LECTURE 3: FUNCTIONS

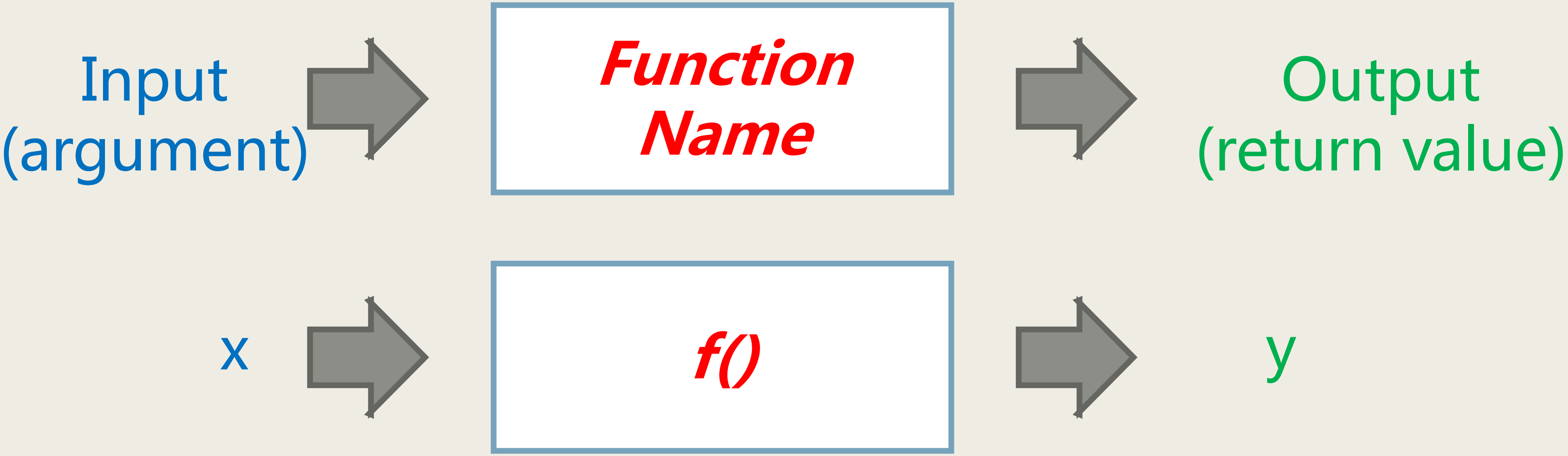
Hung-Yu Wei

What's function?

- Do you remember functions in your math class?

  sin(x), cos(x), tan(x)

$$y=f(x)$$



Input
(argument)

Function Name

Output
(return value)

x

f()

y

Function calls

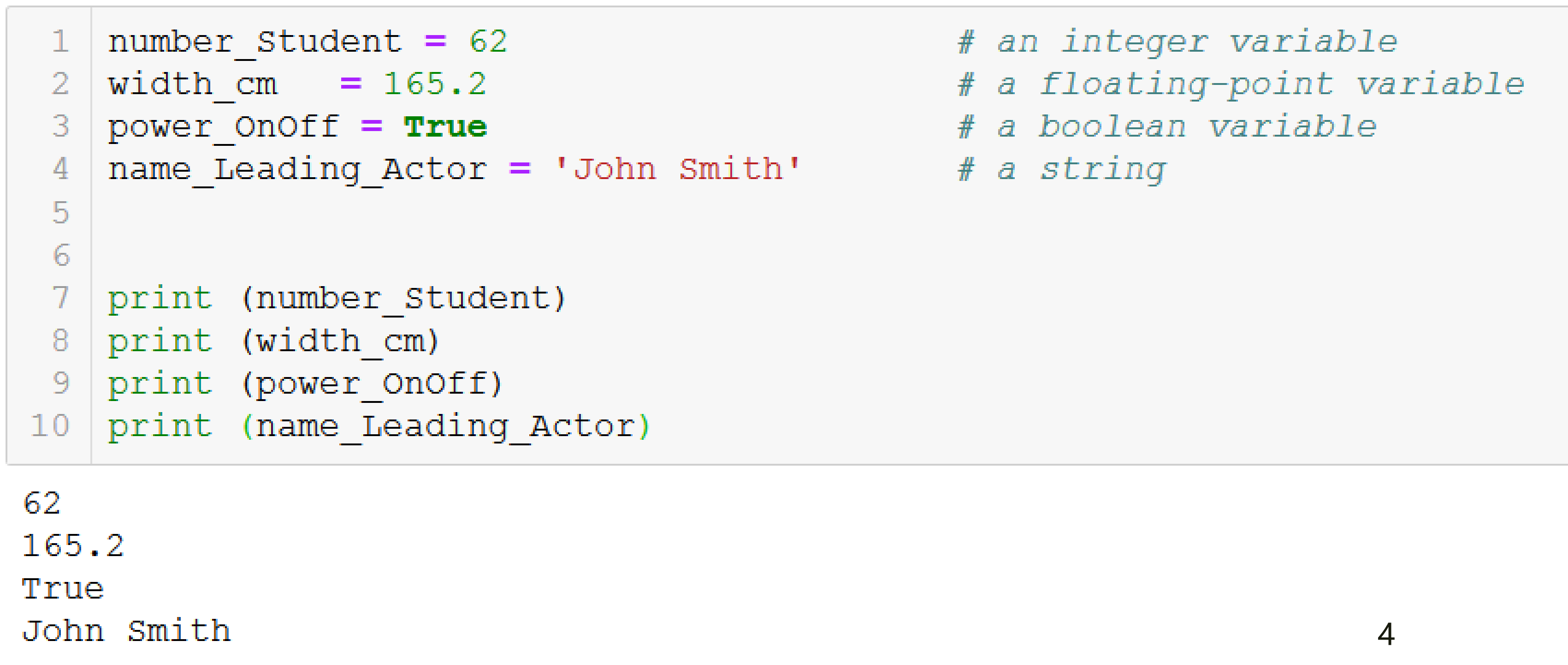
- Call (invoke) a function
  - *Execute this function*

- A function might include input and/or output
  - *Input: **argument***
  - *Output: **return value***

- A function could be
  - *Pre-defined in standard library*
  - *Defined by you*

Review: types of variables

- Number
  - *Integer*
  - *Floating point*
  - *Boolean (True, or False)*
- String

