Week 2

Functions, Selection and Repetition

Functions

Let’s Write Our Own Function!

Function name

Define
(keyword)

Input
(Argument)

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

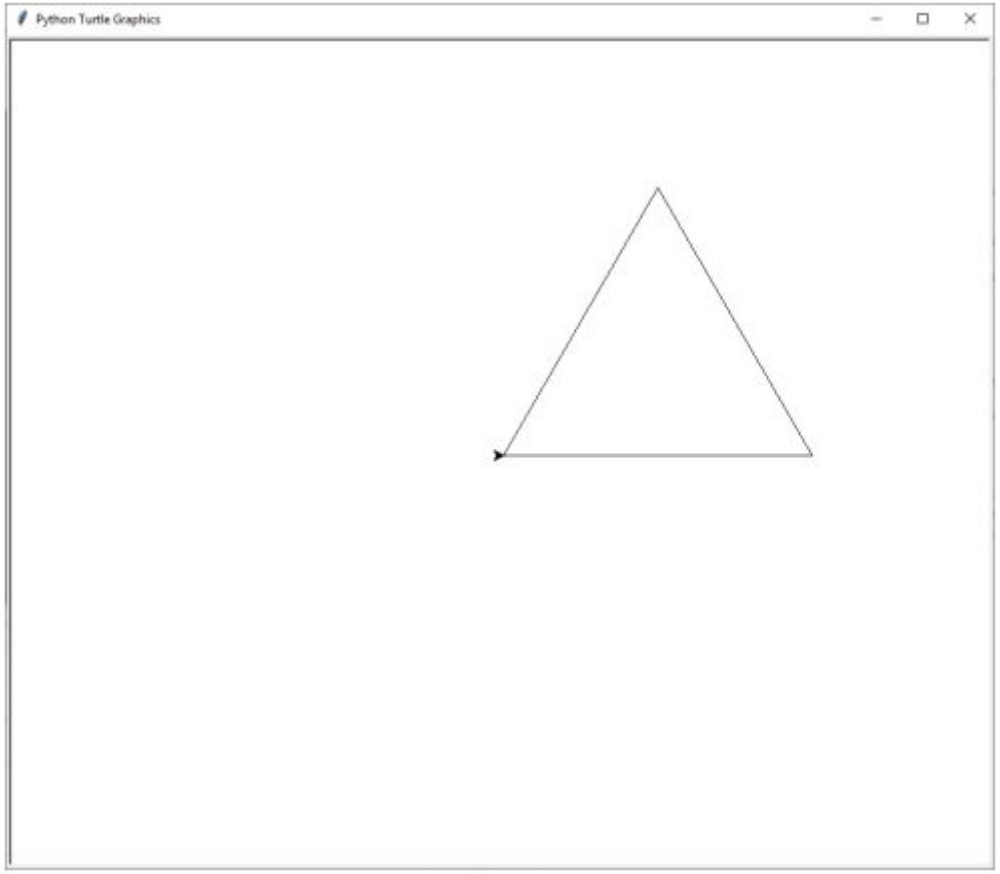
Indentation

Output

Put Statements into a Function

- For the assignment last week
- Your answer should be something like this

```
fd(300)
lt(120)
fd(300)
lt(120)
fd(300)
lt(120)
```



- But if I want to draw it again?
  - It’s too troublesome to type the above lines again and again

