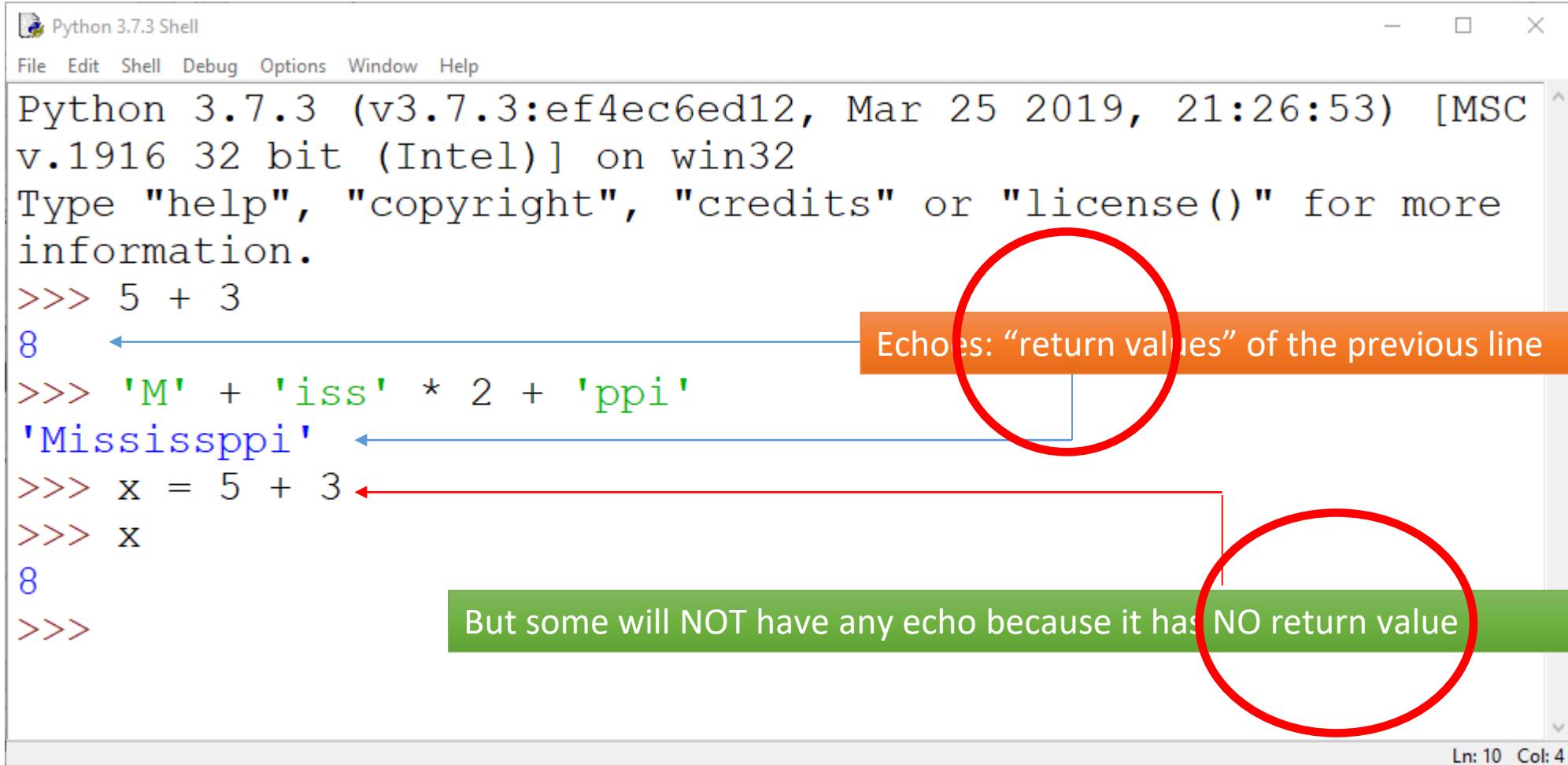


“Return” of Functions

(The previous drawing examples do not have any return values)

Python Shell(Console)



The screenshot shows a Python 3.7.3 Shell window. The console output is as follows:

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>> 5 + 3
8
>>> 'M' + 'iss' * 2 + 'ppi'
'Mississippi'
>>> x = 5 + 3
>>> x
8
>>>
```

A red circle highlights the output of the first command ('8') with the annotation: "Echoes: ‘return values’ of the previous line". A red circle highlights the output of the assignment statement ('Mississippi') with the annotation: "But some will NOT have any echo because it has NO return value".

- However, this should NOT be the main area we work in

A Function may or may not return a value

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

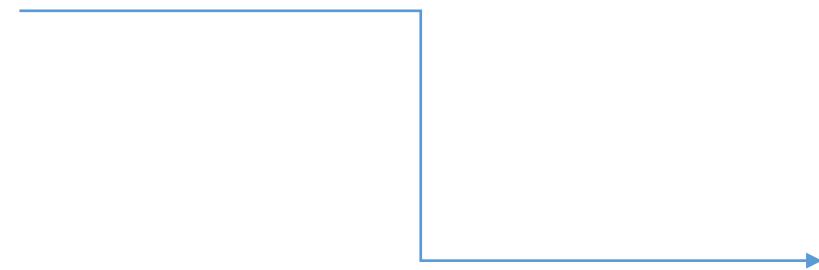
This function returns a value

```
def say3Times(s):  
    print(s)  
    print(s)  
    print(s)
```

This function does NOT return a value
However, in Python, it “returns” a value of
“None”

Python Echo in the Shell

- Wait a minute? I thought you say the second function does not return any value?



