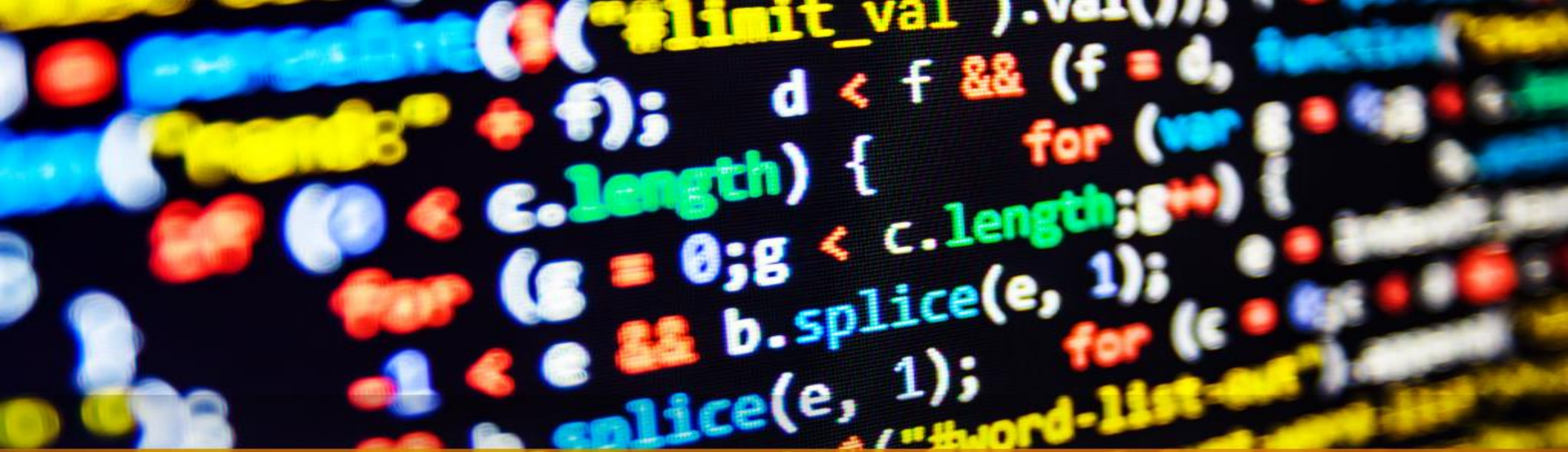


Topic 07

Functions

Prepared by: Suhaini Nordin

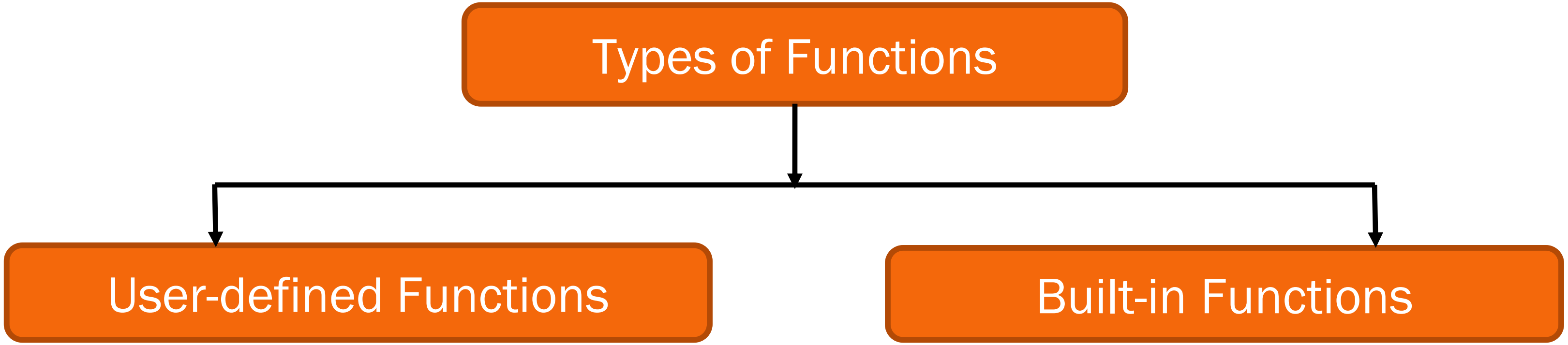
Acknowledgement: Some contents in this notes are edited from original notes prepared by Ban Kar Weng (William)

Objective

1. Explain need for structured programming.
2. Define and call functions
3. Explain use of parameters and return value.
4. Explain difference between local variables and global variables.