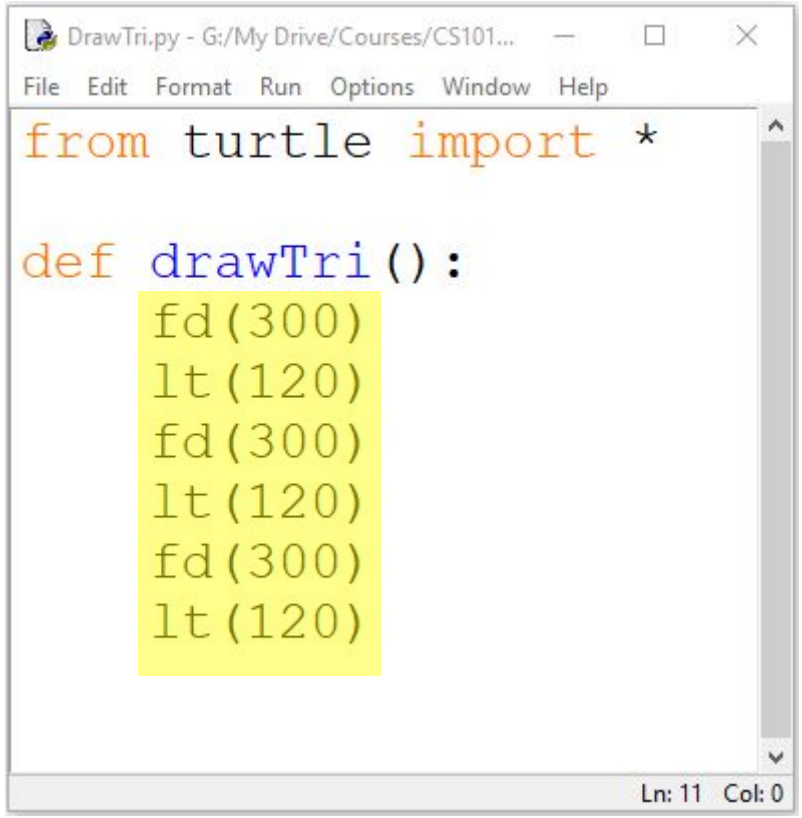
Put Statements into a Function

- Last week Assignment
- Your answer will be something like (yours maybe a bit different)

```
fd(300)
lt(120)
fd(300)
lt(120)
fd(300)
lt(120)
```

- But if I want to draw it again?
  - It's too troublesome to type the above lines again

- We save it into a file by
  - In IDLE, File > New



Put Statements into a Function

- After you save the file and run it
- You can call the function drawTri() by

```
>>> drawTri()
```

- Or
  - Directly put it into the file

- We save it into a file by
  - In IDLE, File > New

```
from turtle import *

def drawTri():
    fd(300)
    lt(120)
    fd(300)
    lt(120)
    fd(300)
    lt(120)

drawTri()
```

Function Parameters

- What if we want to draw a triangle that is larger or smaller
  - Namely, the side length is different from 300?
  - Do we write…

```
def drawTri():
    fd(200)
    lt(120)
    fd(200)
    lt(120)
    fd(200)
    lt(120)
```

```
def drawTri():
    fd(100)
    lt(120)
    fd(100)
    lt(120)
    fd(100)
    lt(120)
```

  - Etc…?

```
from turtle import *

def drawTri():
    fd(300)
    lt(120)
    fd(300)
    lt(120)
    fd(300)
    lt(120)

drawTri()
```

Capture the COMMON Pattern

This is an important skill in computational thinking

- What if we want to draw a triangle that is larger or smaller
  - Namely, the side length is different from 300?
  - Do we write…

```
def drawTri():
    fd(200)
    lt(120)
    fd(200)
    lt(120)
    fd(200)
    lt(120)
```

```
def drawTri():
    fd(100)
    lt(120)
    fd(100)
    lt(120)
    fd(100)
    lt(120)
```

  - Etc…?

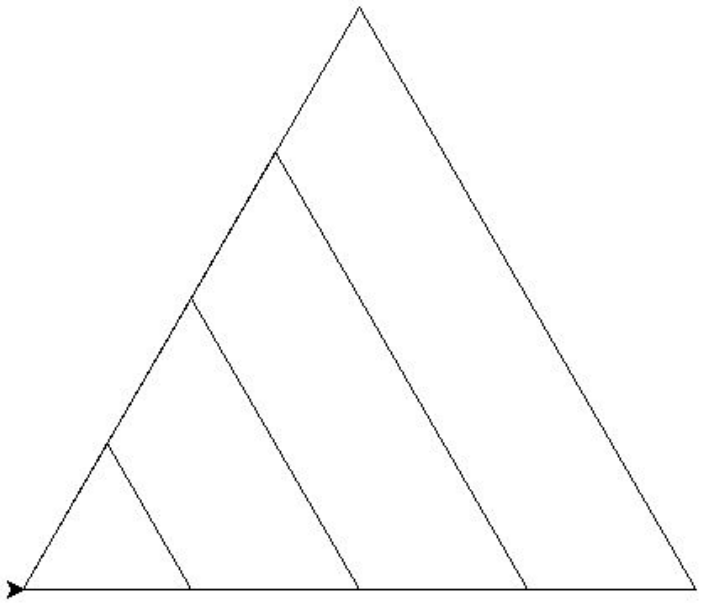
- No, we **capture the common pattern** and make it an input of the function

```
from turtle import *

def drawTri(length):
    fd(length)
    lt(120)
    fd(length)
    lt(120)
    fd(length)
    lt(120)

drawTri(100)
drawTri(200)
drawTri(300)
```