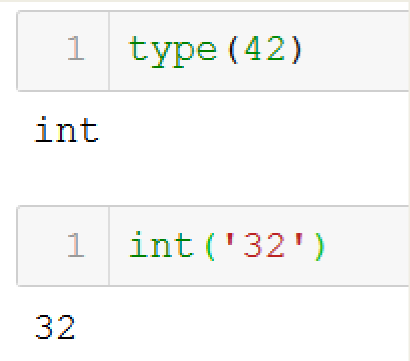


```
number_Student = 62                     # an integer variable
width_cm   = 165.2                      # a floating-point variable
power_OnOff = True                      # a boolean variable
name_Leading_Actor = 'John Smith'       # a string


print (number_Student)
print (width_cm)
print (power_OnOff)
print (name_Leading_Actor)
```

```
62
165.2
True
John Smith
```

Examples of function calls

- **type()**
  - *A function to **show the type** of input*
- **int()**
  - *A function to **convert** the input variable **into integer***
    - It will get rid of the fraction parts

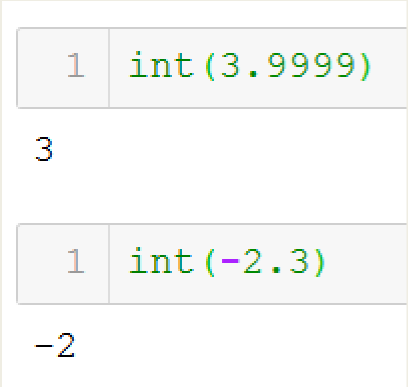


```
type(42)
```

```
int
```

```
int('32')
```

```
32
```

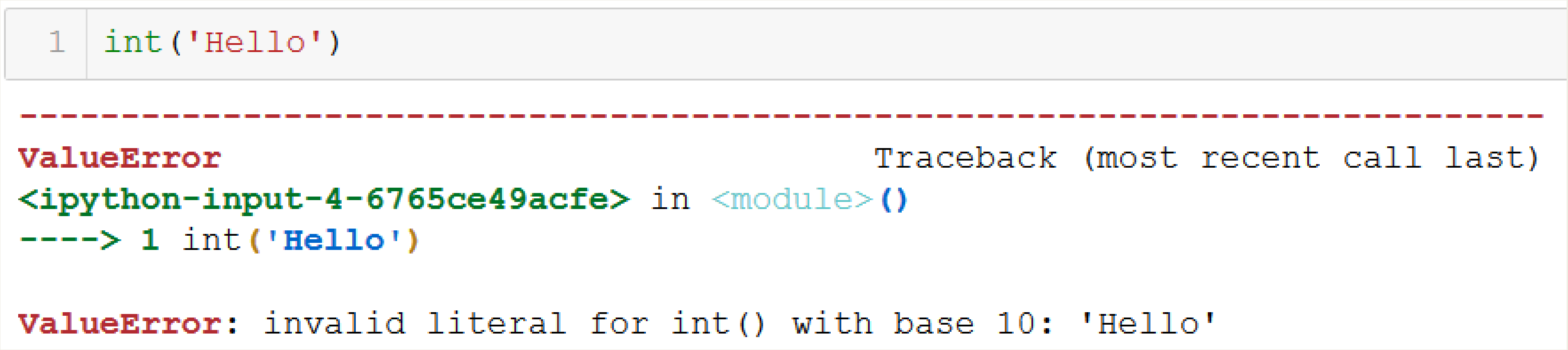


```
int(3.9999)
```

```
3
```

```
int(-2.3)
```

```
-2
```



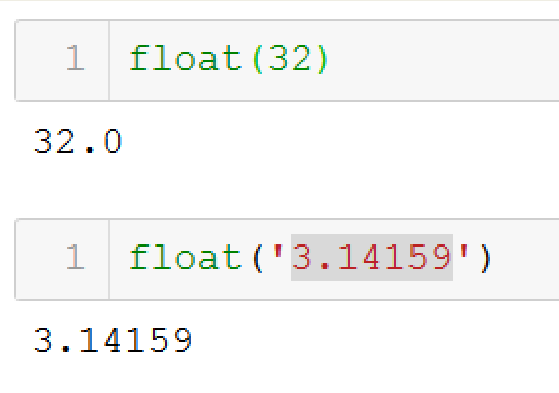
```
int('Hello')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-4-6765ce49acfe> in <module>()
----> 1 int('Hello')

ValueError: invalid literal for int() with base 10: 'Hello'
```

More examples of function calls

- **float()**
  - *A function to* ***convert*** *the input into* ***floating-point***

- **str()**
  - *A function to* ***convert*** *the input into* ***string***

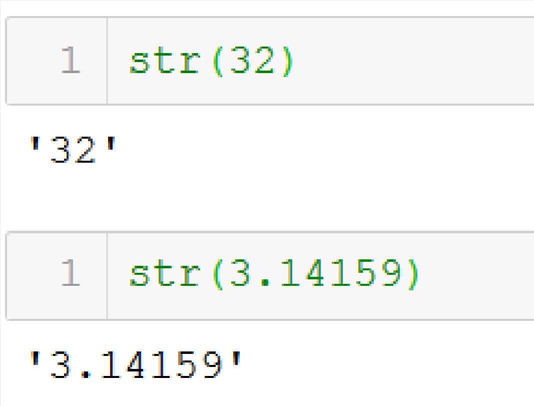


```
float(32)
```

```
32.0
```

```
float('3.14159')
```

```
3.14159
```



```
str(32)
```

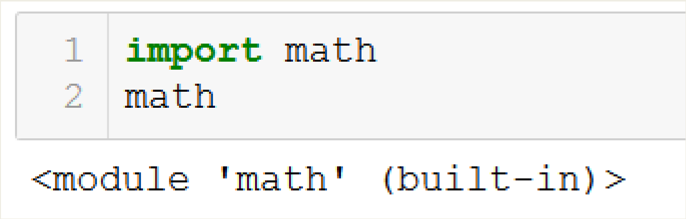
```
'32'
```

```
str(3.14159)
```

```
'3.14159'
```

Math module and functions

- **module**: a collection of functions
  - *a function library*
  - *Many programming languages have built-in standard libraries*
- **import** statement
  - *Import a library before using it*
- **dot** notation
  - *Call a function within a module*

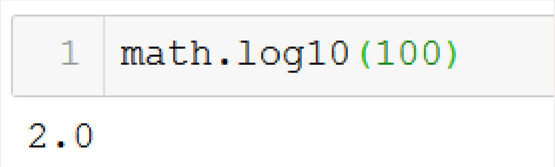


```
import math
math
```

```
<module 'math' (built-in)>
```