```
>>> square(3)
9
>>> say3Times("Hello ")
Hello
Hello
Hello
```

- The 9 is a return value from the function square and Python shell echo it
- The 3 “Hello” are NOT return value but from the “print()” function

Function that “doesn’t” return any value

- Note that the function print also only returns a “None”

```
>>> print(square(3))  
9  
>>> print(say3Times("Hello "))  
Hello  
Hello  
Hello  
None  
>>> print(print())  
None
```

Return Values

Vs “print()”

Print vs Return

```
def foo_print3():
    print(3)
    print(3)

def foo_return3():
    return 3
    return 3
```

```
>>> foo_print3()
3
3
>>> foo_return3()
3
>>>
>>>
```

- “return” will end the function immediately

Print vs Return

```
def foo_print3():
    print(3)
```

```
def foo_return3():
    return 3
```

```
>>> foo_print3()
3
>>> foo_return3()
3
>>>
```

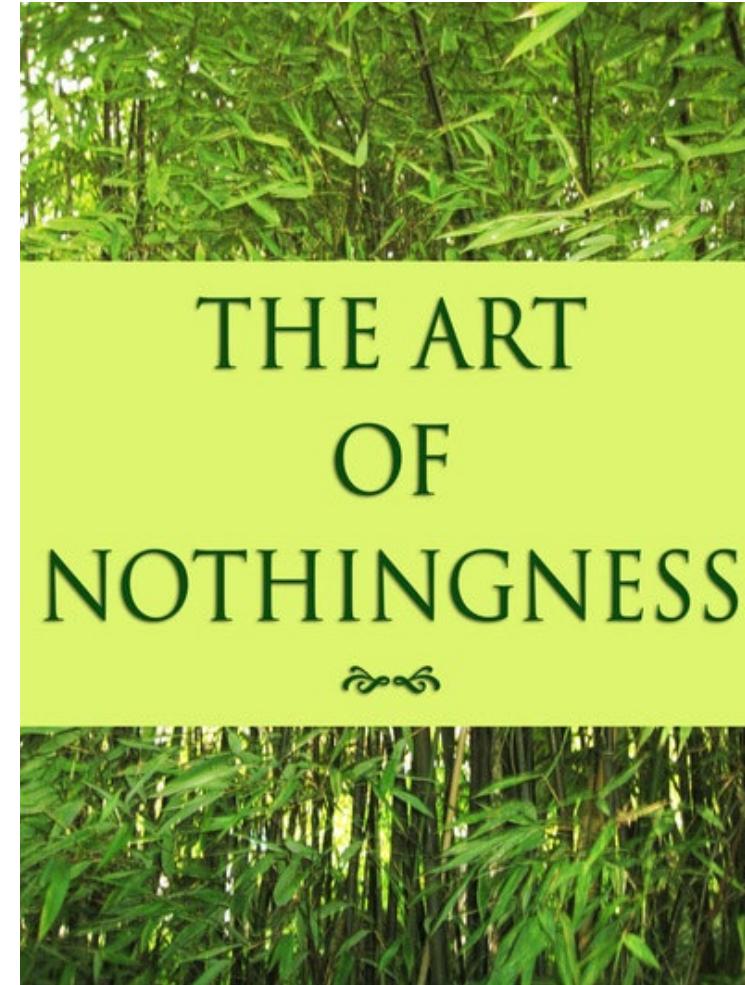


Wait...

```
>>> x = foo_print3()  
3  
>>> y = foo_return3()  
>>> |
```

Nothing?

```
>>> type(x)  
<class 'NoneType'>  
>>> type(y)  
<class 'int'>  
>>> |
```



Print vs Return

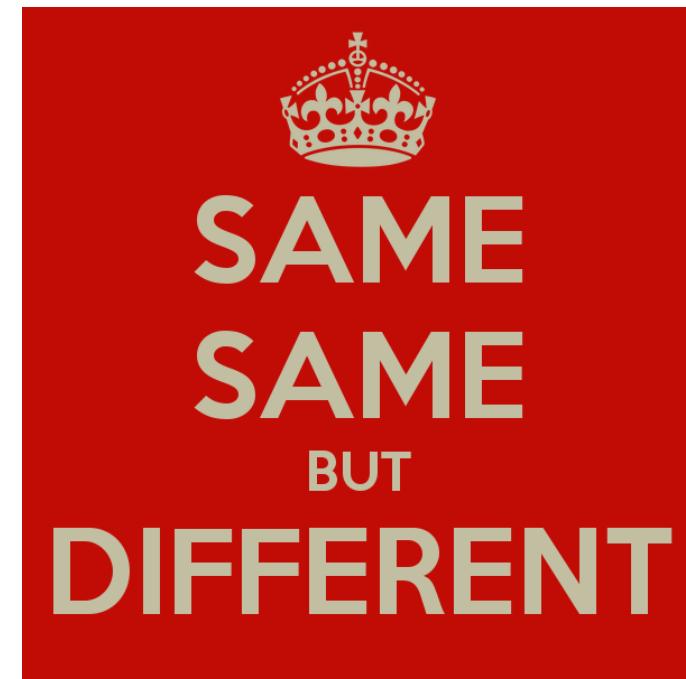
```
def foo_print3():
    print(3)
```