Capture the COMMON Pattern

This is an important skill in computational thinking

```
>>> drawTri(100)
>>> drawTri(200)
>>> drawTri(300)
>>> drawTri(400)
>>>
```

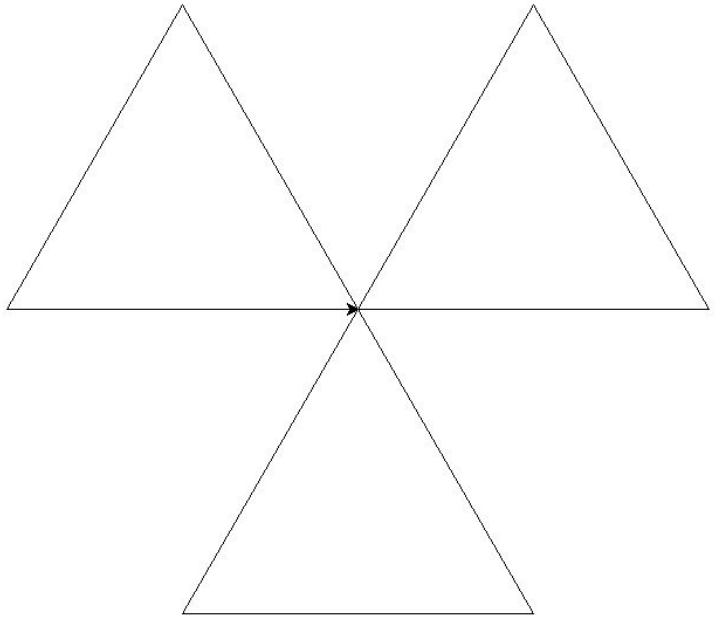


- No, we **capture the common pattern** and make it an input of the function

```
from turtle import *

def drawTri(length):
    fd(length)
    lt(120)
    fd(length)
    lt(120)
    fd(length)
    lt(120)

drawTri(100)
drawTri(200)
drawTri(300)
```

Moreover

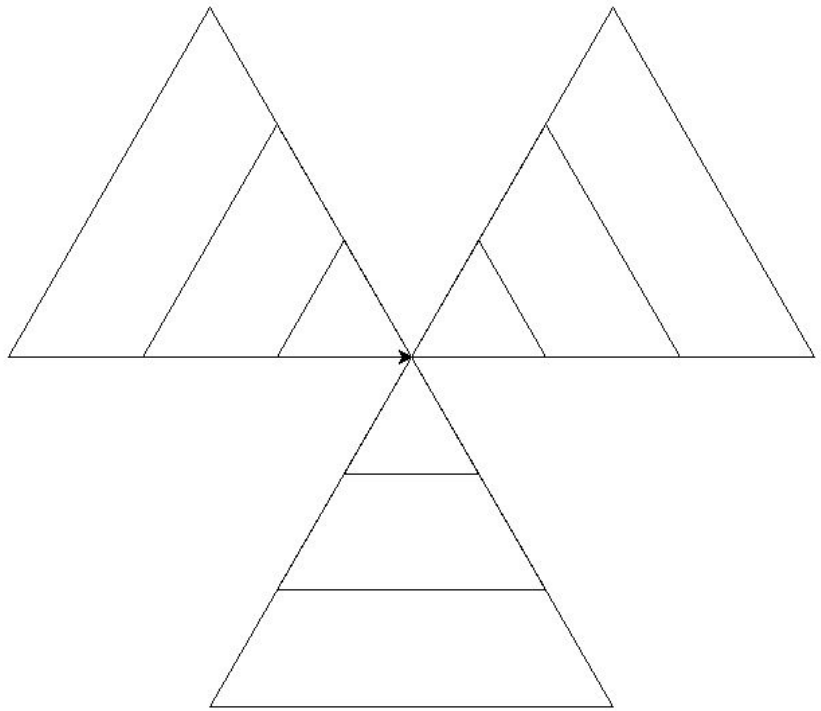
- What does this code do?
  - Output:

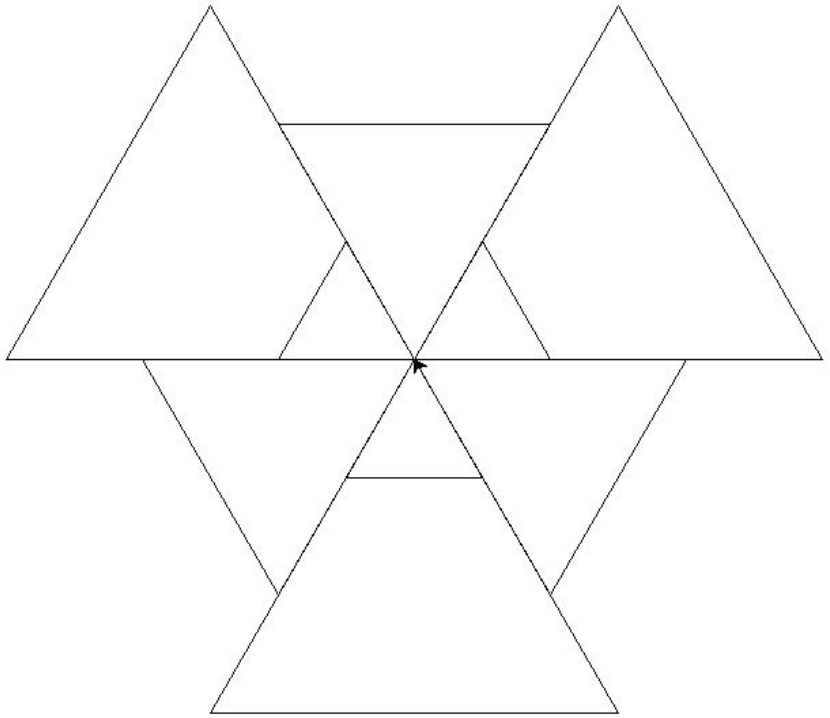


```
from turtle import *

def drawTri(length):
    fd(length)
    lt(120)
    fd(length)
    lt(120)
    fd(length)
    lt(120)

def foo():
    drawTri(100)
    lt(120)
    drawTri(100)
    lt(120)
    drawTri(100)
    lt(120)

foo()
```

Your Task: Draw These

- (10 min)
- Advanced Challenge: Try to draw these with a for loop
  - Or some other interesting patterns





“Return” of Functions

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

Python Shell (Console)

Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

```
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'
'Mississppi'
>>> x = 5 + 3
>>> x
8
>>>
```

Echoes: "return values" of the previous line

But sometimes there will NOT be any echo as there is NO "return value"