- Example
  - ***math module***
  - ***import** math before using any function in this module*
  - *log10 is a function defined in math module*
  - *To use the logarithm function, we use dot **.***

    math.log10

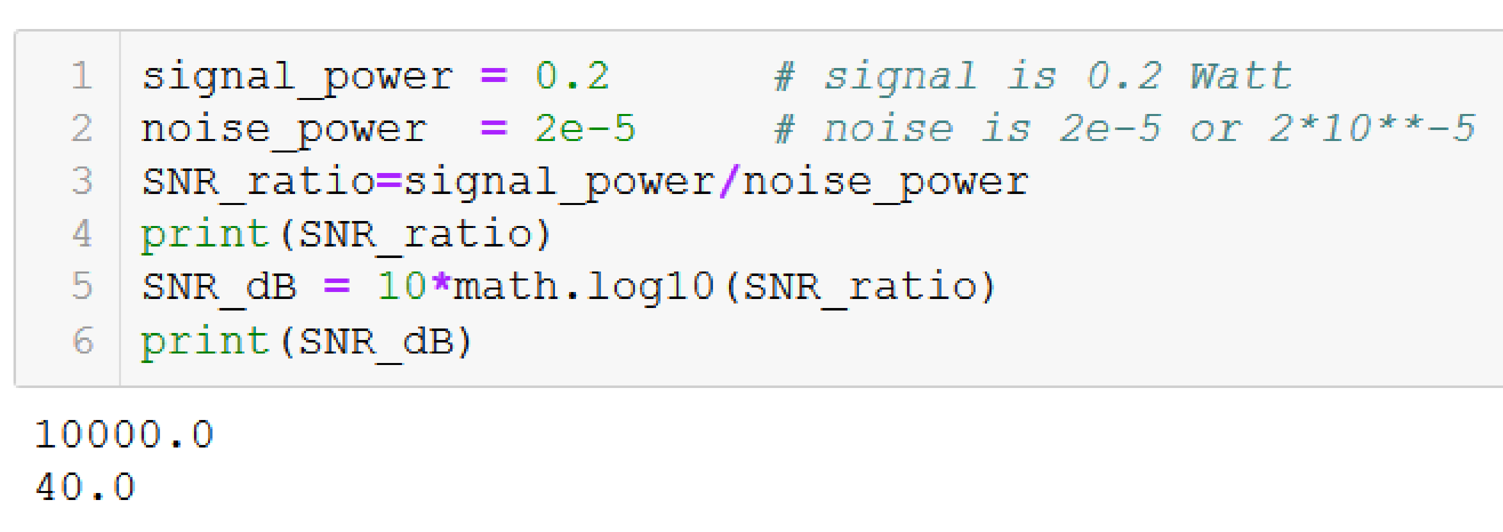


```
math.log10(100)
```

```
2.0
```

More about “math” functions

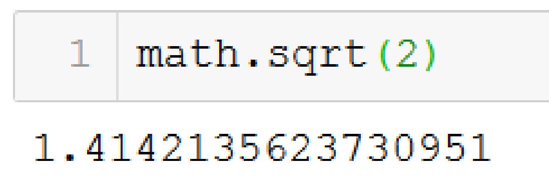
- Logarithm
  - *math.log10()*
  - *Example:*
    - calculating decibel
- Sine
  - *math.sin()*
- π
  - *math.pi*



```
signal_power = 0.2        # signal is 0.2 Watt
noise_power  = 2e-5       # noise is 2e-5 or 2*10**-5
SNR_ratio=signal_power/noise_power
print(SNR_ratio)
SNR_dB = 10*math.log10(SNR_ratio)
print(SNR_dB)
```

```
10000.0
40.0
```

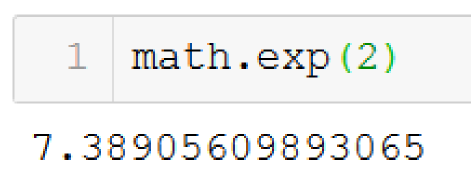
- Square root $\sqrt{x}$
  - *math.sqrt()*



```
math.sqrt(2)
```

```
1.4142135623730951
```

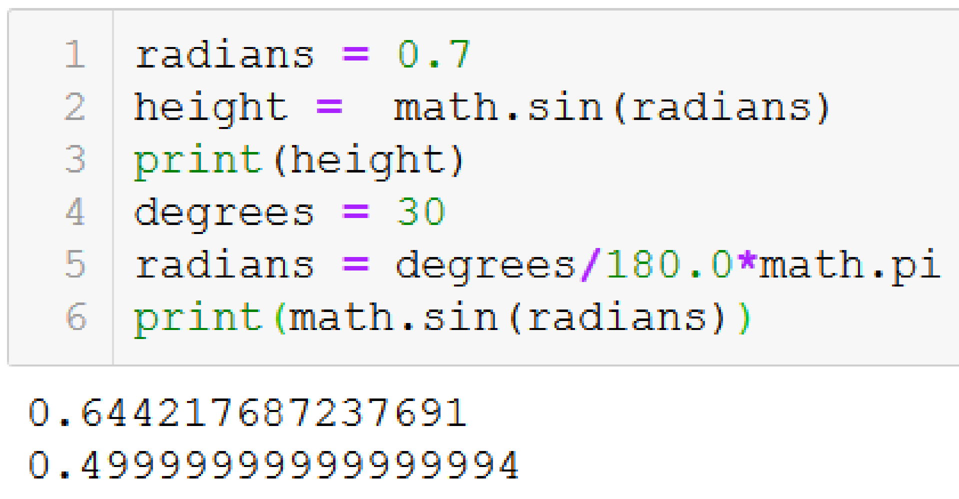
- Exponential e^x
  - *math.exp()*



```
math.exp(2)
```

```
7.38905609893065
```

- Remember to
  import math
- More math functions
  - https://docs.python.org/3/library/math.html

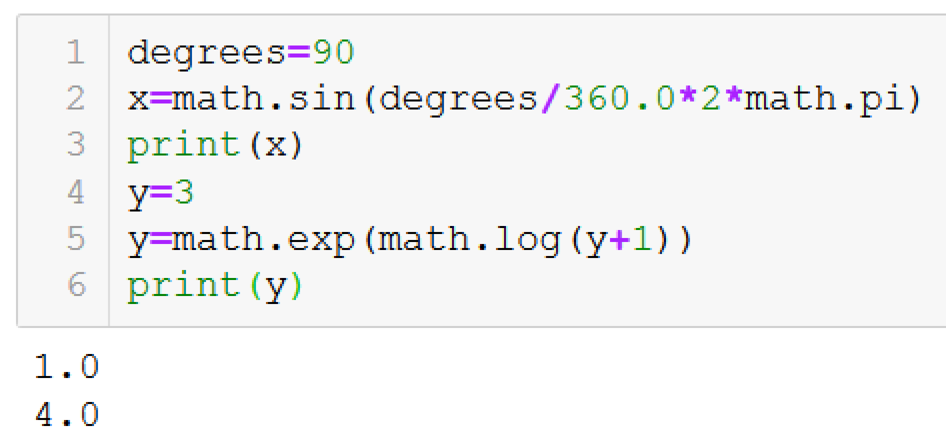


```
radians = 0.7
height =  math.sin(radians)
print(height)
degrees = 30
radians = degrees/180.0*math.pi
print(math.sin(radians))
```

```
0.644217687237691
0.49999999999999994
```