What is a Function?

- A named sequence of statements that performs a specific task.



Types of Functions

User-defined Functions

- Written by developers.

Built-in Functions

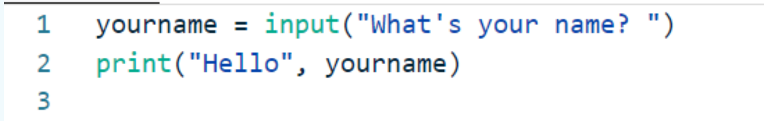
- Provided by the language.
- Developers do not need to define them.

Function

- Books use chapters to separate and group contents. In programming, functions are used to separate and group codes
- A function is a block of codes that performs a specific task and may return value
- Functions improve readability of program
- Function can eliminate the rewriting of identical processes
- Is also commonly referred as module.
  - However, in Python, function and module are two different things

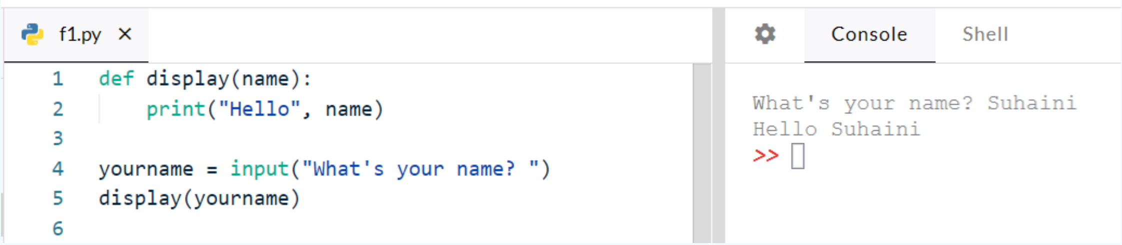
Function vs Module in Python

Without function



```
yourname = input("What's your name? ")
print("Hello", yourname)
```

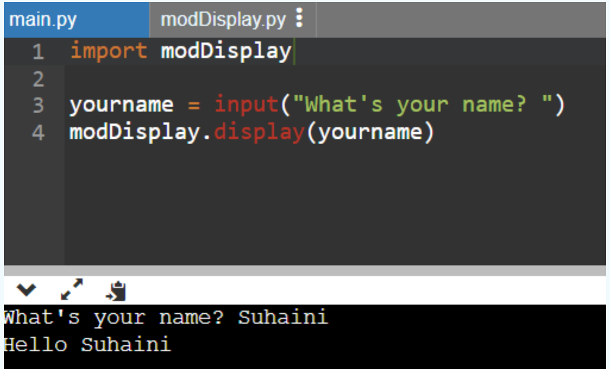
With function



```
def display(name):
    print("Hello", name)

yourname = input("What's your name? ")
display(yourname)
```

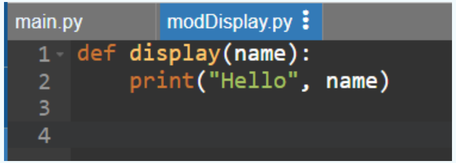
```
What's your name? Suhaini
Hello Suhaini
>> 
```

Module



```
import modDisplay

yourname = input("What's your name? ")
modDisplay.display(yourname)
```