```
def foo_return3():
    return 3
```

```
>>> foo_print3()
3
>>> foo_return3()
3
>>>
```

By the print function

IDLE's echo



Print vs Return

```
def foo_print3():
    print(3)
```

```
def foo_return3():
    return 3
```

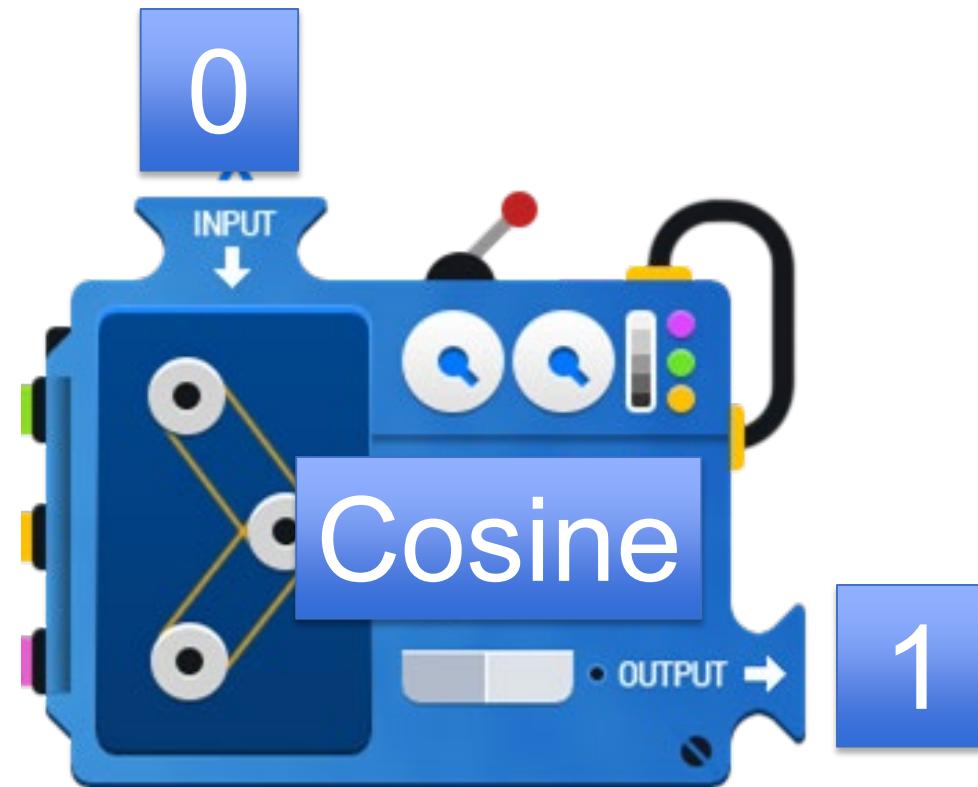
- `foo_print3()` does not “return” a value

```
>>> x = foo_print3()
3
>>> y = foo_return3()
>>> |
```

IDLE echoes “nothing”

Function

- “Cosine” is a function
 - Input 0
 - Output/**return** 1
 - $x = \cos(0)$
 - return
 - That's why $x = 1$



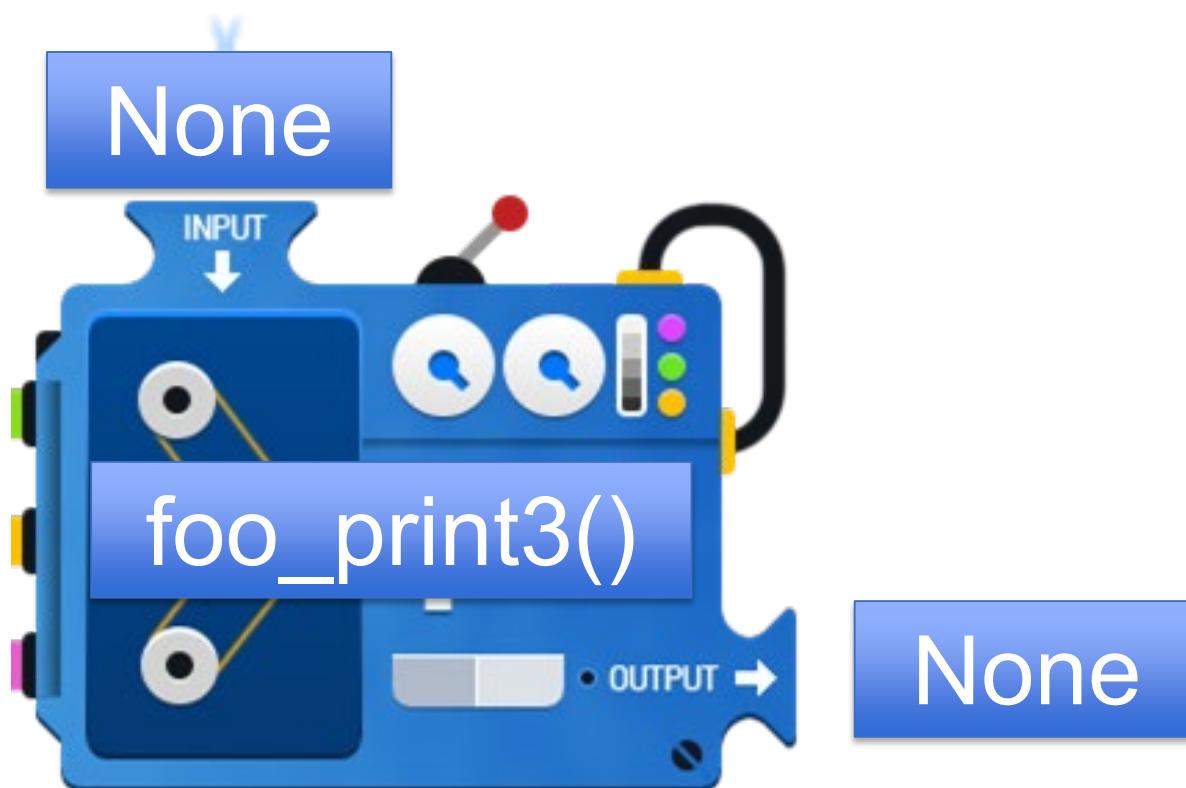
Function

- “`foo_print3()`” is a function
 - Input nothing
 - No output