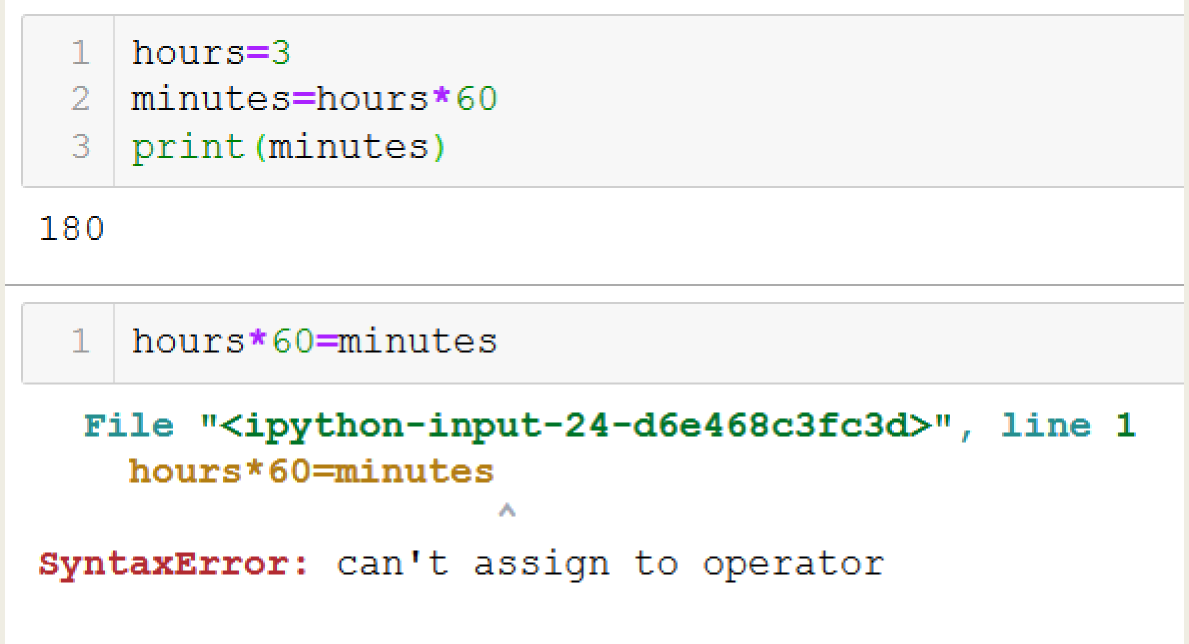
Composition

- Use (compose) multiple building blocks
  - $e^{\log(y+1)} = y + 1$



```
degrees=90
x=math.sin(degrees/360.0*2*math.pi)
print(x)
y=3
y=math.exp(math.log(y+1))
print(y)
```

```
1.0
4.0
```



```
hours=3
minutes=hours*60
print(minutes)
```

```
180
```

```
hours*60=minutes
```

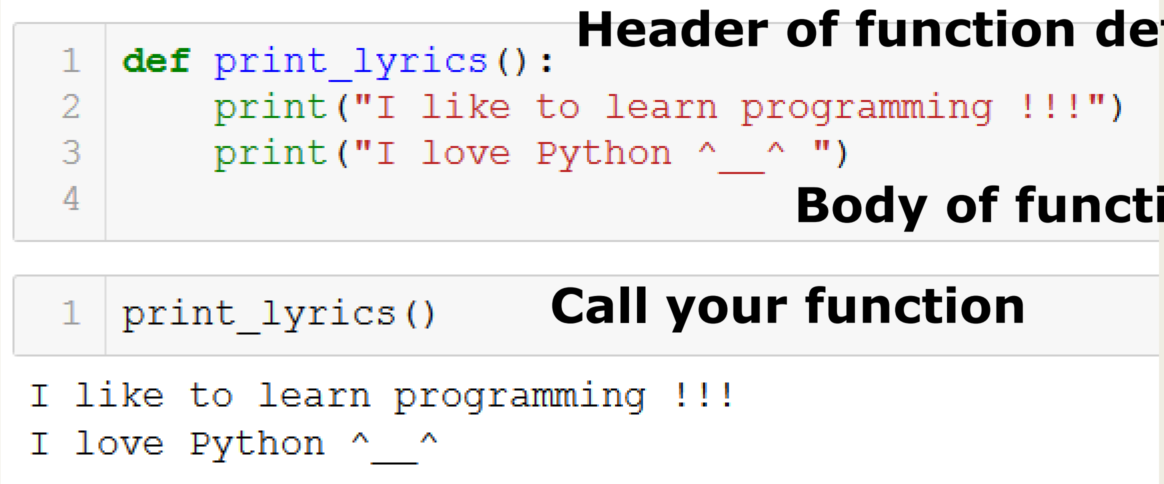
```
  File "<ipython-input-24-d6e468c3fc3d>", line 1
    hours*60=minutes
                    ^
SyntaxError: can't assign to operator
```

Create new functions

- I can use built-in functions. However, I also want to create my own functions.
  - *Flexibility in defining new components*
- Syntax: **define** a new function

  def function_name() **:**

  … *some operations in your function* …



```
def print_lyrics():
    print("I like to learn programming !!!")
    print("I love Python ^__^ ")

```

```
print_lyrics()
```

```
I like to learn programming !!!
I love Python ^__^
```