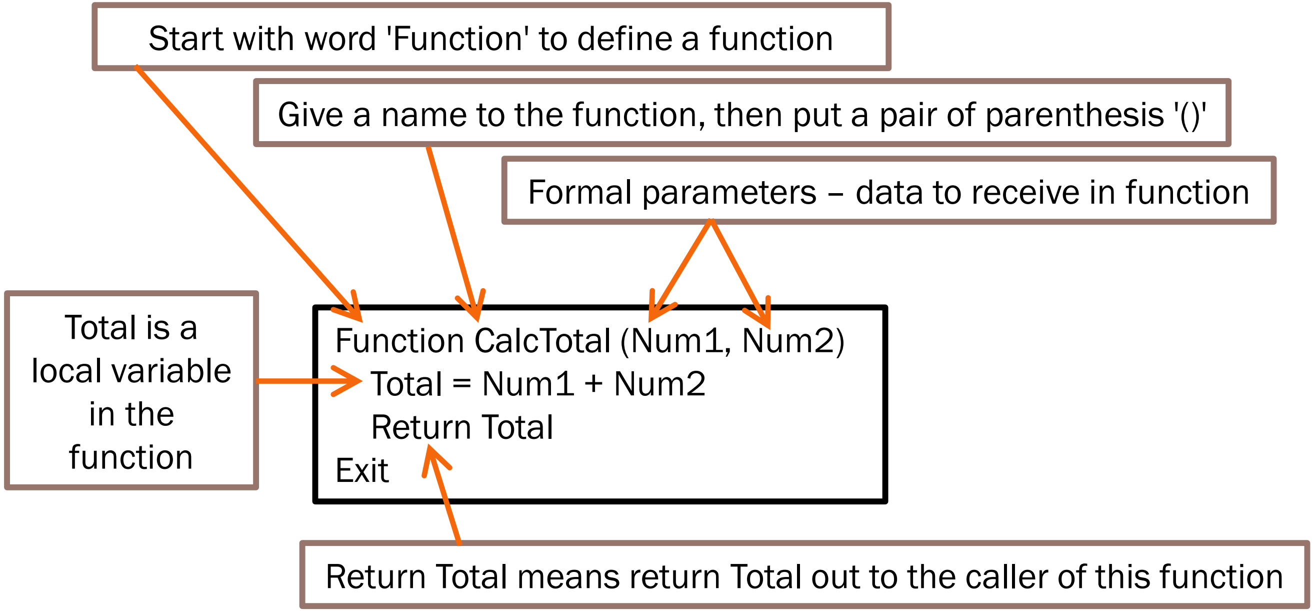
```
What's your name? Suhaini
Hello Suhaini
```



```
def display(name):
    print("Hello", name)
```

Define/Create a Function

- Define a function to find the total of 2 numbers



Start with word 'Function' to define a function

Give a name to the function, then put a pair of parenthesis '()'

Formal parameters – data to receive in function

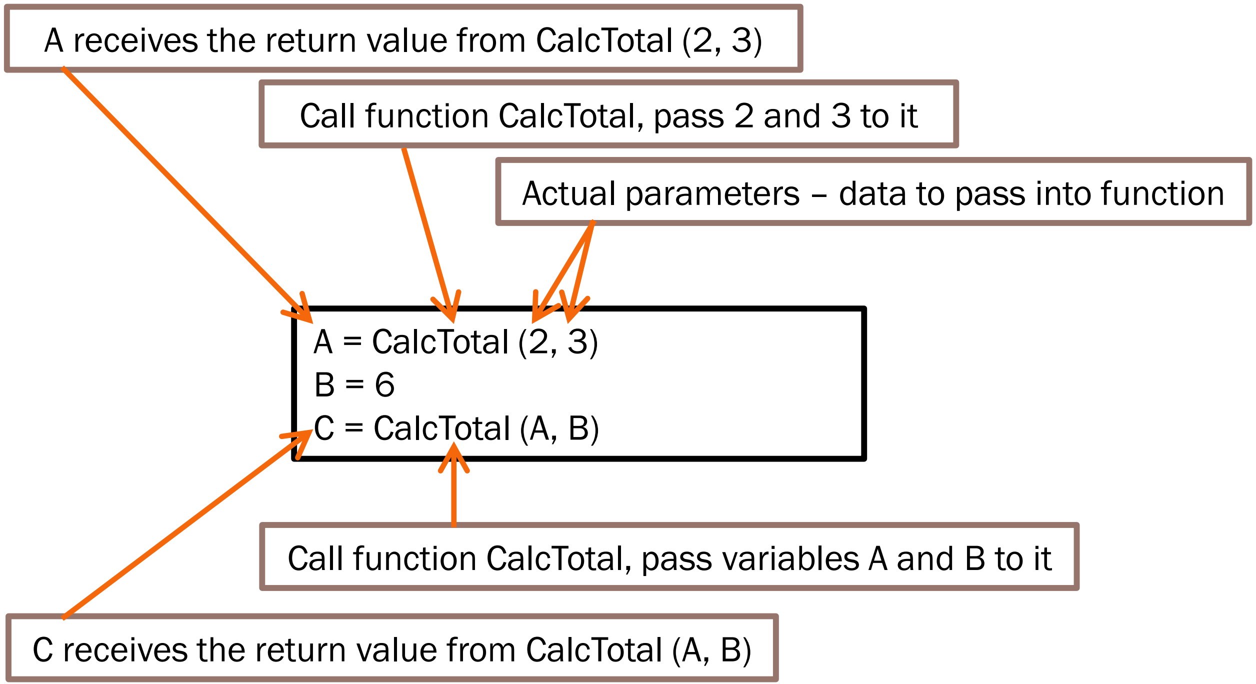
Total is a local variable in the function

```
Function CalcTotal (Num1, Num2)
   Total = Num1 + Num2
   Return Total
Exit
```