```
y = foo_print3()  
      ↘  
      return "None"
```

In general, we called all these “functions”

But for a function that “returns” nothing. Sometime we call it a “procedure”

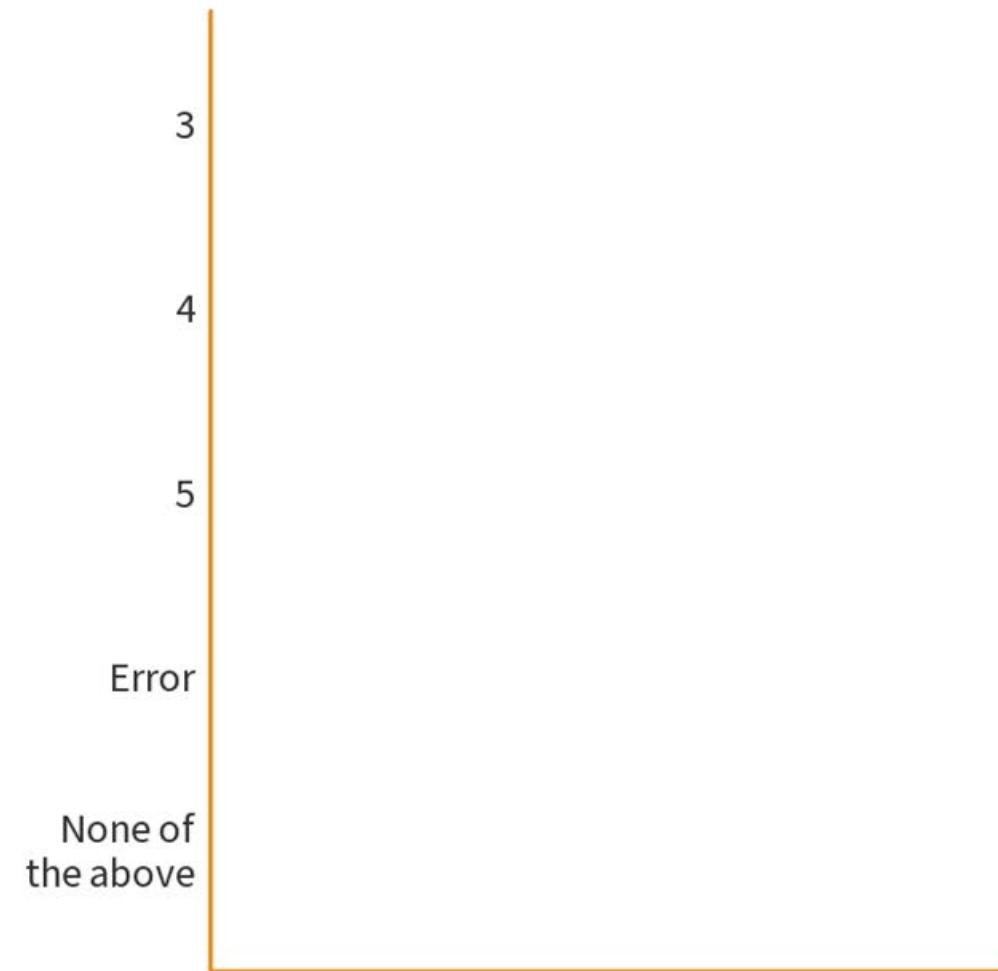


Return Values

- All functions returns “something”
- `foo_return3()` return the integer 3
- `foo_print3()`
 - Do not have any return statement
 - So it returns “None”

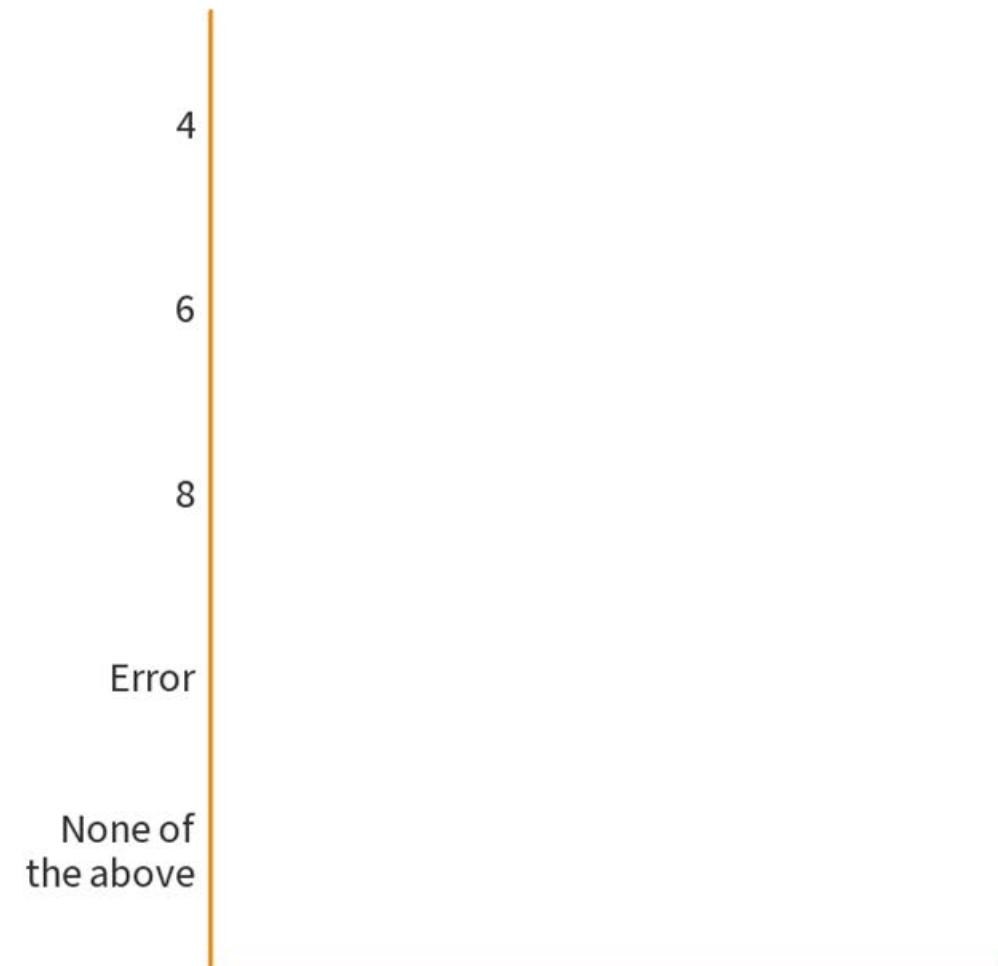
Question: Can we assume that a function always return something of the same TYPE?