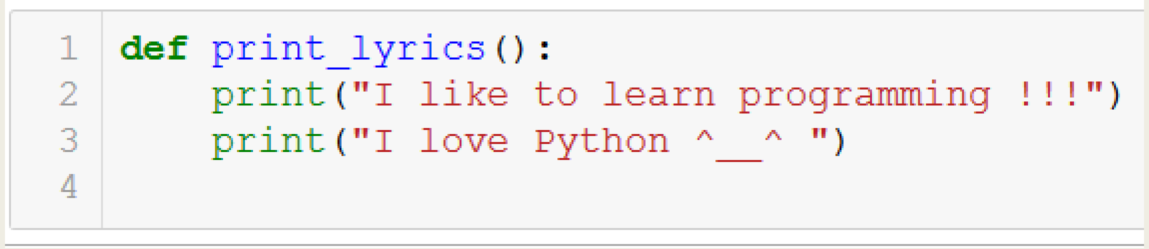
Header of function definition

Body of function definition

Call your function

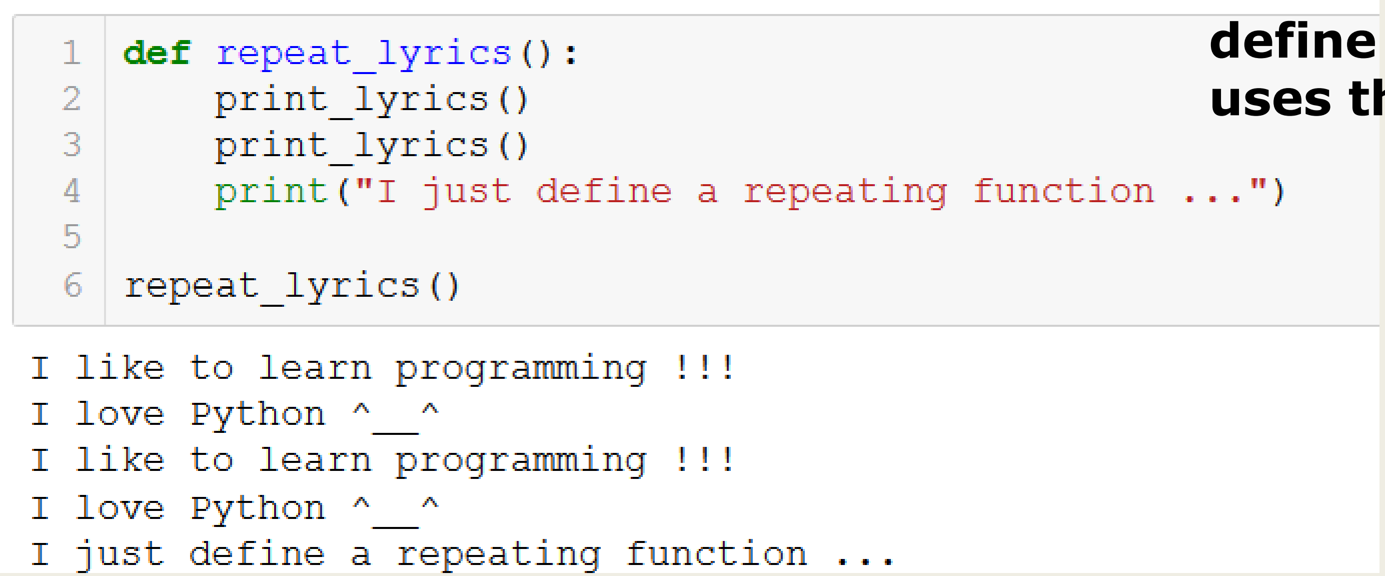
- Tips: remember ":" and "()"

A function that calls another function



```python
def print_lyrics():
    print("I like to learn programming !!!")
    print("I love Python ^__^ ")

```

define the 1st function



```python
def repeat_lyrics():
    print_lyrics()
    print_lyrics()
    print("I just define a repeating function ...")

repeat_lyrics()
```

```
I like to learn programming !!!
I love Python ^__^
I like to learn programming !!!
I love Python ^__^
I just define a repeating function ...
```

define the 2nd function that uses the 1st function

Argument and parameter

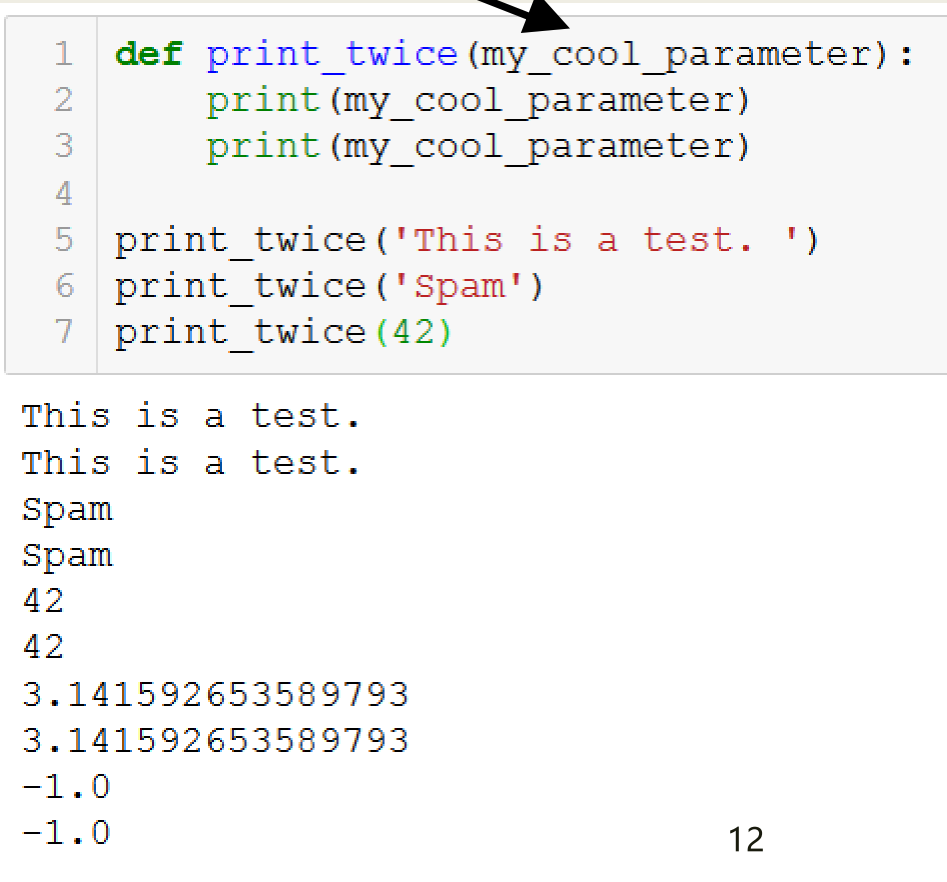
- **Parameter**: local variable to handle the input (e.g. my_cool_parameter)

