# 5. FUNCTION

Victor Lee

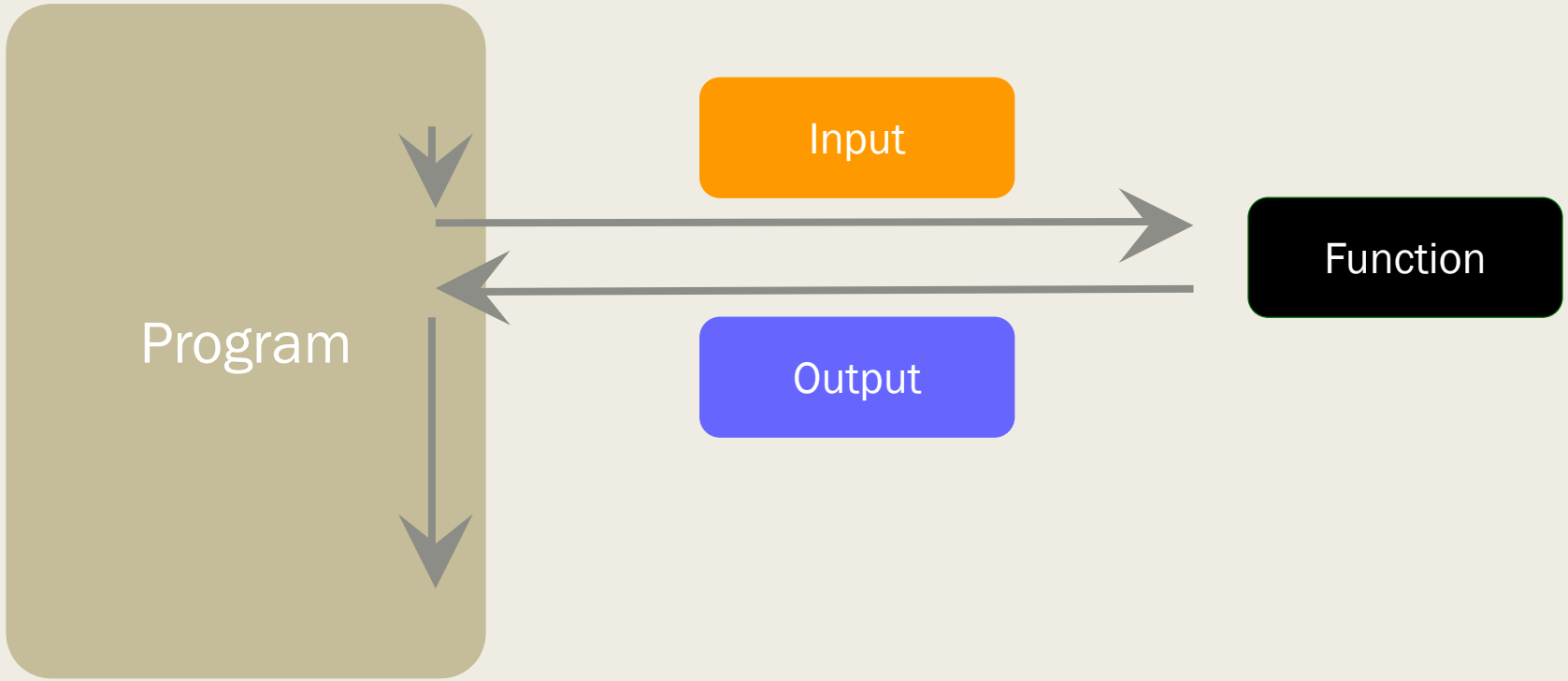
**Department of Electrical and Electronic Engineering**