```
def strange(a,b):  
    c = a * b // b  
    return c  
    return c+1  
    return c+2  
  
>>> strange(3,100)
```



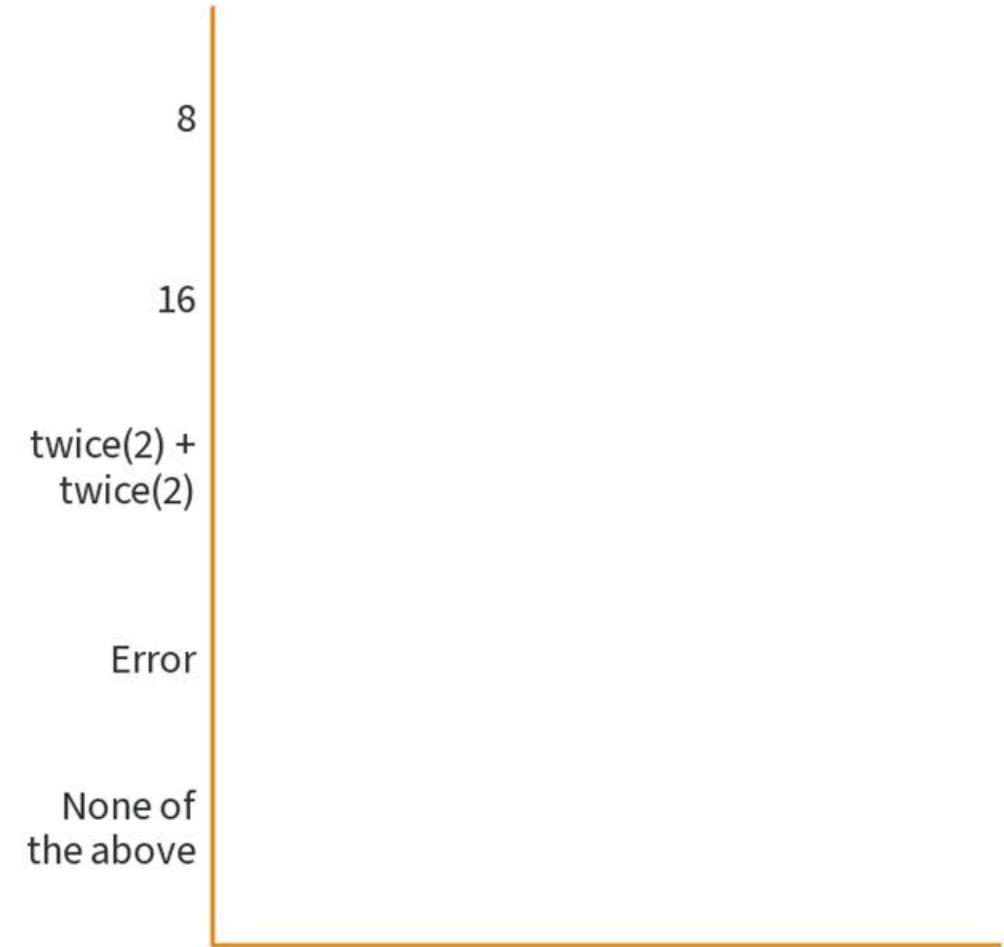
```
def g(a,b):  
    (a + b) * 2
```