Ln: 10 Col: 4

A Function may or may not return a value

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

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

This function returns a value

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 said 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 the Python shell echoed it
- The 3 “Hello” are NOT return values 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
```

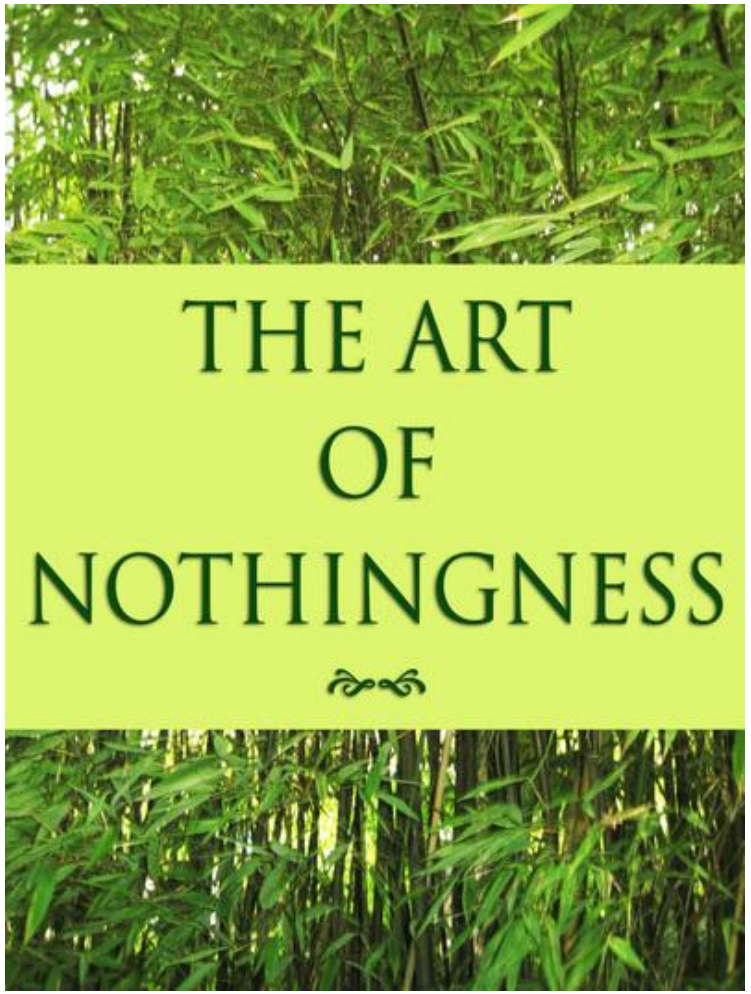
```
>>> foo_print3()
3
>>> foo_return3()
3
>>>
```

Wait…

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

Nothing?

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



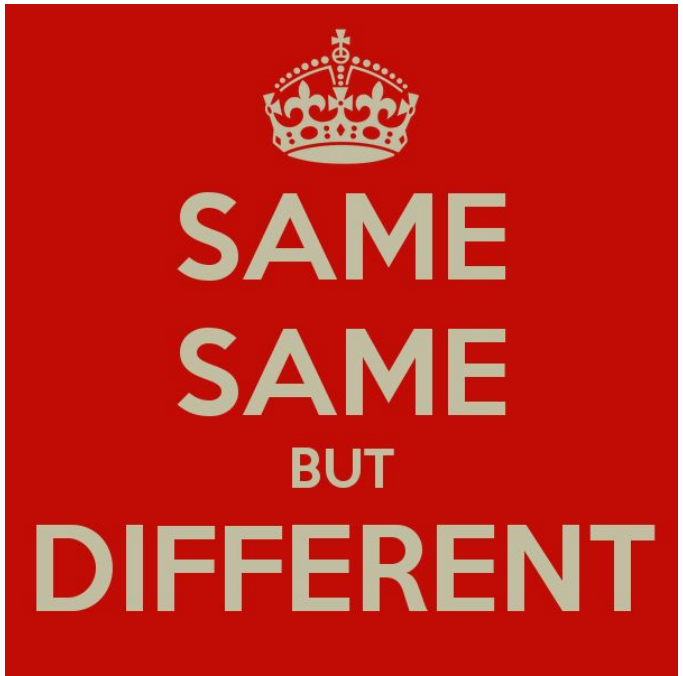
Print vs Return

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

def foo_return3():
    return 3
```