  *def myFunc(**parameter_in_myFunc**)*

  ……………………

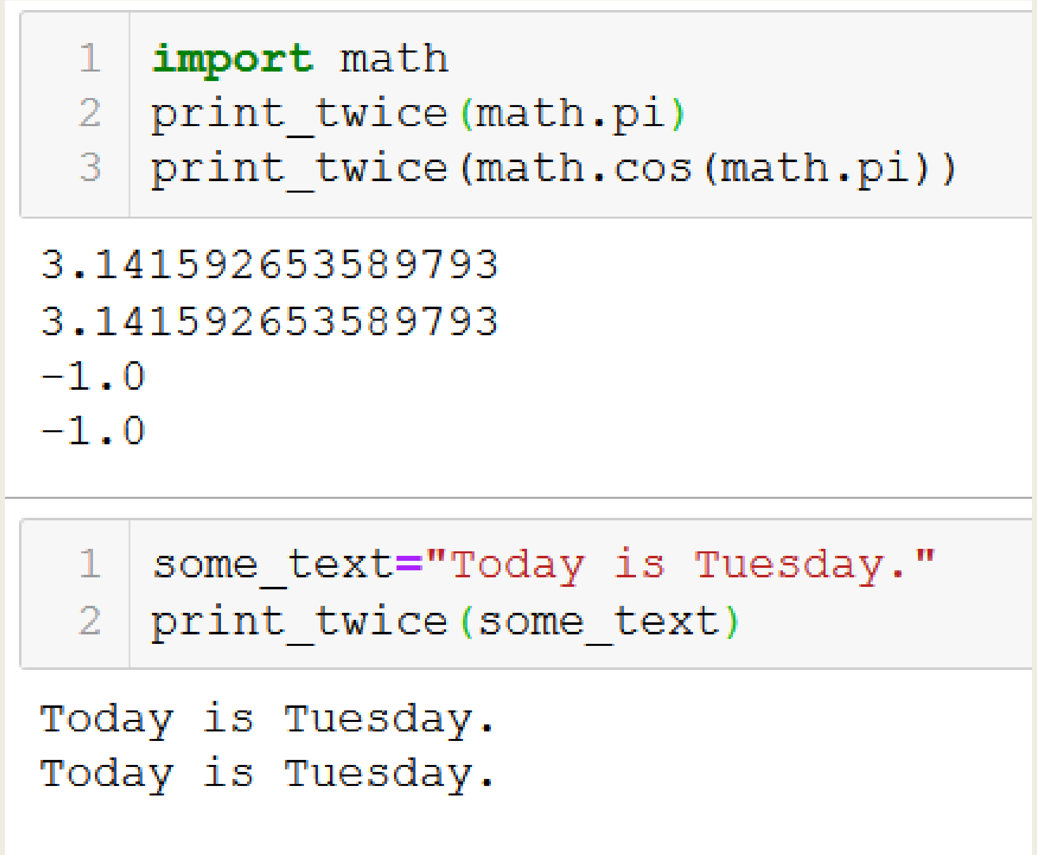
- **Argument**: input of a function

  *myFunc(**I_am_argument**)*



```python
def print_twice(my_cool_parameter):
    print(my_cool_parameter)
    print(my_cool_parameter)

print_twice('This is a test. ')
print_twice('Spam')
print_twice(42)
```

```
This is a test.
This is a test.
Spam
Spam
42
42
3.141592653589793
3.141592653589793
-1.0
-1.0
```



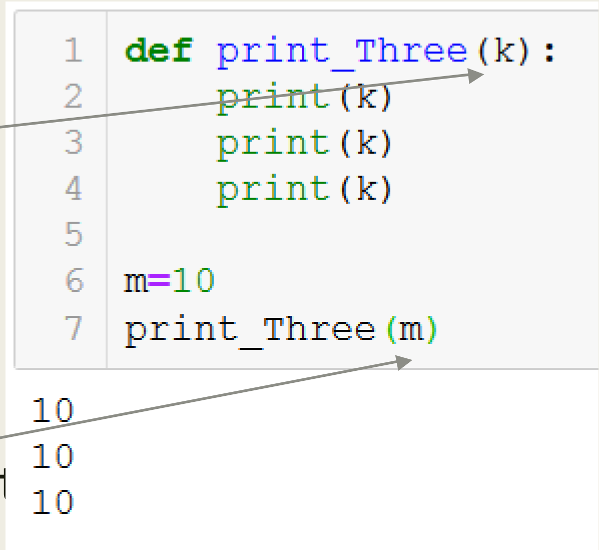
```python
import math
print_twice(math.pi)
print_twice(math.cos(math.pi))
```

```
3.141592653589793
3.141592653589793
-1.0
-1.0
```

```python
some_text="Today is Tuesday."
print_twice(some_text)
```

```
Today is Tuesday.
Today is Tuesday.
```

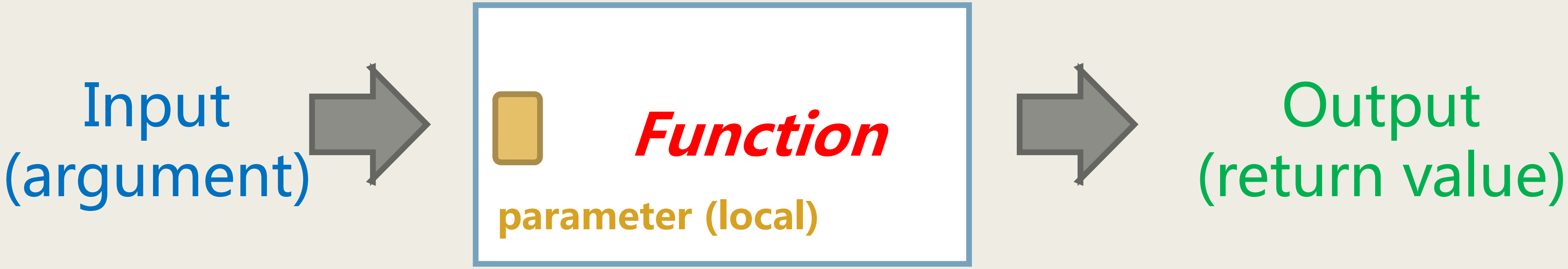
Clarification on Argument and Parameter



```
def print_Three(k):
    print(k)
    print(k)
    print(k)

m=10
print_Three(m)
```

```
10
10
10
```

- Parameter --- **k**
  - *This is **local***
  - *Only valid within function print_Three*
    - Local parameter will be destroyed at the end of funct
- Function Argument -- **m**



Input (argument)