```
def f(x,y):  
    res = g(x,y)  
    return res + res
```



*What is the output from the call
twice(twice(twice(2)))?*

```
def twice(x):  
    return x + x
```



Get's get some real coding!



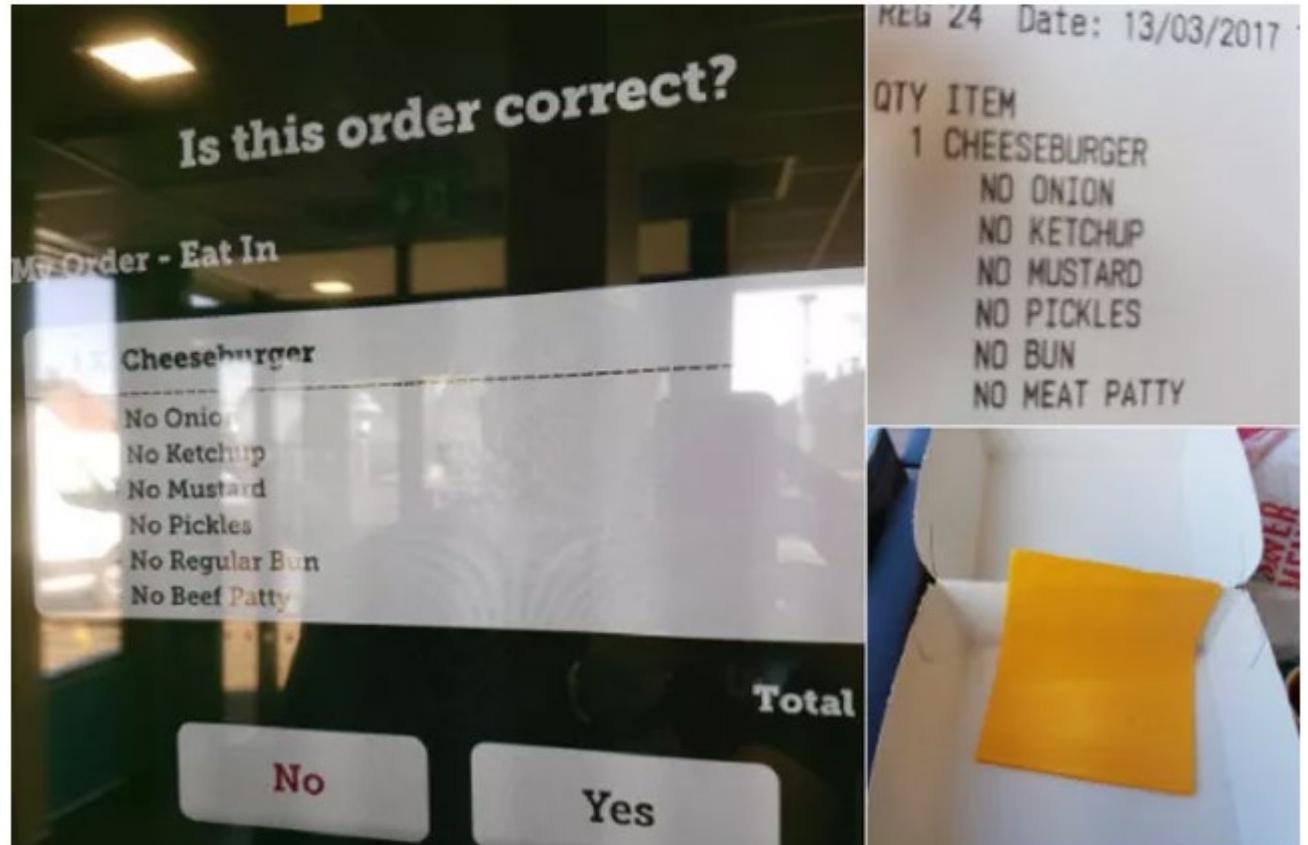
Some Fast food has Food Customization

- Meaning, you can micro-manage what will be or will not be in your burgers

This guy went to McDonald's and ended up just ordering a slice of cheese



Miranda Larbi Wednesday 15 Mar 2017 11:36 am



Burger Customization

- ‘B’ stands for a piece of bun
- ‘C’ stands for cheese
- ‘P’ stands for patty
- ‘V’ stands for veggies
- ‘O’ stands for onions
- ‘M’ stands for mushroom
- (maybe you can come up with more?)

Encoding as a String

- A simple burger
 - ‘BVPB’
- A double cheese burger
 - ‘BVCPCPB’
- A Big Mac?



Write a function burgerPrice() to calculate the **price**

- ‘B’ stands for a piece of bun \$0.5
- ‘C’ stands for cheese \$0.8
- ‘P’ stands for patty \$1.5
- ‘V’ stands for veggies \$0.7
- ‘O’ stands for onions \$0.4
- ‘M’ stands for mushroom \$0.9
- E.g.