By the print function

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

IDLE's **echo**



Print vs Return

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

def foo_return3():
    return 3
```

- foo_print3() does not “return” a value

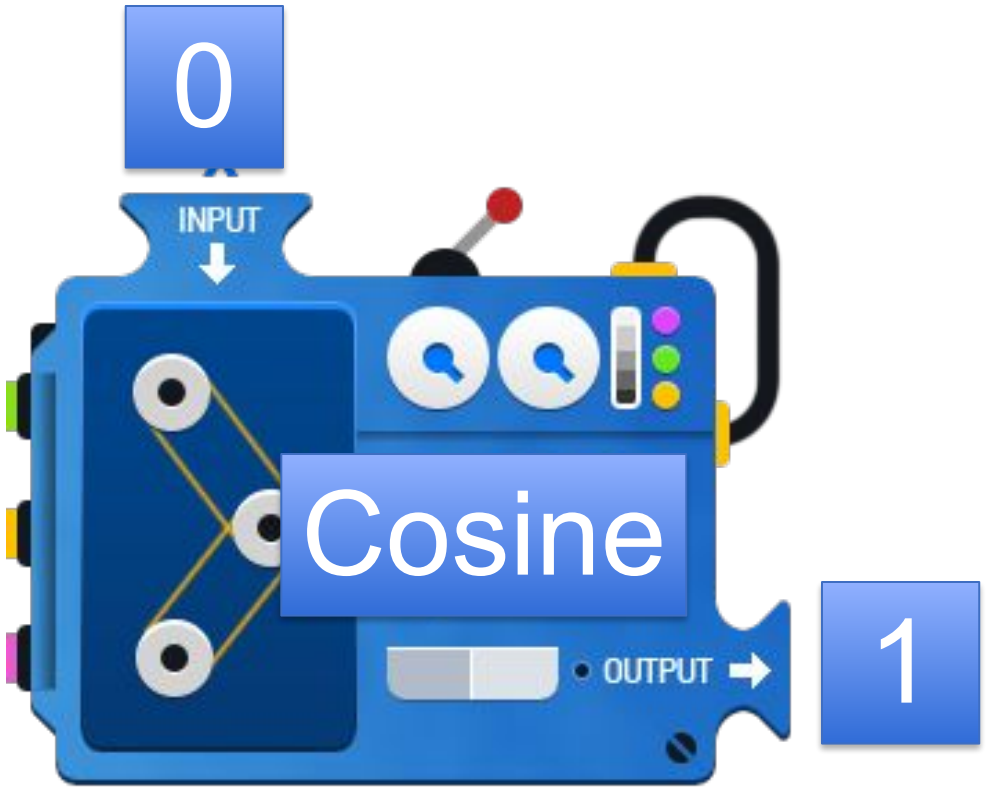
```
>>> x = foo_print3()
3
>>> y = foo_return3()
>>> |
```

IDLE echoes “nothing”

Function

- “Cosine” is a function
  - Input 0
  - Output/return 1
  - x = cos(0)

    return
  - That’s why x = 1

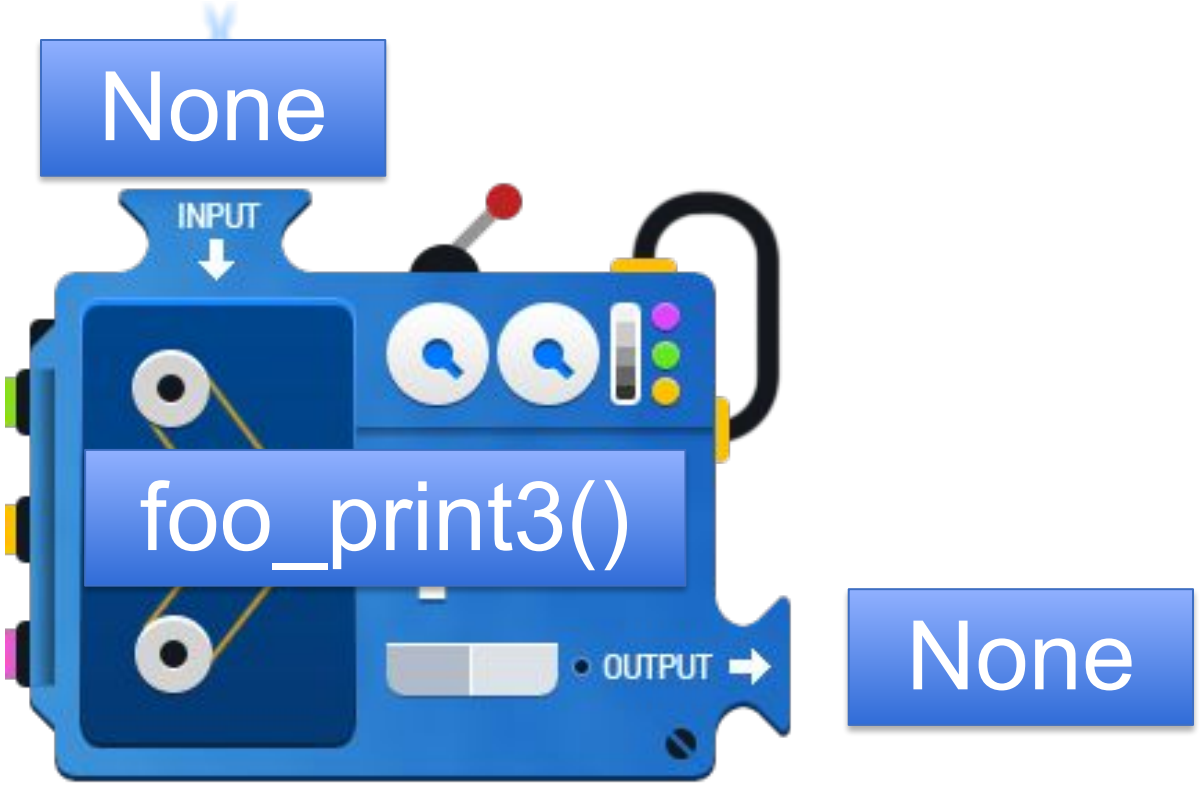


Function

- “`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, sometimes we call it a “procedure”

Return Values

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

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

Selection Statements

What will it return?

