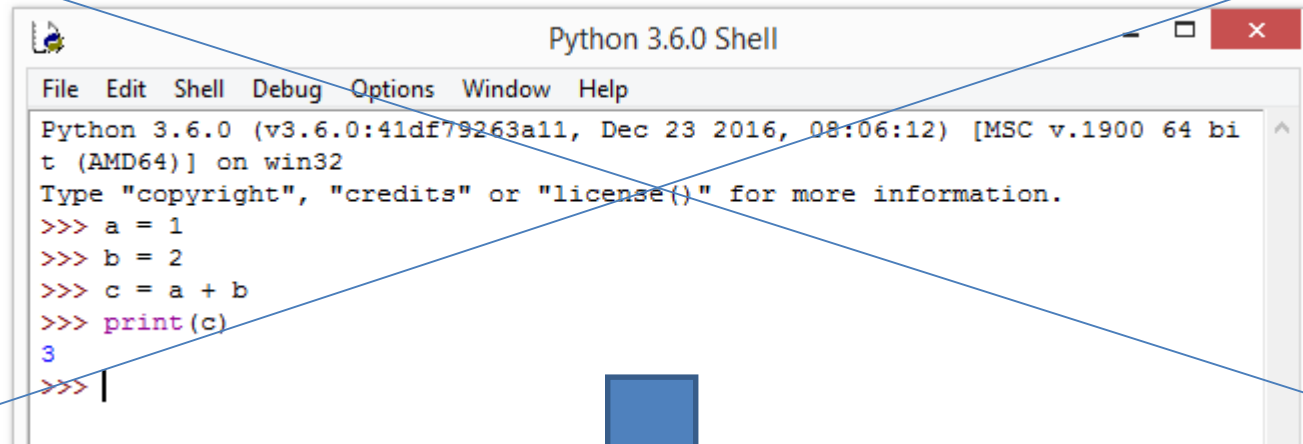
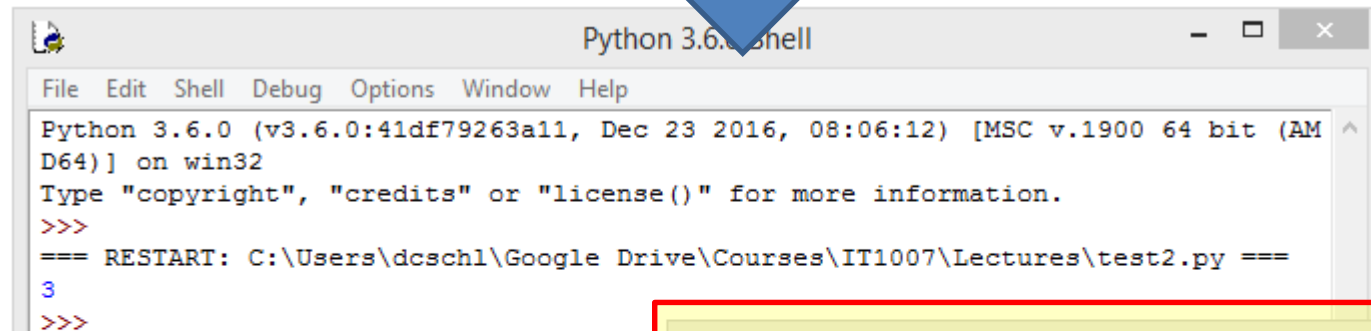


# Let's Move Out of the Console



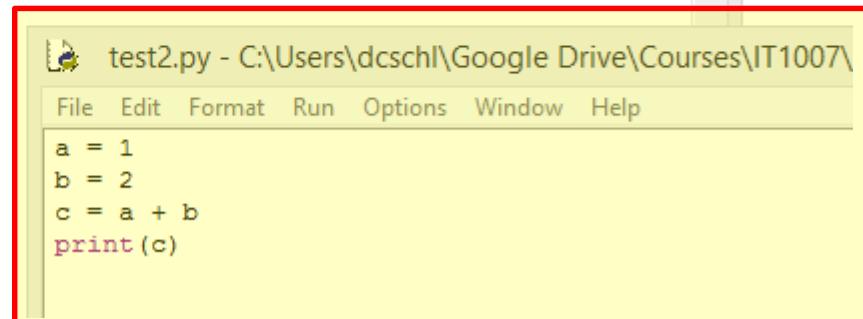
A screenshot of a Python 3.6.0 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The text inside shows the Python version and build information, followed by a prompt for copyright information. Below that, there are four lines of code: `>>> a = 1`, `>>> b = 2`, `>>> c = a + b`, and `>>> print(c)`. The output `3` is shown below the last line. A large blue arrow points from this window down to the next one.

```
Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a = 1
>>> b = 2
>>> c = a + b
>>> print(c)
3
>>> |
```