```
>>> burgerPrice('BVPB')
```

Discuss With your Neighbor
on how to start/do it

(5 min)

Write a function burgerPrice() to calculate the **price**

- ‘B’ stands for a piece of bun \$0.5
- ‘C’ stands for cheese \$0.8
- ‘P’ stands for patty \$1.5
- ‘V’ stands for veggies \$0.7
- ‘O’ stands for onions \$0.4
- ‘M’ stands for mushroom \$0.9
- E.g.

```
>>> burgerPrice('BVPB')
```

How will you do that in real life?

- You receive a string into your function
- Go through each character of the string one by one
- Accumulate the price for that character
- Output the final price

Which line(s) is the repetition if there are many characters in the

- E.g.

```
>>> burgerPrice('BVPB')
```

3.2

How will you do that in real life?

- You receive a string into your function
- Go through each character of the string one by one
 - Accumulate the price for that character
- Output the final price

Which line(s) is the repetition if there are many characters in the

- E.g.
- ```
>>> burgerPrice('BVPB')
```

3.2

# How will you do that in real life?

- You receive a string into your function
- Set the “**final price**” to be zero
- Go through each character of the string one by one
  - Accumulate the price for that character to the “**final price**”
- Output the **final price**
- E.g.  
`>>> burgerPrice( 'BVPB' )`

Then you need to start with 0

Whenever you want to accumulate some kind of sum or produce, you need a variable to store it

# How do I go through each character of the sting?

- You receive a string into your function
- Set the “**final price**” to be zero
- Go through each character of the string one by one
  - Accumulate the price for that character to the “**final price**”
- Output the **final price**

- E.g.

```
>>> burgerPrice('BVPB')
```

3.2

Now it's a good time to start  
your IDLE and code together!



# How do I go through each character of the string?

E.g. Just print out the letters one-by-one

- You receive a string into your function
- Set the “final price” to be zero
- Go through each character of the string one by one
  - Accumulate the price for that character to the “final price”
- Output the final price

```
def burgerPrice(burger):
 length = len(burger)
 for i in range(length):
 print(burger[i])
```

```
burgerPrice('BVPB')
```

Output:

```
>>> burgerPrice('BPB')
B
P
B
```

Note that this is NOT the final code.

However, we usually write some immediate code to make sure what is right.  
E.g. This code make sure that “burger[i]” will give you each character in the loop

# How do I find the price of each ingredient?

- You receive a string into your function
- Set the “**final price**” to be zero
- Go through each character of the string one by one
  - Accumulate the price for that character to the “**final price**”
- Output the **final price**

Output:

```
burgerPrice('BVPB')
0.5
0.7
1.5
0.5
>>>
```

# How do I find the price of each ingredient?

```
def burgerPrice(burger):
 length = len(burger)
 for i in range(length):
 if burger[i] == 'B':
 print(0.5)
 elif burger[i] == 'C':
 print(0.8)
 elif burger[i] == 'P':
 print(1.5)
 elif burger[i] == 'V':
 print(0.7)
```

```
burgerPrice('BVPB')
```

- Output:

```
0.5
0.7
1.5
0.5
>>>
```

- How to sum them?

# “Finally”

```
def burgerPrice(burger):
 price = 0
 length = len(burger)
 for i in range(length):
 if burger[i] == 'B':
 price = price + (0.5)
 elif burger[i] == 'C':
 price = price + (0.8)
 elif burger[i] == 'P':
 price = price + (1.5)
 elif burger[i] == 'V':
 price = price + (0.7)
 return price
```

```
burgerPrice('BVPB')
```

- Wait? Nothing happened if I run this code?