Function

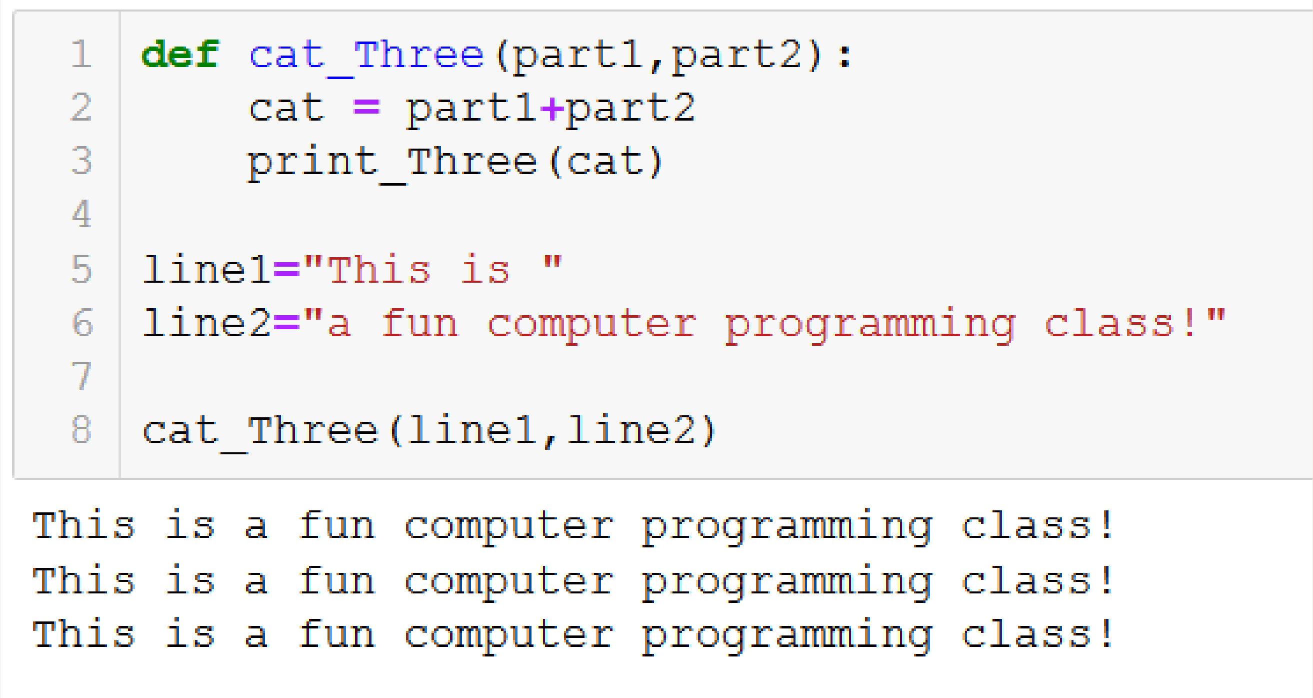
parameter (local)

Output (return value)

Local variable and parameter

- Local variable
  - *cat*

- Local parameter
  - *part1, part2*

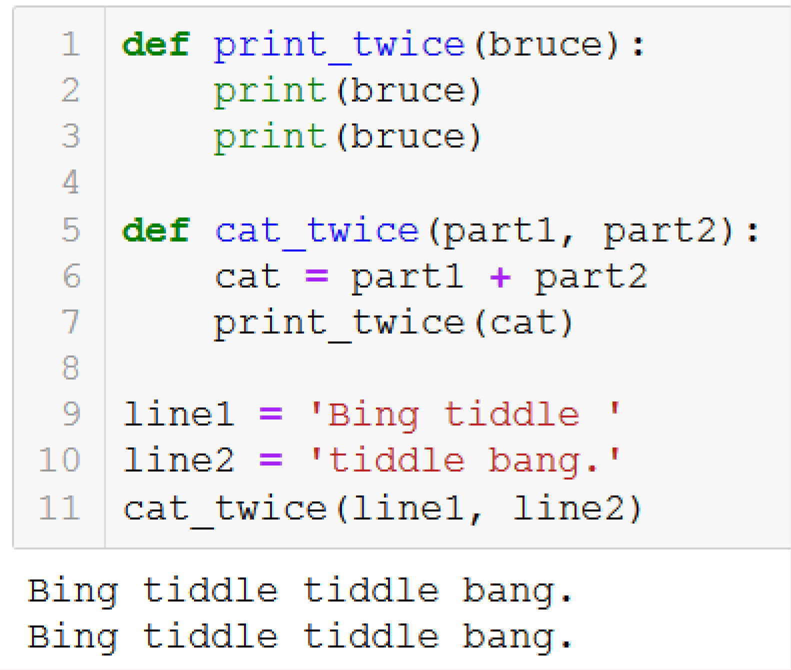


```python
def cat_Three(part1,part2):
    cat = part1+part2
    print_Three(cat)

line1="This is "
line2="a fun computer programming class!"

cat_Three(line1,line2)
```

```
This is a fun computer programming class!
This is a fun computer programming class!
This is a fun computer programming class!
```