# Outline

- Function Declaration
- Parameter Passing
- Return Value
- Scope of Variable

# Computer Program vs. Function

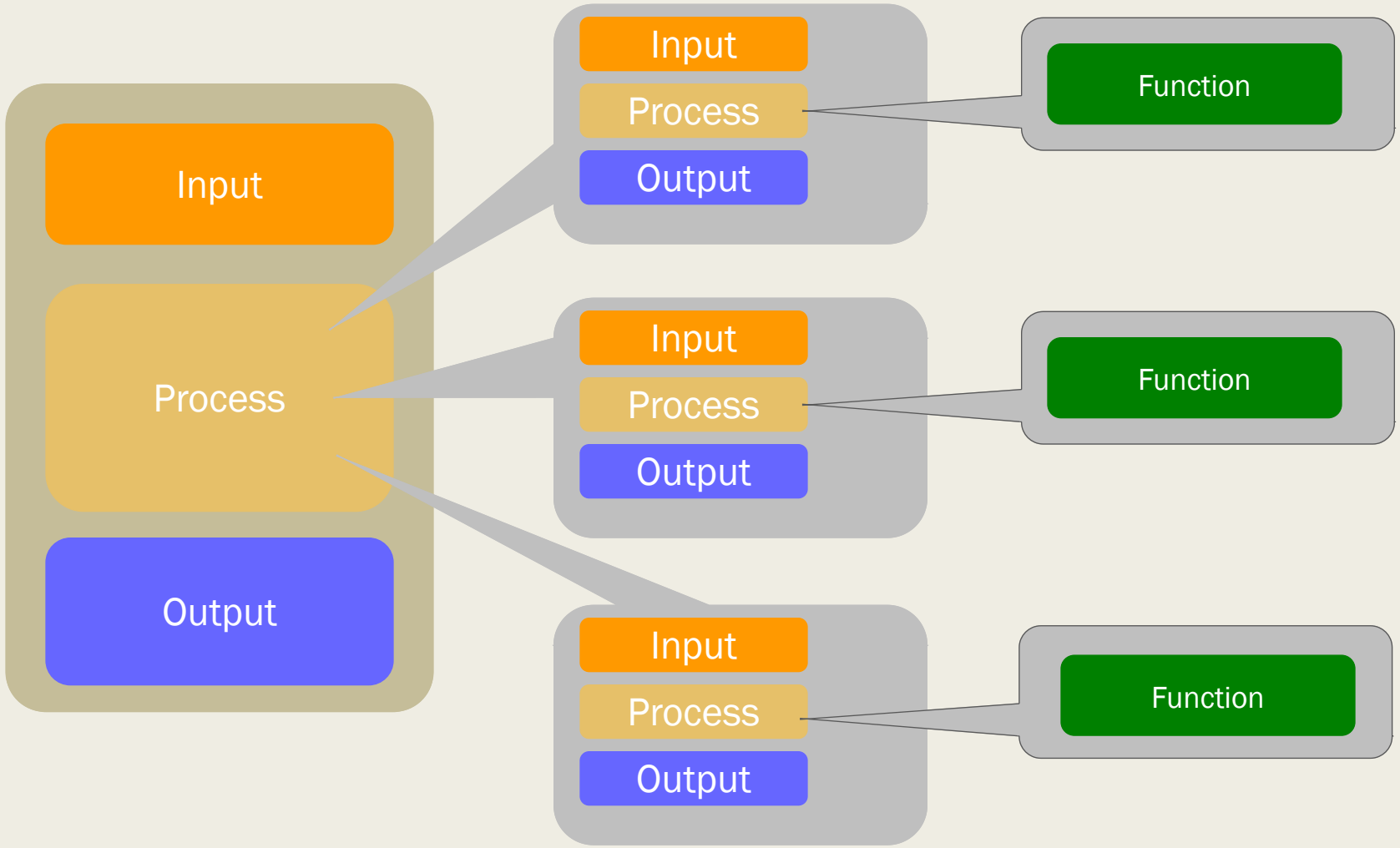
# Computer Program vs. Function



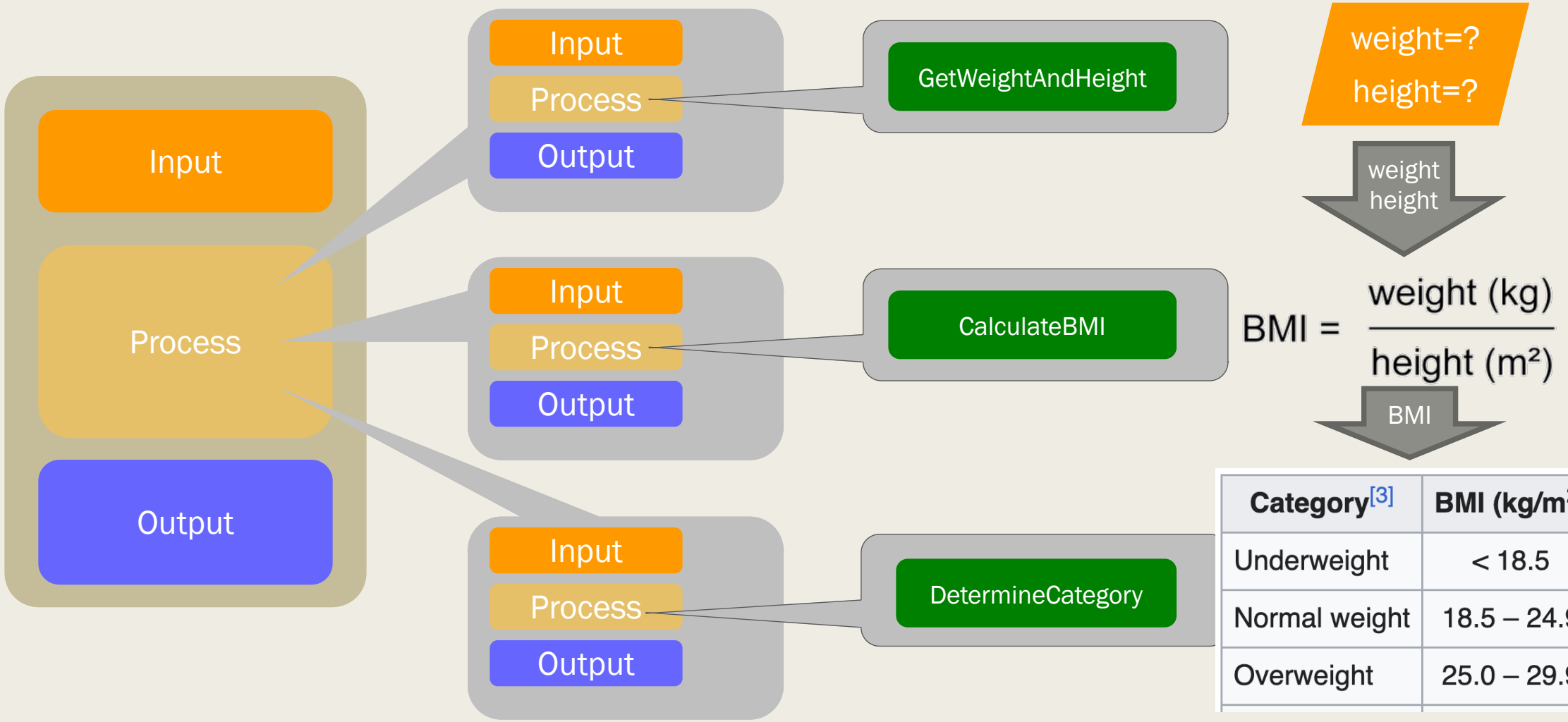
# What is a Function? Why Functions?

- Functions are used to break a complex problem down into manageable pieces of tasks (stepwise refinement).
- A function is a collection of statements that perform a specific task
  - *making your program easier to read.*
- Write several small functions to solve specific parts of the problem
  - *allowing you to debug the parts one at a time and then assemble them into a whole working program.*
- A function can be invoked multiple times. No need to repeat the same code in multiple parts of the program
  - *making the program smaller. Later, if you make a change, you only have to make it in one place.*

# Computer Program vs. Function



# Example: Am I Fat?




| Category[3] | BMI (kg/m²) |
|---|---|
| Underweight | < 18.5 |
| Normal weight | 18.5 – 24.9 |
| Overweight | 25.0 – 29.9 |

# Built-in Functions

- Input / output
  - `input()`, `print()`
- Data
  - `type()`, `int()`, `float()`, `str()`
- Math
  - `abs()`, `bin()`, `hex()`, `min()`, `max()`

List of Python built-in functions

# How to use function written by others?

```
import math

print("Please enter an angle")
a=float(input())
print("The sin value of "+str(a)+" is "+str(math.sin(a)))
```

A module is a separate Python file containing definitions and statements.

Tell interpreter that you are going to use functions defined in math module

Call the function with module name as prefix followed by a dot

`a` is the **argument** of the function

# Useful Functions in Math Module

| | |
|---|---|
| `ceil(x)` | Return the smallest integer greater than or equal to x |
| `floor(x)` | Return the largest integer less than or equal to x |
| `exp(x)` | Return e raised to the power of x |
| `log(x [, base])` | Return the natural log of x (to base e) |
| `log10(x)` | Return the base-10 log of x |
| `pow(x,y)` | Return x raised to the power y |
| `sqrt(x)` | Return the square root of x |
| `sin(x), cos(x), tan(x)` | Return the sine, cosine and tangent of x radians |
| `asin(x), acos(x), atan(x)` | Return the arc sine, arc cosine and arc tangent in x radians |
| `degrees(x), radians(x)` | Convert angle x from radians to degrees and vice versa |

The math module also contains related variables such as `pi`. So, it can be accessed using `math.pi`

# Define Your Own Function

```
def function_name():
      first_statement
      second_statement
      last_statement
```

Header: First line of the function, start with keyword `def`, followed by the function name, a pair of parentheses and a colon

