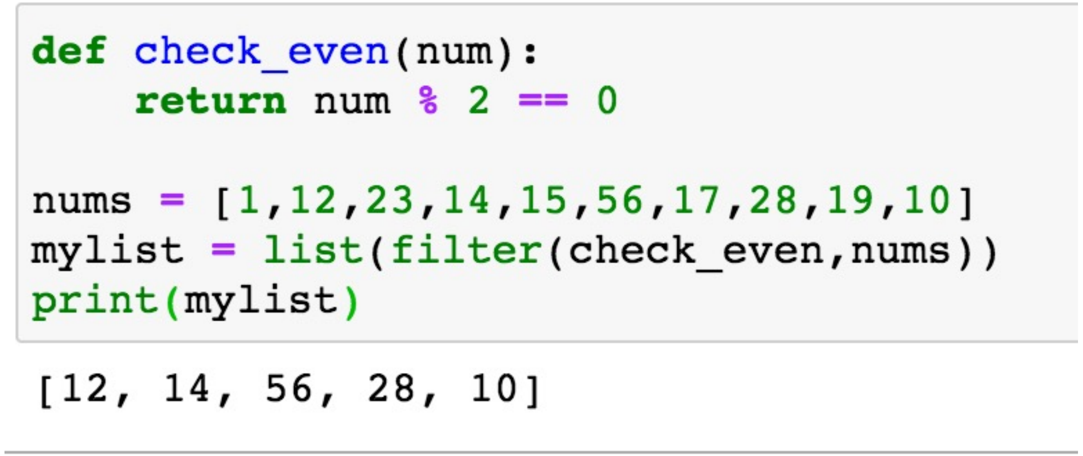


```
def square(num):
    return num**2

my_nums = [1,2,3,4,5]
print(map(square,my_nums))

maplist = list(map(square, my_nums))
print(maplist)
```

```
<map object at 0x106393f90>
[1, 4, 9, 16, 25]
```

list(map(lambda num: num ** 2, my_nums))

Lambda expression:



```
def check_even(num):
    return num % 2 == 0

nums = [1,12,23,14,15,56,17,28,19,10]
mylist = list(filter(check_even,nums))
print(mylist)
```

```
[12, 14, 56, 28, 10]
```

list(filter(lambda num: num%2==0, nums))

CODING OUR WAY TO THE FUTURE

THANKS FOR YOUR TIME

FANTAGEEKS
大福媽瘋程式