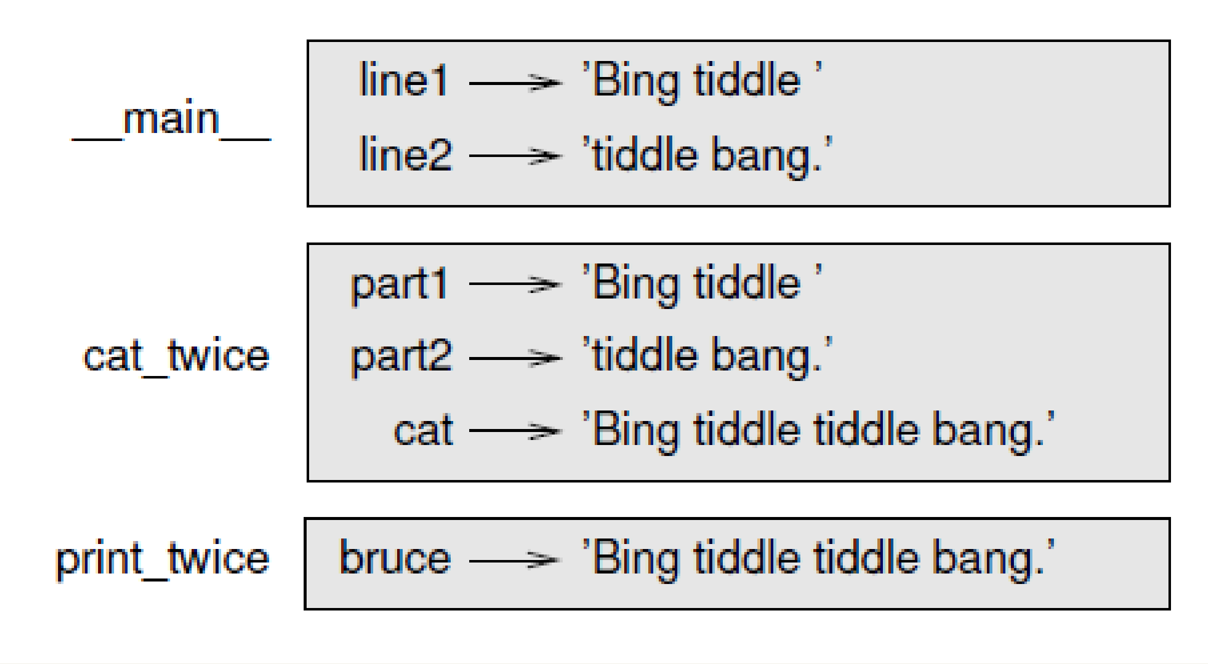
Stack Diagram

- Scope of variables



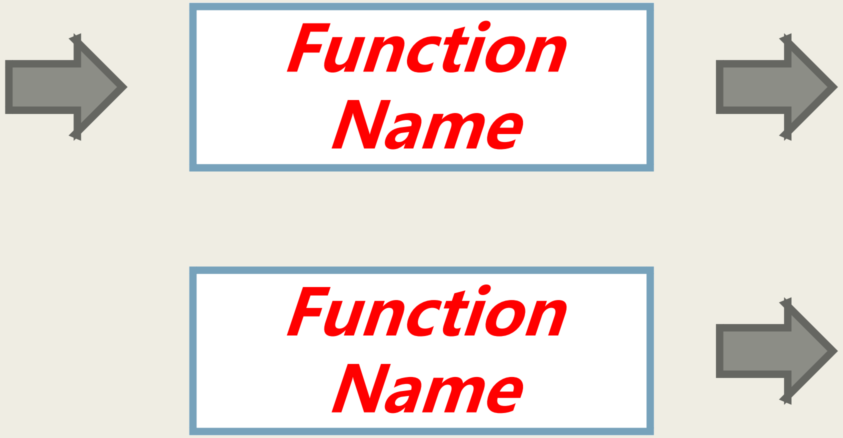
```
def print_twice(bruce):
    print(bruce)
    print(bruce)

def cat_twice(part1, part2):
    cat = part1 + part2
    print_twice(cat)

line1 = 'Bing tiddle '
line2 = 'tiddle bang.'
cat_twice(line1, line2)
```

```
Bing tiddle tiddle bang.
Bing tiddle tiddle bang.
```

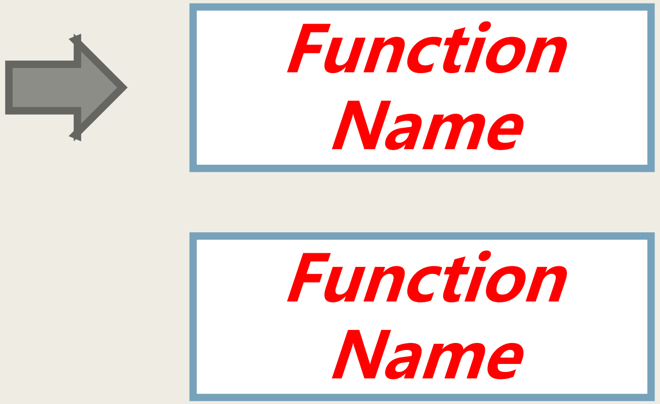


Functions With/Without Return

- Function with return
  - *Fruitful function*

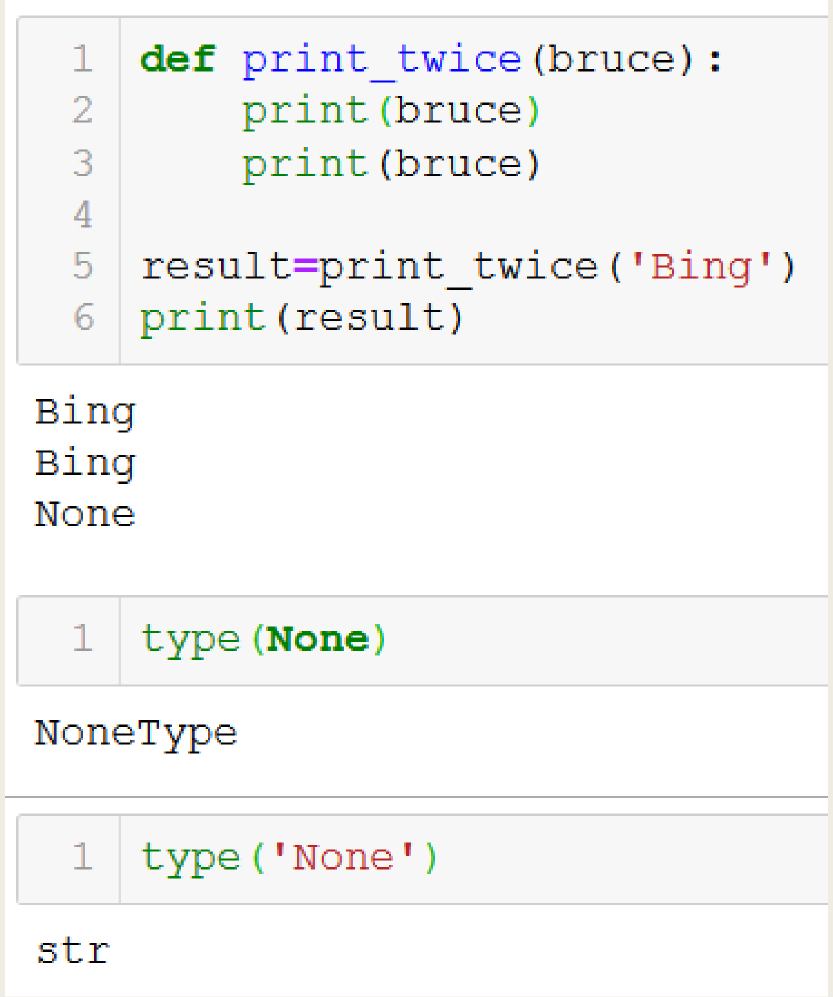


- Function without return
  - *Void function*



Void function and None

- None
  - *A special value for void function return*
  - *Different from string ‘None’*



```
def print_twice(bruce):
    print(bruce)
    print(bruce)

result=print_twice('Bing')
print(result)
```

```
Bing
Bing
None
```

```
type(None)
```

```
NoneType
```

```
type('None')
```

```
str
```

Benefits of using function

- Clarity and readability
  - *Creating a new function gives you an opportunity to name a group of statements, which makes your program **easier to read and debug**.*
- Eliminate duplication
  - *Functions can make a program smaller by **eliminating repetitive code**. Later, if you make a change, you only have to make it in one place.*
- Divide into smaller building blocks (make sure each small block is OK)
  - *Dividing a long program into functions allows you to **debug the parts one at a time** and then assemble them into a working whole.*
- Reusability
  - *Well-designed functions are often useful for many programs. Once you write and debug one, you can reuse it.*

Summary

- Function
  - *Use functions in standard library*
    - Import
  - *Define your function*
    - def

Reading

- Chapter 3 in textbook “Think Python”