Empty parentheses indicates no parameter (to be introduced)

Body: may contain any number of **equally indented** statements.
Add an empty line at the end if a function is defined in interactive mode

```
def sayHello():
    print("Hello, how are you?)
```

# Calling a Function

- The statements inside the function do not run until the function is called.
- To call a function, we only need to specify a function name and provide argument(s), if any, in the pair of ().

```
def sayHello():
    print("Hello, how are you?")

sayHello()
```

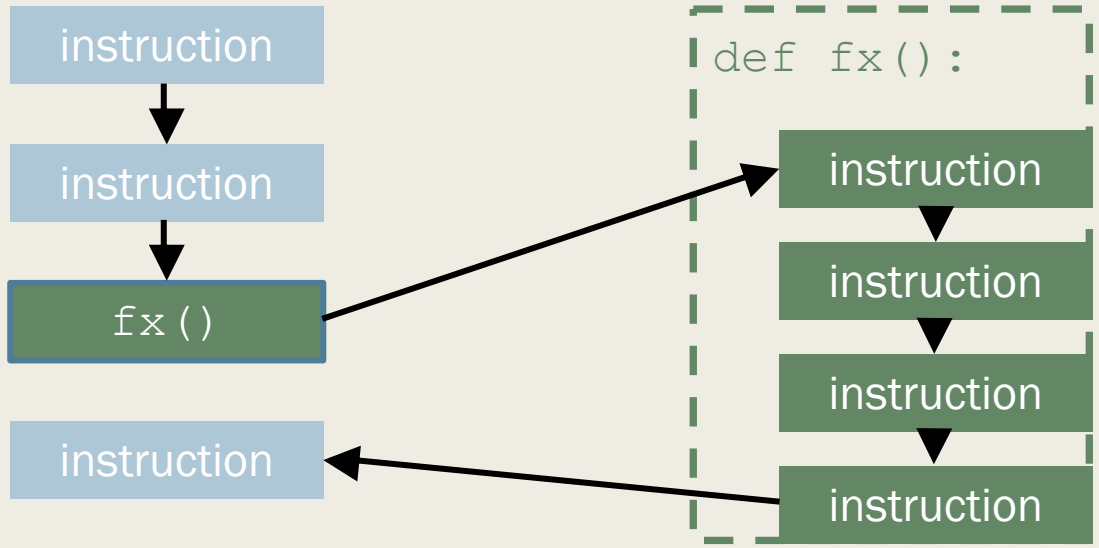
Function name

```
sayHello()

def sayHello():
    print("Hello, how are you?")
```

Can I do it this way?

# Program Control in Calling a Function

- During program execution, when a function name followed by parentheses is encountered, the function is invoked and the program control is passed to that function.
- When the function ends, program control is returned to the statement immediately after the function call in the caller function/program.



# Parameters and Arguments

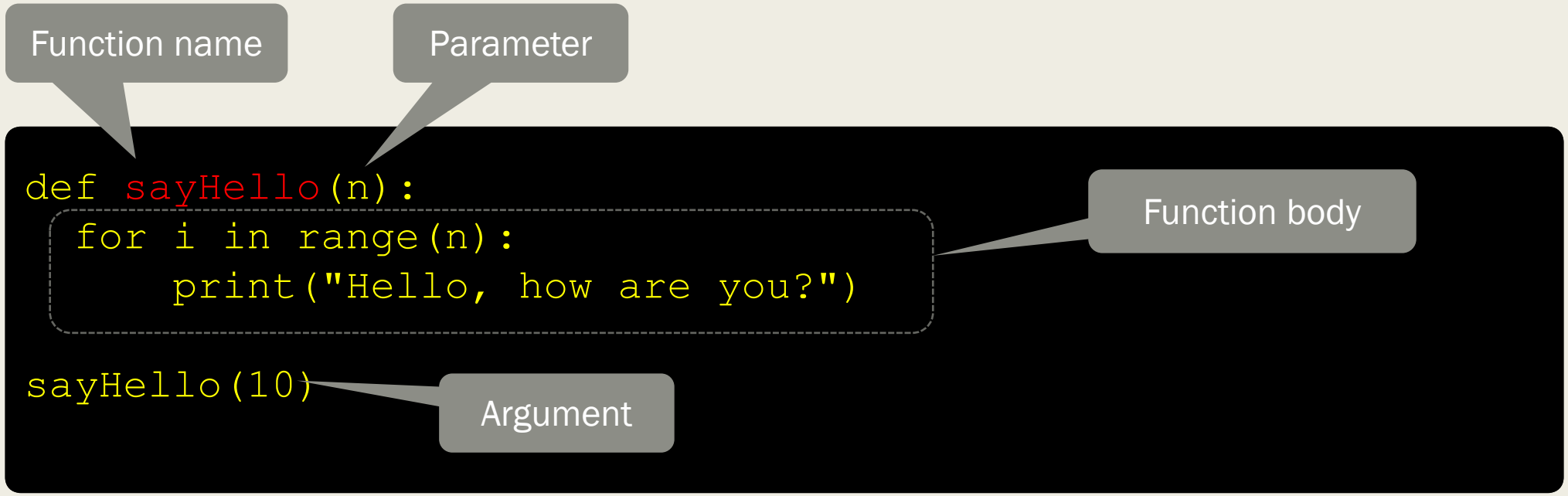
- Arguments
  - *Data passing to a function when the function is called*
  - *Will be assigned to variables called **parameters** inside the function*

```
def function_name(parameter):
      first_statement
      second_statement
      last_statement
```

```
def function_name(parameter1, parameter2):
      first_statement
      second_statement
      last_statement
```

Multiple parameters should be separated by ","

# Define a Function that Accepts a Parameter




- `n` is a parameter.
- When `sayHello(10)` is called, the caller passes the number `10` as an argument to the function.
- The function assigns `10` to `n` and prints the message 10 times.

# Passing Value to a Function

```
def sayHello(n):
  for i in range(n):
      print("Hello")

x=2
sayHello(10)
sayHello(x)
sayHello(x+3)
```

- The argument can be an expression.
- The argument is evaluated before the function is called.

# Define and Call a Function with Multiple Parameters

```
def sayHello(msg,n):
    for i in range(n):
        print(msg)
```

```
sayHello("Hello", 10)
```

The number of arguments should match the number of parameters.

`"Hello"` is copied to `msg`

`10` is copied to `n`

# Optional Parameters

- Python allows default values for a function's parameters.

```
def sayHello(msg,n=10):
      for i in range(n):
            print(msg)
```

```
sayHello("Hello")
```

If only one argument is provided, the value of `n` will be equal to the default value 10.

# Function Returns Value

- In addition to passing data to a function, function can return data to the caller.

```
def function_name(parameter):
    first_statement
    second_statement
    last_statement
    return return_value
```

- It can return a value / an object (covered later).
- Must be the **last** statement of a function.