```
def burgerPrice(burger):
 price = 0
 length = len(burger)
 for i in range(length):
 if burger[i] == 'B':
 price = price + (0.5)
 elif burger[i] == 'C':
 price = price + (0.8)
 elif burger[i] == 'P':
 price = price + (1.5)
 elif burger[i] == 'V':
 price = price + (0.7)
 return price
```

```
print(burgerPrice('BVPB'))
```

- If you run a .py file, there will be NO Python Echo

# Are we done?

- Always give it a thought on
  - Can we do it another way?
  - Or, is there any other better way?
- In lecture we learnt:

```
for <var> in <sequence>:
 <body>
```
- **sequence**
  - a sequence of values
- **var**
  - variable that take each value in the sequence
- **body**
  - statement(s) that will be evaluated for each value in the sequence

Other than “range”, a string is a “**sequence**”!!!

# “for i in <sequence>:”

- Originally
- The variable i is the index of the string
  - So you need to get the character by burger[i]

```
def burgerPrice(burger):
 length = len(burger)
 for i in range(length):
 print(burger[i])
```



```
burgerPrice('BVPB')
```

- However, Python can **iterate** through a sequence by giving each **element** in the sequence directly in a for loop
  - The variable c is a character in burger

```
def burgerPrice(burger):
 for c in burger:
 print(c)
```

```
burgerPrice('BVPB')
```

# Finally

- New version

```
def burgerPrice(burger):
 price = 0
 for char in burger:
 if char == 'B':
 price = price + (0.5)
 elif char == 'C':
 price = price + (0.8)
 elif char == 'P':
 price = price + (1.5)
 elif char == 'V':
 price = price + (0.7)
 return price

print(burgerPrice('BVPB'))
```

- Compare to the old version

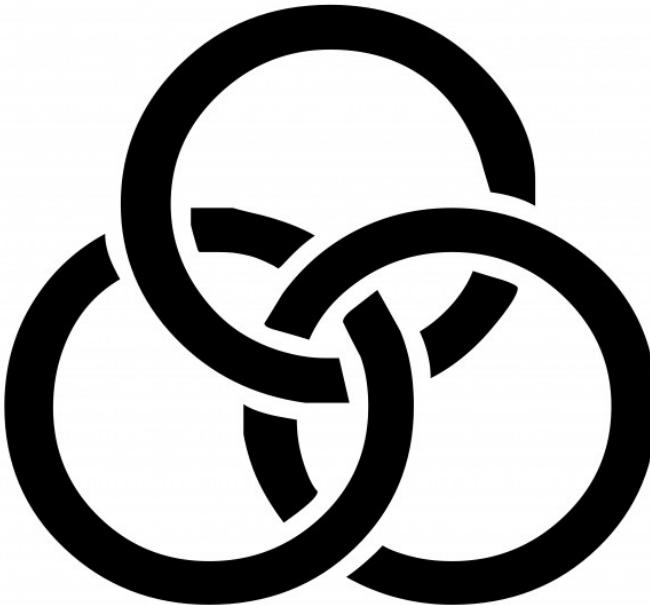
```
def burgerPrice(burger):
 price = 0
 length = len(burger)
 for i in range(length):
 if burger[i] == 'B':
 price = price + (0.5)
 elif burger[i] == 'C':
 price = price + (0.8)
 elif burger[i] == 'P':
 price = price + (1.5)
 elif burger[i] == 'V':
 price = price + (0.7)
 return price

print(burgerPrice('BVPB'))
```

# Learning Points

- Not only about how to get the final code but..
- Plan and write your code in English first
- You may need to write some intermediate code for a “semi finished product” to test out your idea
- After you finally get your code working, you should think about how to improve it
  - Not only for that single shot, you are improving your coding skill for your future coding

# Three Types of Loops



# Three Types of Loops

- For A and C, it means you know the number N when your loop starts
  - A. Must run exactly N times (definite)
  - B. Run any number of times (indefinite)
  - C. Run at most N times (definite loop that may break)
    - Check all True (or check all False)
    - Find any True (or False)

# Which Type is it?

- Iteration version of computing the factorial of N
  - A. Must run exactly N times (definite)
  - B. Run any number of times (indefinite)
  - C. Run at most N times (definite loop that may break)
    - Check all True (or check all False)
    - Find any True (or False)

# Which Type is it?

- Given a string, e.g. ‘abcdef’, compute its length
    - The function `len()`
    - First, think of how to do it without using the function `len()`
- A. Must run exactly N times (definite)
- B. Run any number of times (indefinite)
- C. Run at most N times (definite loop that may break)
  - Check all True (or check all False)
  - Find any True (or False)

# Which Type is it?

- Check if a string contains any vowel, e.g. the word ‘sky’ does not have any vowel
  - A. Must run exactly N times (definite)
  - B. Run any number of times (indefinite)
  - C. Run at most N times (definite loop that may break)
    - Check all True (or check all False)
    - Find any True (or False)

# Which Type is it?

- Check if a number  $N$  is prime
  - A. Must run exactly  $N$  times (definite)
  - B. Run any number of times (indefinite)
  - C. Run at most  $N$  times (definite loop that may break)
    - Check all True (or check all False)
    - Find any True (or False)

# Think of an Example of Each Type?

- A. Must run exactly N times (definite)
- B. Run any number of times (indefinite)
- C. Run at most N times (definite loop that may break)
  - Check all True (or check all False)
  - Find any True (or False)

# Comments in Python

# Comments in Python

- Usually denoted by # at the start of a line
- Can also be done between pairs of triple quotes

```
#Example of single line comment
```

```
'''
```

```
Example of triple quotes comment
```

```
Wow I can do multiple lines
```

```
'''
```

# Comments in Python

- Good habit to have comments in your code
  - Remind yourself what the code is for
  - Help others understand your code
- 
- Remember to make sure you mark out your comments properly. Otherwise, you might get an error when trying to run your program.