Return Total means return Total out to the caller of this function

Call/Use a Function

- To call/use function CalcTotal



A receives the return value from CalcTotal (2, 3)

Call function CalcTotal, pass 2 and 3 to it

Actual parameters – data to pass into function

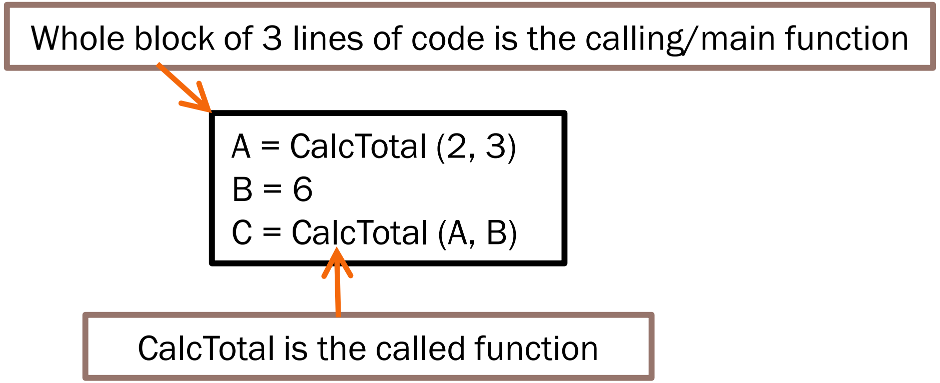
```
A = CalcTotal (2, 3)
B = 6
C = CalcTotal (A, B)
```

Call function CalcTotal, pass variables A and B to it

C receives the return value from CalcTotal (A, B)

Function cont.

- Calling function is the function that calls another function
- Main function is usually referred to as calling function
- Called function is the function being called



Whole block of 3 lines of code is the calling/main function

```
A = CalcTotal (2, 3)
B = 6
C = CalcTotal (A, B)
```

CalcTotal is the called function

Call/Pass by Value

Call by Value
• When a parameter is passed to a function, a copy of actual parameter is passed to formal parameter of the called function.
• Any change made to the formal parameter in the called function have no effect on the values of actual parameter in the calling function.
• To keep any change made to the formal parameter, return the parameter.

```
Function ChangeValue (A, B)
    A = 5
    B = 2
Exit

Start
   A = 4
   B = 9
   ChangeValue (A, B)
   C = A + B
   Print C
End
```

Return Value

```
Function ChangeValue (A, B)
    A = 5
    B = 2
    Return A, B
Exit

Start
   Set A = 4
   Set B = 9
   A, B = ChangeValue (A, B)
   C = A + B
   Print C
End
```

```
Function ChangeValue (A, B)
    A = 5
    B = 2
    Return B
Exit

Start
   Set A = 4
   Set B = 9
   B = ChangeValue (A, B)
   C = A + B
   Print C
End
```

When to Include Parameter?

- Parameters are the data passed to the function in order for the function to perform a calculation or process.
- A function can have any number of parameters depends on its purpose. Separate the parameters by comma ','.
- It is possible for a function to have zero parameter. This happens when the function don't have to receive any data from calling function.

Function with 2 parameters

```
Function CalcTotal (Num1, Num2)
    Total = Num1 + Num2
    Return Total
Exit
```

Function with zero parameter

```
Function Get2Nums()
    Get Num1
    Get Num2
    Return Num1, Num2
Exit
```

When to Include Return?