# Multiple Parameters with Value Returned

```
def sayHello(msg, n):
    for i in range(n):
        print(msg)
    return n
    print("end of function call")


c=sayHello("Hello", 10)
print("The message has been printed "+str(c)+" times")
```

This statement will never be executed.

When a function call is part of a statement, program control is returned to the same statement to complete its execution.

# Flow of Control

```
def sayHello(msg, n):
    for i in range(n):
        print(msg)
    return n

c=sayHello("Hello", 10)
print("The message has been printed "+str(c)+" times")
```

1. The body of `sayHello` is executed

2. `SayHello` returns the value of `n` and passes the control back to the caller program

3. Returned value is assigned to variable `c`

`sayHello()` is called, the string `"Hello"` is copied to variable `msg` and the number `10` is copied to variable `n`

# Function with Value Returned

- The value to be returned is specified by the keyword `return`.
  - `return 3`
  - `return a`
  - `return a+3`

- The returned value can be assigned to any expression.
  - `a= sayHello("hello", 2)`
  - `a= sayHello("hello", 2)*2+1`

# Multiple Values Returned

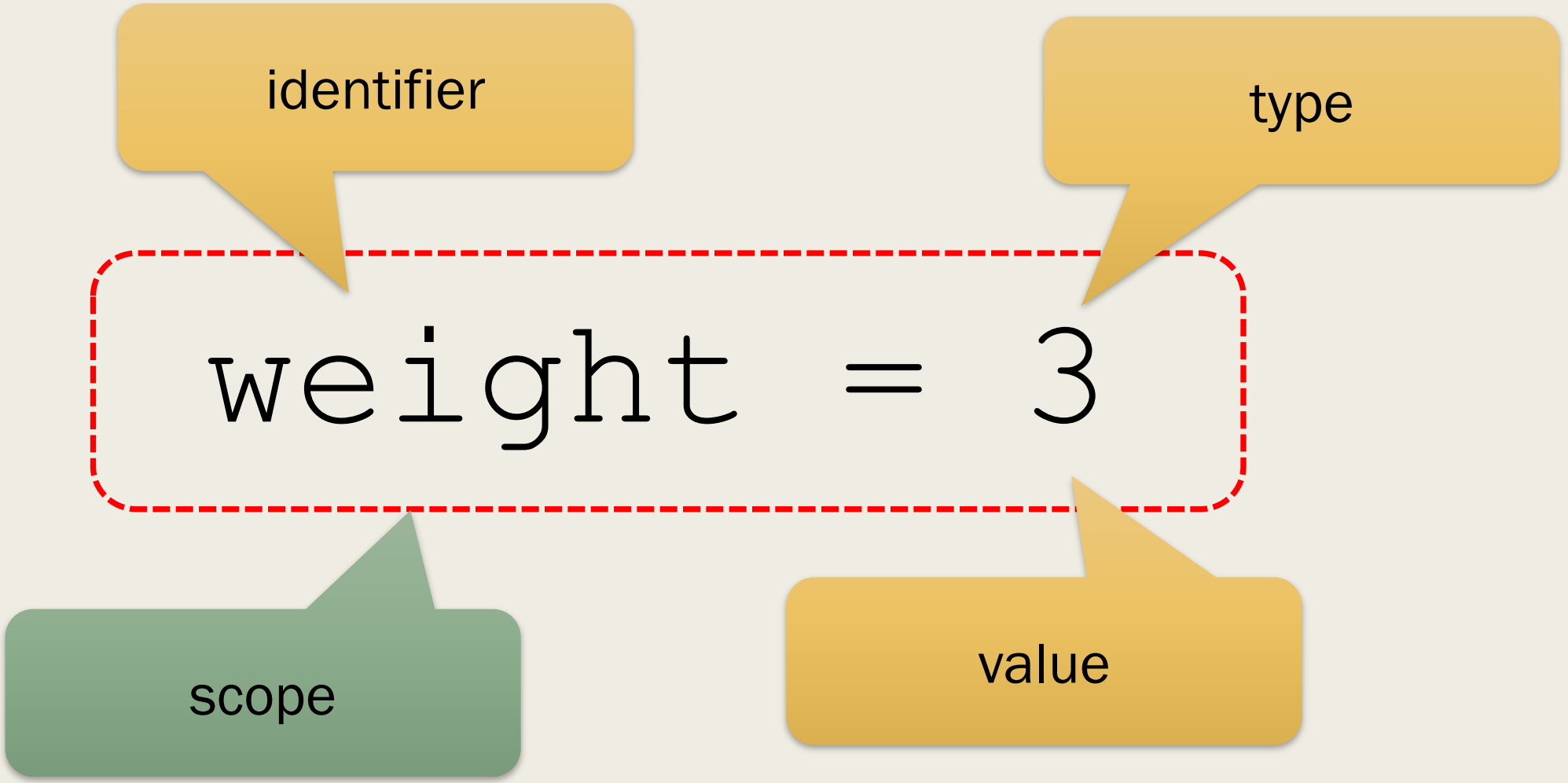
```
def sayHello(msg, n):
    for i in range(n):
        print(msg)
    return msg, n
```

Multiple values separated by ","

Number of variables to be assigned must be equal to the number of values returned

```
s,c=sayHello("Hello", 10)
print("The message "+s+" has been printed "+str(c)+" times")
```

# Variable



# Variable Scope: Global vs. Local

- Scope determines the region of the program in which a defined variable / object is visible (can be accessed).
- **Global variable**
  - Variable created (assigned) outside *any* function (global area).
  - Can be accessed by all functions
- **Local variable**
  - Variable created inside a function.
  - Parameters of a function are also local.
  - Can only be accessed within the function.
  - Try to access a local variable outside the function will generate `NameError`.
  - Local variables created in a function must have unique names
  - Local variables of **different** functions may use the same names. Although they have the same names, they are different variables.