A screenshot of a Python 3.6.0 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The text inside shows the Python version and build information, followed by a prompt for copyright information. Below that, there is a line of code: `=== RESTART: C:\Users\dcscshl\Google Drive\Courses\IT1007\Lectures\test2.py ===`. The output `3` is shown below the line. The prompt `>>>` is shown at the bottom.

```
Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:\Users\dcscshl\Google Drive\Courses\IT1007\Lectures\test2.py ===
3
>>>
```



A screenshot of a text editor window titled 'test2.py - C:\Users\dcscshl\Google Drive\Courses\IT1007\Lectures\test2.py'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The text inside shows the code from the previous window: `a = 1`, `b = 2`, `c = a + b`, and `print(c)`.

```
test2.py - C:\Users\dcscshl\Google Drive\Courses\IT1007\Lectures\test2.py
File Edit Format Run Options Window Help
a = 1
b = 2
c = a + b
print(c)
```

# Recap: Simple Functions

Define  
(keyword)

Function name

Input  
(Argument)

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

Indentation

Output

The diagram illustrates the components of a Python function definition. The code `def square(x):` is shown with `def` in yellow, `square` in blue, and `(x):` in black. An arrow points from 'Define (keyword)' to `def`, another from 'Function name' to `square`, and a third from 'Input (Argument)' to `(x)`. Below the code, a curly bracket under the indented line `return x * x` is labeled 'Indentation'. Another curly bracket under the expression `x * x` is labeled 'Output'.

# Parameters of Functions

= input

= arguments

# Input Parameters

```
def add2things(a,b)  
    return a + b
```

Must be the same  
number of items

```
>>> add2things(1,2)
```

```
3
```

```
>>> add2things(1)
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#94>", line 1, in <module>
```

```
    add2things(1)
```

```
TypeError: add2things() missing 1 required positional argument: 'b'
```

```
>>> add2things()
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#95>", line 1, in <module>
```

```
    add2things()
```

```
TypeError: add2things() missing 2 required positional arguments: 'a' and 'b'
```

```
>>> add2things(1,2,3)
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#96>", line 1, in <module>
```

```
    add2things(1,2,3)
```

```
TypeError: add2things() takes 2 positional arguments but 3 were given
```

```
^^^
```

# Parameter Types

- In Python, parameters have no declared types. We can pass any kind of variable to the function....

```
>>> add2things(3.14, 2.71)
```

```
5.85
```

```
>>> add2things('Hello ', 'world!')
```

```
'Hello world!'
```

```
>>> add2things(True, True)
```

```
2
```

```
>>>
```

.... as far as the function works

# Pass by Values

```
x = 0
```

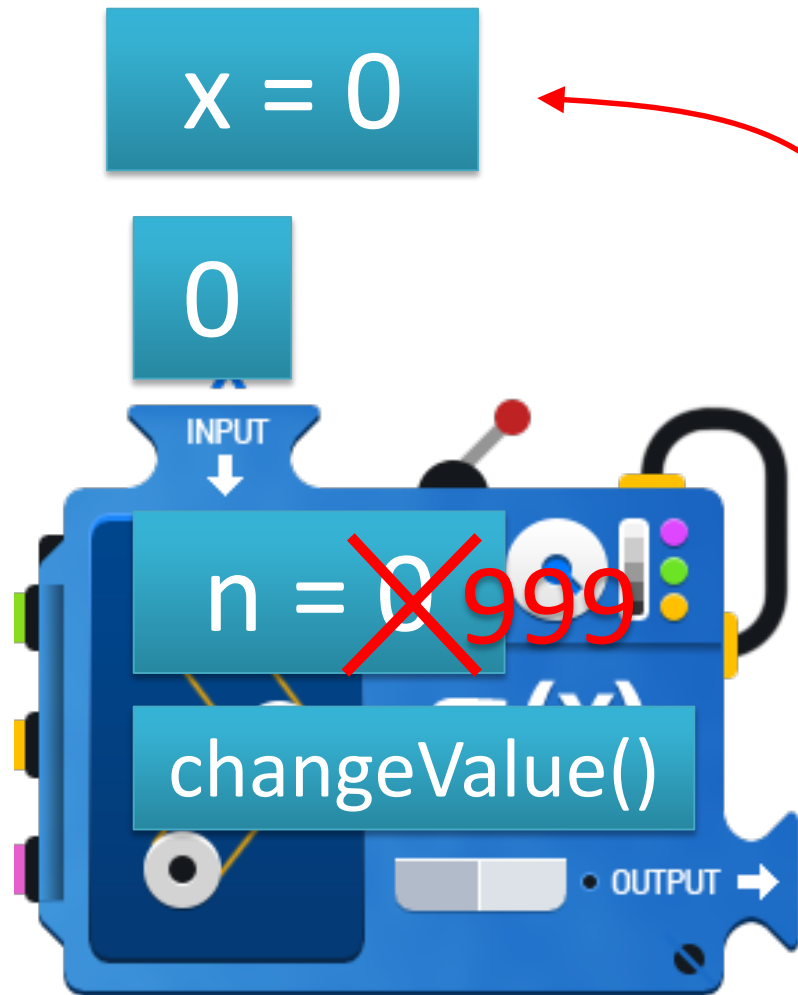
```
def changeValue(n):  
    n = 999  
    print(n)
```

```
changeValue(x)  
print(x)
```