- If no return statement is written, the function automatically exit when it reaches the end of the function and return nothing to the calling function.

No return statement

```
Function ShowTotal (Total)
   Print Total
Exit
```

When to Include Return? cont.

- An explicit return statement is required when a function has to:
  1. Pass value from local variable or updated parameter out to the calling function, or
  2. Exit the function before the end of function.
- It is possible to return nothing.

Pass local variable

```
Function Get2Nums()
    Get Num1
    Get Num2
    Return Num1, Num2
Exit
```

Pass updated parameter

```
Function Half (Num)
    Num = Num /2
    Return Num
Exit
```

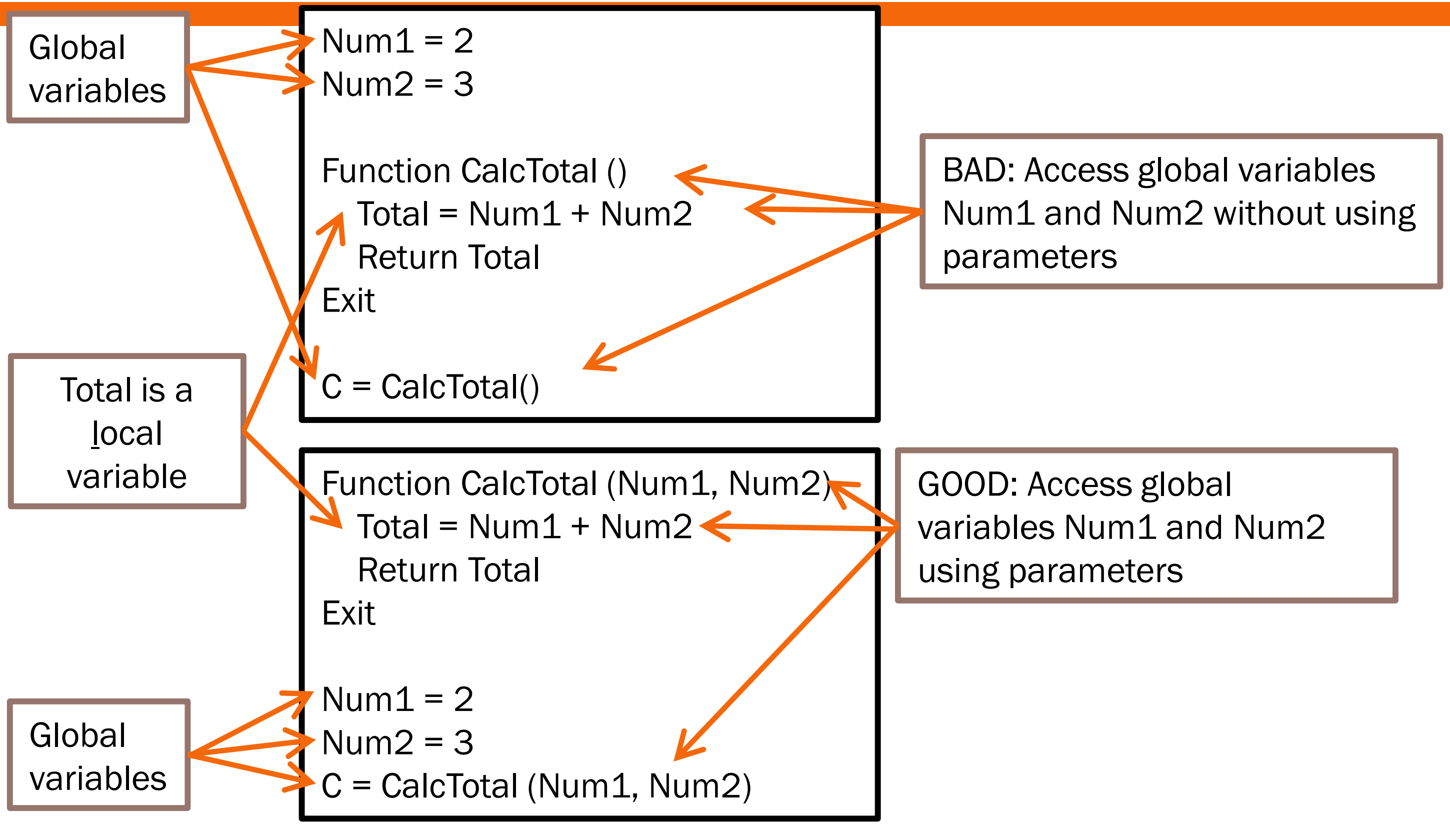
Exit before end

```
Function LongStories ()
    Print "Long story 1"
    Get Continue
    If Continue == "No"
        Return
    Endif
    Print "Long story 2"
Exit
```