# Example

```
def msg(s):
    sig="--Message from "
    print(s+sig+course)



course="ENGG1330"
msg("Welcome")
```

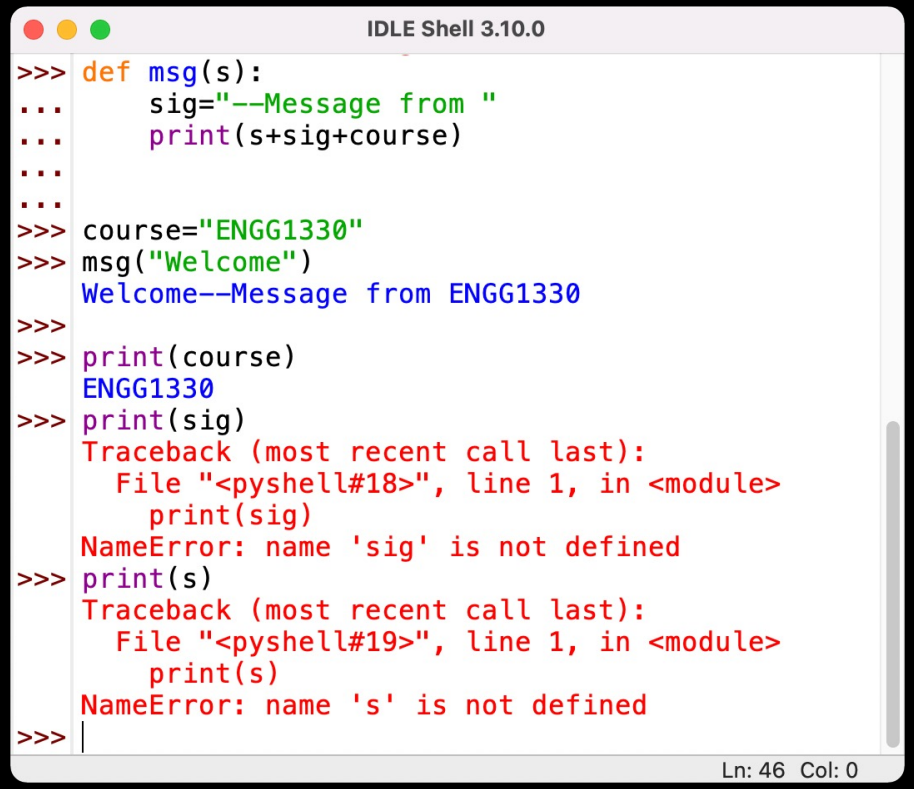
Local variables `s` and `sig`

Access global variable `course`

Global variable

# Rule 1: Local Variable Cannot Be Accessed Outside the Function

```
def msg(s):
    sig="--Message from "
    print(s+sig+course)


course="ENGG1330"
msg("Welcome")

print(sig)
```



# Rule 2: Global Variable Cannot Be Changed Within a Function, Unless....

```
def msg(s):
    sig="--Message from "
    print(s+sig+course)
    course="ENGG1330 Programming I"



course="ENGG1330"
msg("Welcome")
print(course)
```

2. Then, it is an error to refer a variable before assignment

1. Interpreter will assume you want to create a local variable call "course"

# Keyword `global` is Used

```
def msg(s):
    global course
    sig="--Message from "
    print(s+sig+course)
    course="ENGG1330 Programming I"



course="ENGG1330"
msg("Welcome")
print(course)
```

Use `global` to tell interpreter that you want to update the global variable

It may be hard to debug if a global variable can be updated by many functions

# Rule 3: Variable Created in Function is Local Regardless of its Name

```
def msg(s):

    sig="--Message from "
    course="ENGG1330 Programming I"
    print(s+sig+course)



course="ENGG1330"
msg("Welcome")
print(course)
```

A local variable, also called `course` has been created.

Modify `course` will not update the global variable

# Rule 3: Variable Created in Function is Local Regardless its Name

```
def msg(s, course):

    sig="--Message from "
    course="ENGG1330 Programming I"
    print(s+sig+course)



course="ENGG1330"
msg("Welcome", course)
print(course)
```

The parameter `course` is still a local variable

Modify `course` will not update the global variable

# Dive into Argument Passing

- Arguments are passed to a function by assigning values/objects to local variables (parameters).
- If the arguments are **immutable** like integers, floats or strings, changing their values inside a function does **not** affect the values of the variables outside the function.
- If the arguments are **mutable** (to be covered later in the course), changes in a function may impact the caller.

# Example

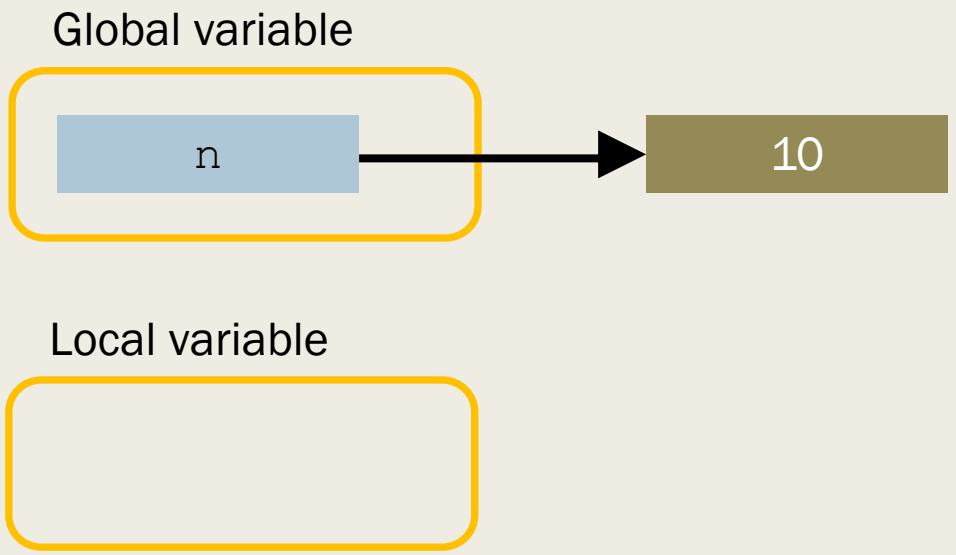
- Suppose we want to write a function to read an integer from user input, one may come up with this code

```
def getInt(userInput):
    print("Please enter an integer:")
    userInput=int(input())


n=10
getInt(n)
print(n)
```

What is the value of `n` printed on the screen, if user enters "23"?

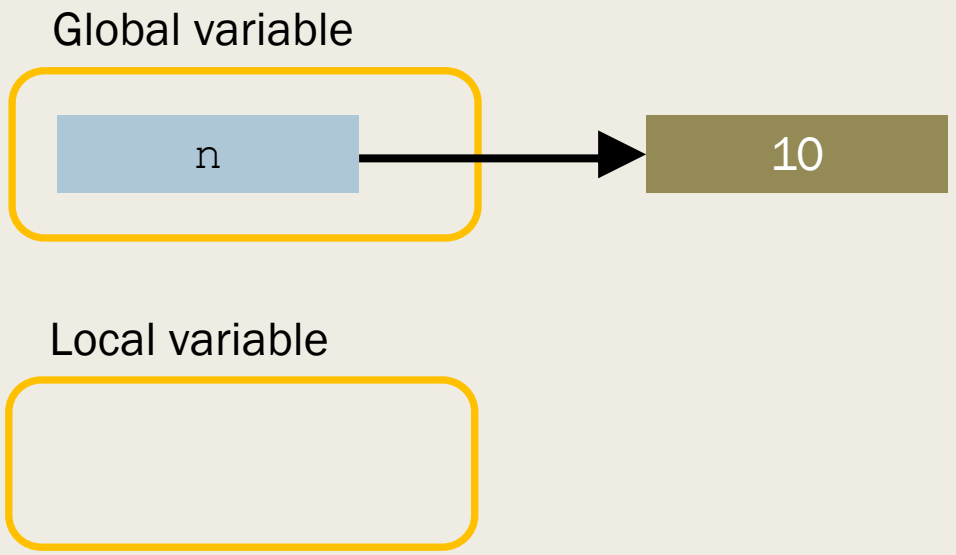
# Behind the Scene.....

```
def getInt(userInput):
    print("Please enter an integer:")
    userInput=int(input())


n=10
getInt(n)
print(n)
```