```
def foo():
    if True:
        if False:
            print(1)
        else:
            print(2)
```

A Computer Science thing:
If you don't want to name a function, just name it "foo()"

But DON'T do it in practice

if-else

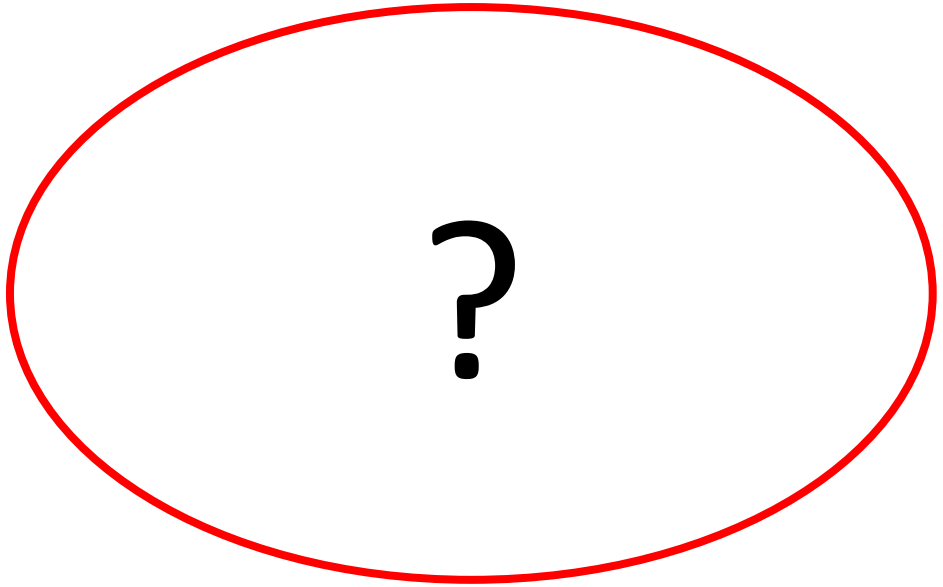
```
def foo():
    if False:
        if True:
            return 1
    else:
        return 2
```

```
def foo():
    if False:
        return 1
    elif False:
        return 2
    elif True:
        return 3
    elif True:
        return 4
    else:
        return 5
```

elif statements will ‘break’ the moment one of them is True

if-else

```
def foo():
    if not True:
        if True:
            print(1)
        else:
            print(2)
```



Be careful with your if-else. You might return nothing!

Can you spot the difference?

Example 1

```
def foo():
    if True:
        if False:
            print(1)
        else:
            print(2)
```

Example 2

```
def foo():
    if True:
        if False:
            print(1)
    else:
        print(2)
```

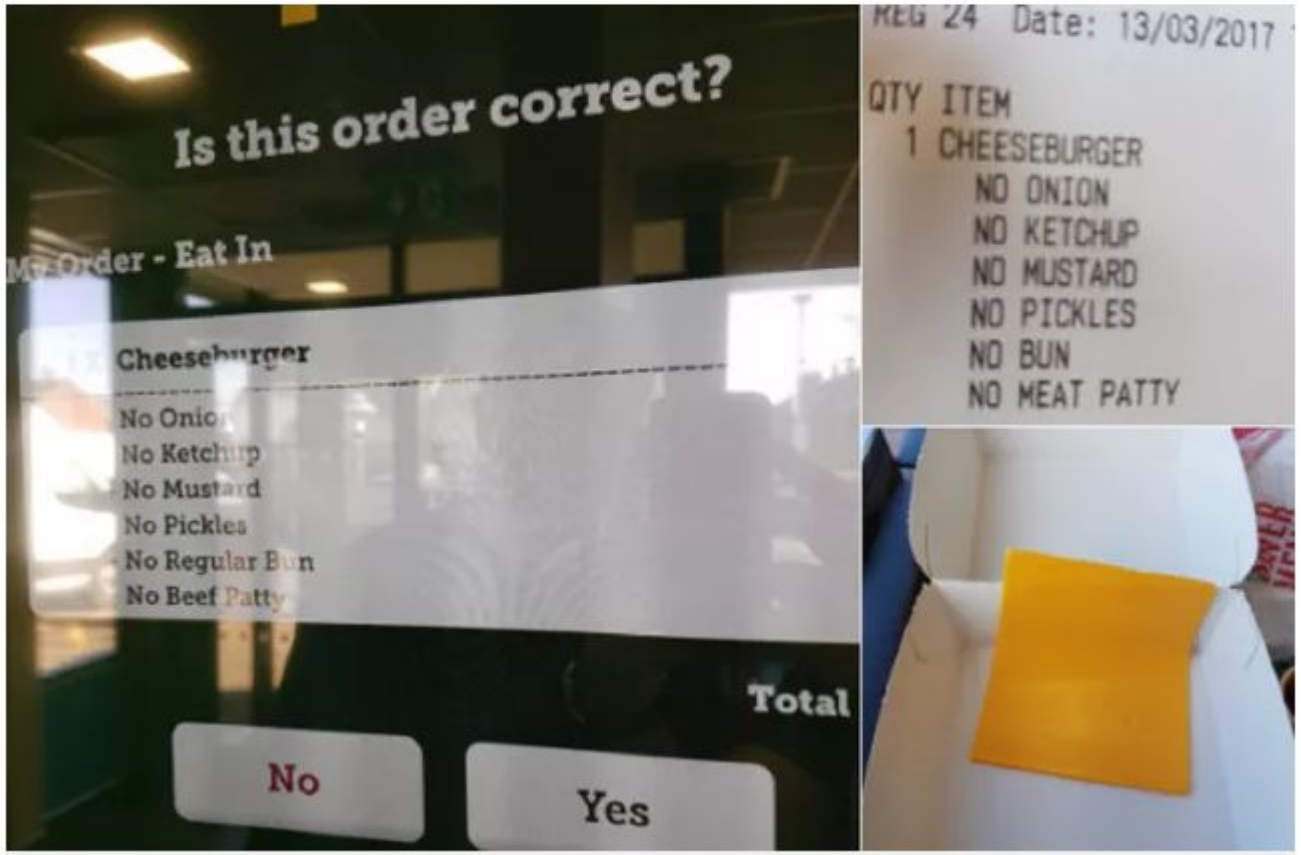
Let’s do some real coding!

Some Fast Foods offer the option for Customization

- Meaning, you can personalize what will or will not be inside 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 some 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** of a burger

- ‘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’)
3.2
```