- The print () in “changeValue” will print 999
- But how about the last print(x)?
  - Will x becomes 999?
- (So actually this function will NOT change the value of x)

# Function

- “changeValue()” is a function
  - $x = 0$
  - Input  $x$  into the function
  - Another  $n$  inside the function copied that value
  - And you changed that  $n$ , it doesn't affect  $x$



# Pass by Values

```
x = 0
```

```
def changeValue(n):  
    n = 999  
    print(n)
```

```
changeValue(x)  
print(x)
```

- n is another copy of x
- You can deem it as

```
def changeValue(x):  
    n = x  
    n = 999  
    print(n)
```



# For Parameters that are Primitives

- Primitive data:
  - int, float, bool, etc.
- Parameters are passed by values
- But NOT for *some* parameters
  - E.g. sequences
  - Will discuss about this in later lectures

# Return Values

Vs “print()”

# Print vs Return

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

```
>>> foo_print3()
```

```
3
```

```
3
```

```
>>> foo_return3()
```

```
3
```

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

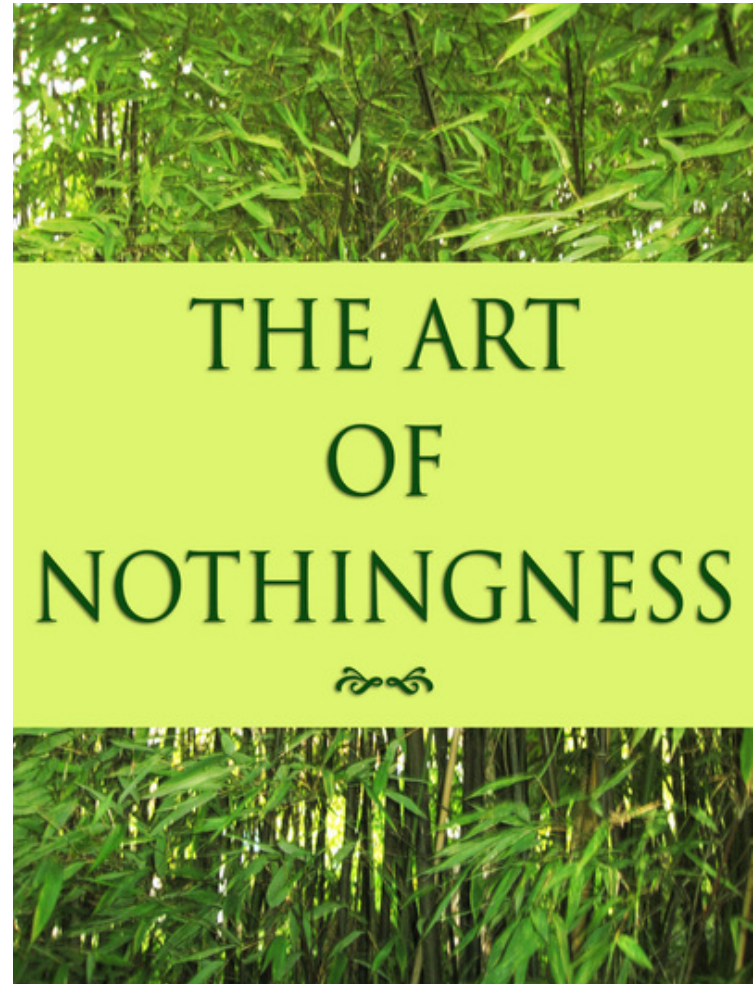


# Wait...

```
>>> 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
```

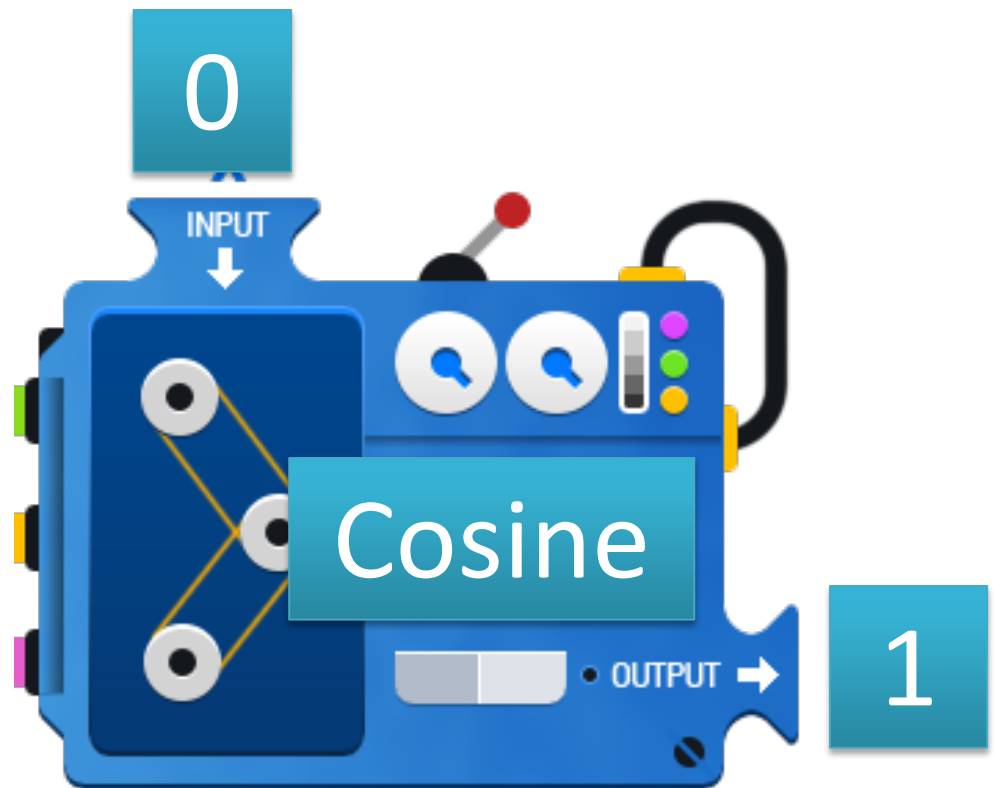
```
>>> y = foo_return3()  
>>> |
```



IDEL 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

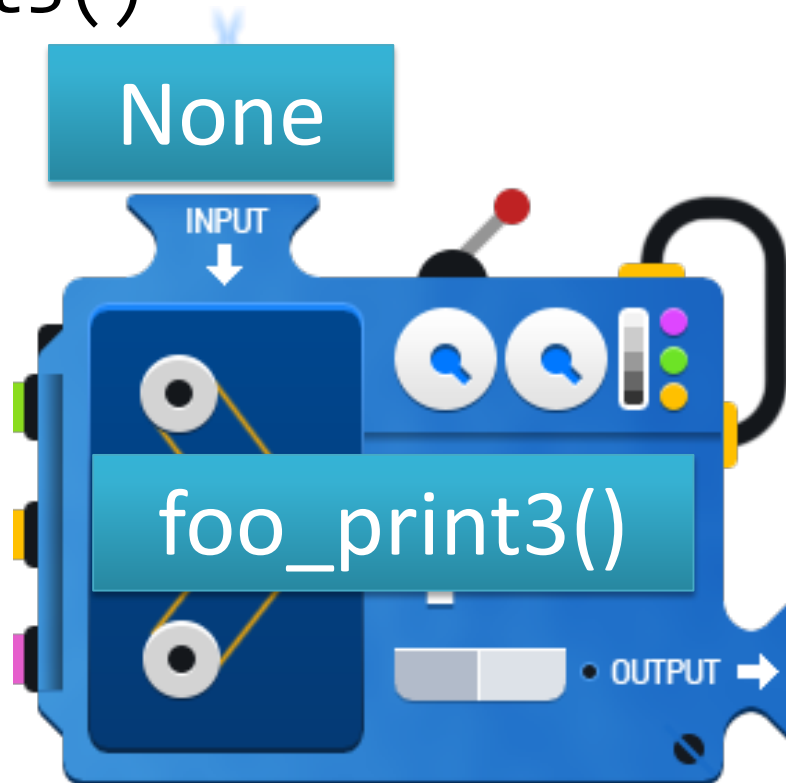
- Input nothing

- No output

y = foo\_print3()



return “None”



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?