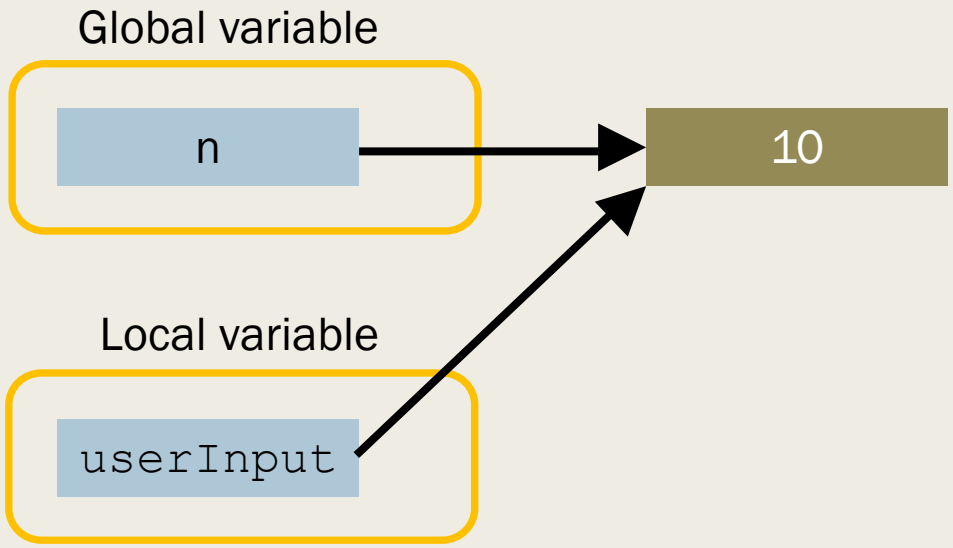
# Behind the Scene.....

```
def getInt(userInput):
    print("Please enter an integer:")
    userInput=int(input())


n=10
getInt(n)
print(n)
```



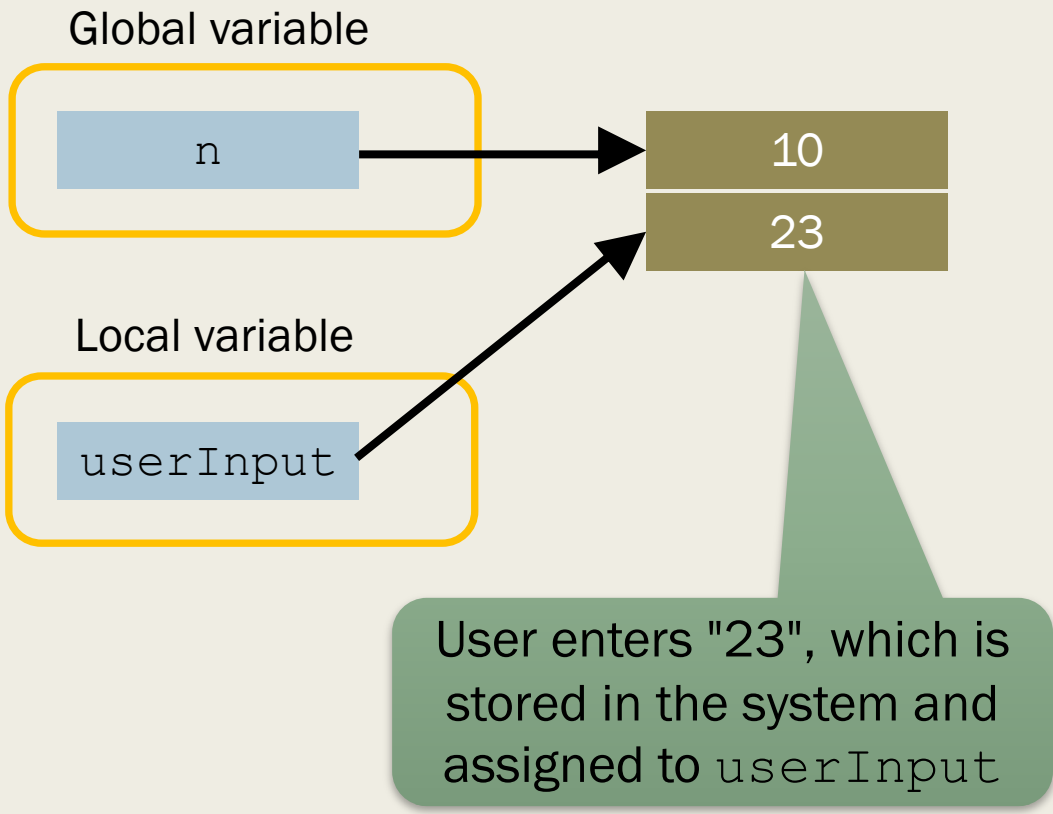
# Behind the Scene.....

```
def getInt(userInput):
    print("Please enter an integer:")
    userInput=int(input())


n=10
getInt(n)
print(n)
```