Discuss with your neighbour on how to start/do it

(5 min)

Write a function `burgerPrice()` to calculate the **price** of a burger

- ‘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’)
3.2
```

How would 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 step(s) are repeated if there are many characters in the string?

- E.g.

```
>>> burgerPrice('BVPB')
3.2
```

How would 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 step(s) are repeated if there are many characters in the string?

- E.g.

```
>>> burgerPrice('BVPB')
3.2
```

How would 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"

Then you need to start with 0

Whenever you want to accumulate some sum or product, you need a variable to store it

- E.g.

```
>>> burgerPrice('BVPB')
3.2
```

How to go through each character of the string?

- 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])
```

Output:

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

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

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 makes 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”

```python
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')
```

```python
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'))
```

- Wait? Nothing happened after running this code?
- If you run a .py file, there will be NO Python echo

Are we done?

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

```
for <var> in <sequence>:
    <body>
```

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

- 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

“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

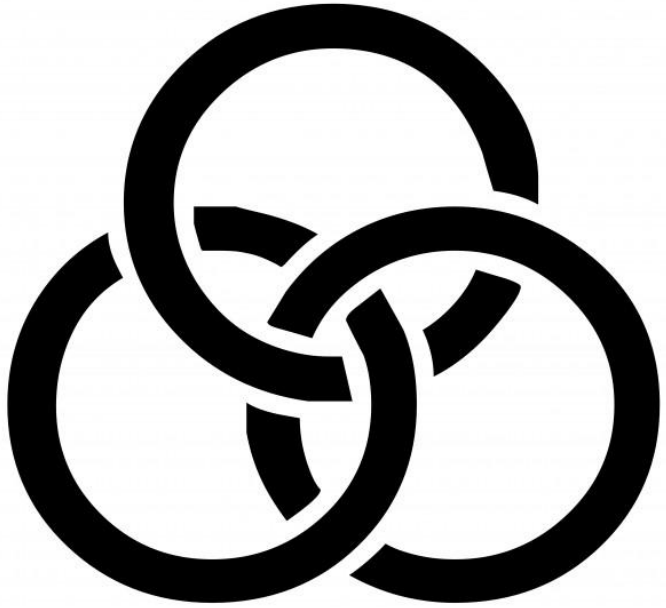
- Not only about how to get the final code but..
- Plan and write your code in English first
- You may need to write some intermediate code for a “semi finished product” to test out your idea
- After you finally get your code working, you should think about how to improve it
  - Not only for that single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

Extra: “match” (switch case)

- For those who know C++ or Java, Python’s match statement is a bit different
  - It doesn’t need a “break” in each case. So each case will not be “spread” to the next one
  - ”case _” will be the wildcard
  - On top of matching a variable x in “match x”, Python can match patterns/structures, but let’s leave it for later

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

Three Types of Loops



Which Type is it?

For A and C, it means you know the number N when your loop starts

A. Must run exactly N times (definite)
B. Runs any number of times (indefinite)
C. Runs 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?

- Iterative version of computing the factorial of N

A. Must run exactly N times (definite)
B. Runs any number of times (indefinite)
C. Runs 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. Runs any number of times (indefinite)
C. Runs 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. Runs any number of times (indefinite)
C. Runs 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 is prime

A. Must run exactly N times (definite)
B. Runs any number of times (indefinite)
C. Runs 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. Runs any number of times (indefinite)
C. Runs 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.