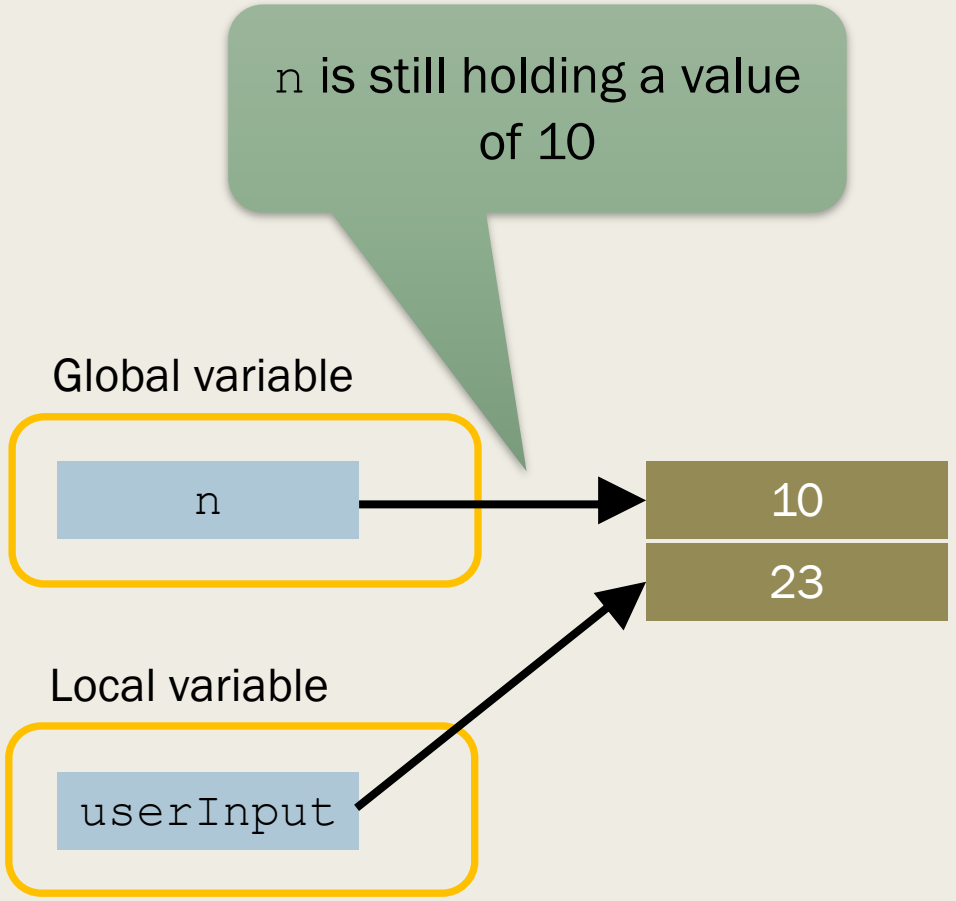
# Behind the Scene.....

```
def getInt(userInput):
    print("Please enter an integer:")
    userInput=int(input())


n=10
getInt(n)
print(n)
```



# Behind the Scene.....

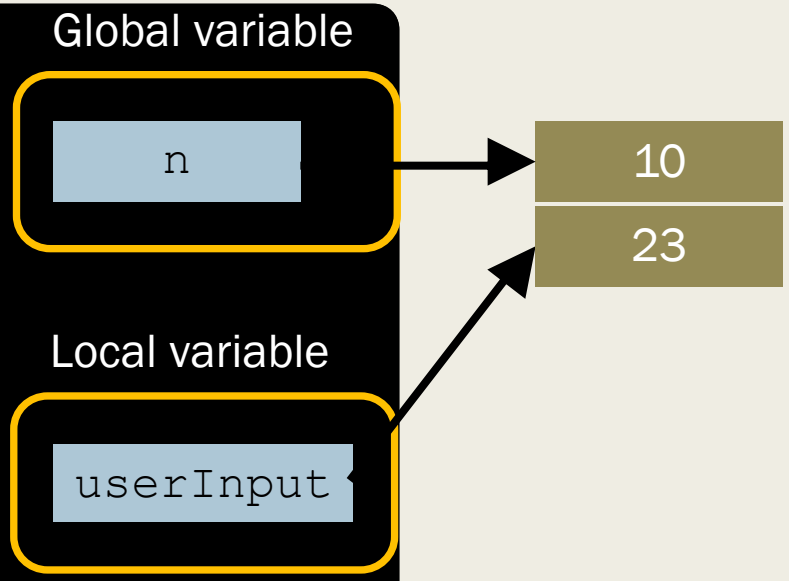
```
def getInt(userInput):
    print("Please enter an integer:")
    userInput=int(input())


n=10
getInt(n)
print(n)
```



# Prove using `id()`

- The built-in function `id(n)` returns the identity of an object referenced by the variable `n`.

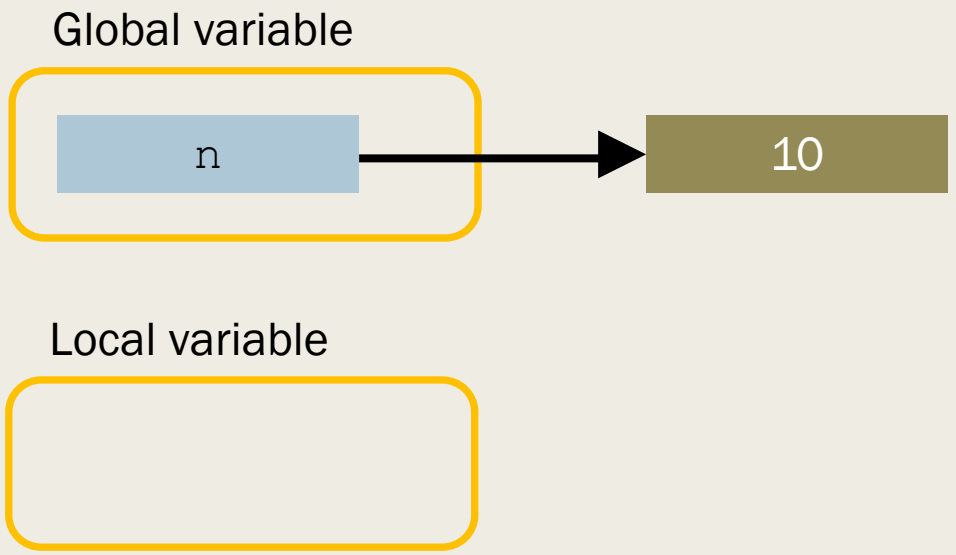
```
def getInt(userInput):
    print("userInput", userInput, id(userInput))
    print("Please enter an integer:")
    userInput=int(input())
    print("userInput", userInput, id(userInput))

n=10
print("n", n, id(n))
getInt(n)
print("n", n, id(n))
```




```
================== RESTART: /Users/csvlee/Documents/temp/test.py ================
n 10 4536004248
userInput 10 4536004248
Please enter an integer:
23
userInput 23 4536004664
n 10 4536004248
```

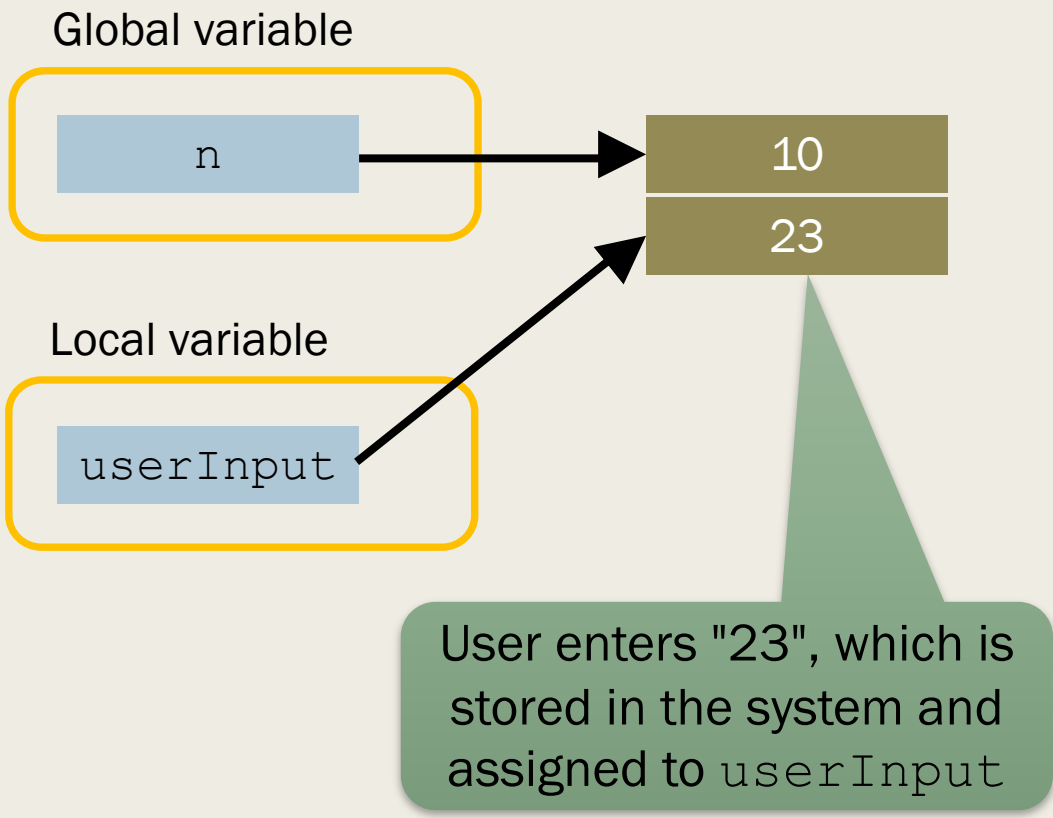
# Another Case

```
def getInt(userInput):
    print("Please enter an integer:")
    userInput=int(input())
    return userInput


n=10
n=getInt(n)
print(n)
```



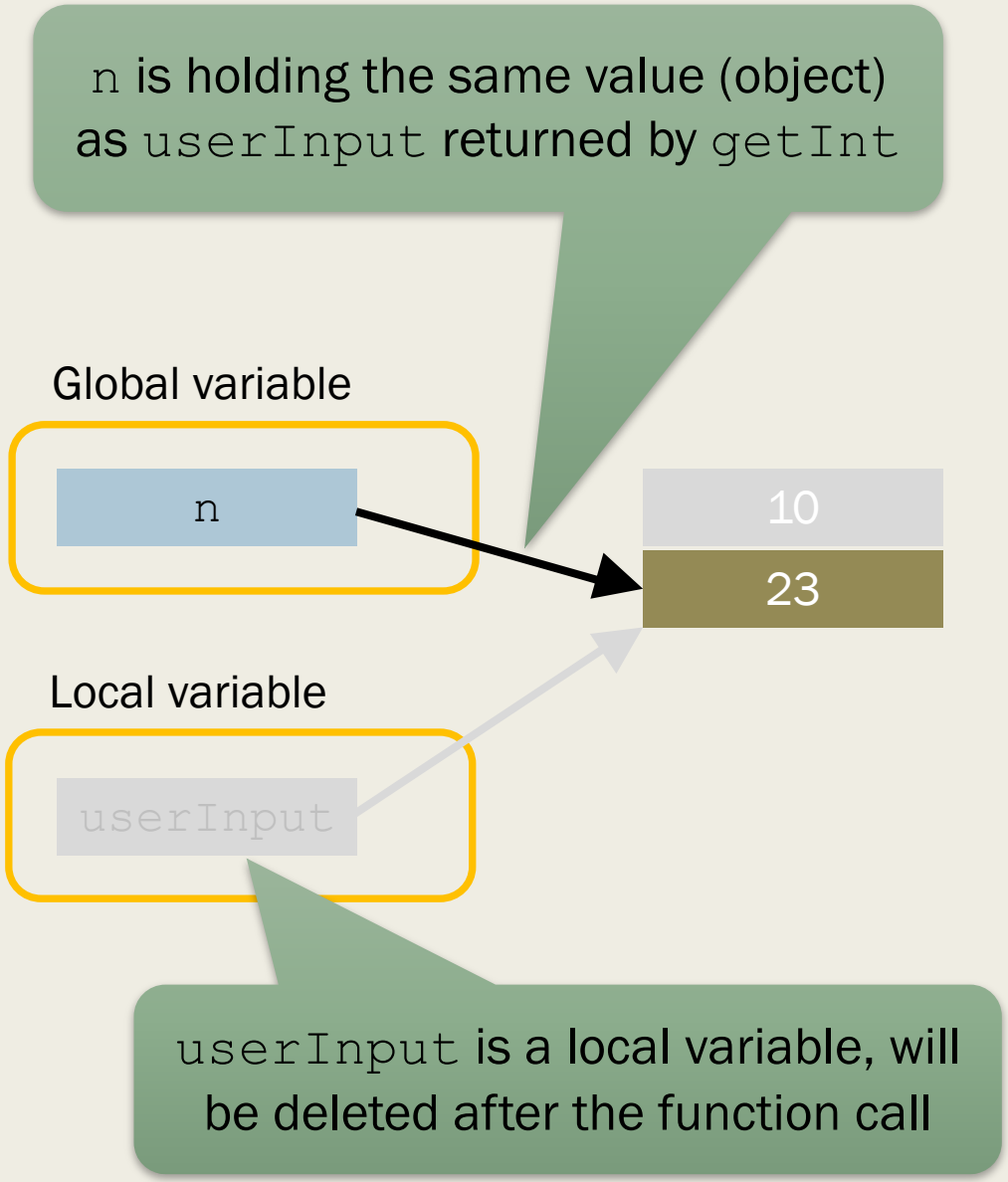
# Another Case

```
def getInt(userInput):
    print("Please enter an integer:")
    userInput=int(input())
    return userInput


n=10
n=getInt(n)
print(n)
```



# Another Case

```
def getInt(userInput):
    print("Please enter an integer:")
    userInput=int(input())
    return userInput


n=10
n=getInt(n)
print(n)
```



# Summary

- Functions help programmer write a simpler program and make the problem easier to solve
- `def function_name(parameter_list):`
- Variable created in function is local variable
- Global variable can only be modified within function with the keyword `global`
- Change of parameter's value within function will not affect the corresponding variable in caller's space (unless the variable is mutable, to be covered later)
- To get the updated value from a function call, use `return`