Local and Global Variables

- Local variables – variables declared inside a function and may be used only by the function itself.
- Global variables – variables declared in main function, are global to the program and can be seen by all functions.
- Global variables should be passed to functions through parameters.

Local and Global Variables cont.



Global variables

```
Num1 = 2
Num2 = 3

Function CalcTotal ()
    Total = Num1 + Num2
    Return Total
Exit

C = CalcTotal()
```

BAD: Access global variables Num1 and Num2 without using parameters

Total is a local variable

```
Function CalcTotal (Num1, Num2)
    Total = Num1 + Num2
    Return Total
Exit

Num1 = 2
Num2 = 3
C = CalcTotal (Num1, Num2)
```

GOOD: Access global variables Num1 and Num2 using parameters

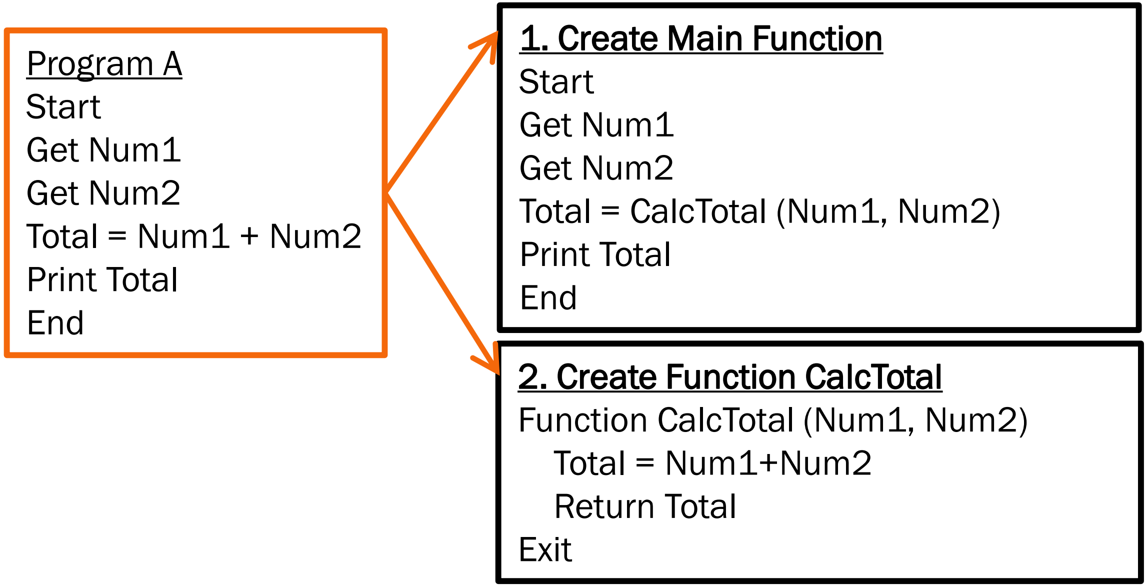
Global variables

Terminology

- Arguments/Parameters: Variables that are passed or called from one function to another.
  - Allows communication between functions
- Calling function: function that calls/processes another function
- Called function: function that is being called/processed
- Actual parameters: parameters that follow the function name being processed in the calling function
- Formal parameters: parameters that follow the function name at the beginning of the function definition
- Return value: The result of a function to return to the calling function

Example 1

- Based on program A, write the new pseudocode that has 2 functions: main function, and function definition named CalcTotal which calculates the total of 2 numbers.



Program A

```
Start
Get Num1
Get Num2
Total = Num1 + Num2
Print Total
End
```

1. Create Main Function

```
Start
Get Num1
Get Num2
Total = CalcTotal (Num1, Num2)
Print Total
End
```

2. Create Function CalcTotal

```
Function CalcTotal (Num1, Num2)
    Total = Num1+Num2
    Return Total
Exit
```

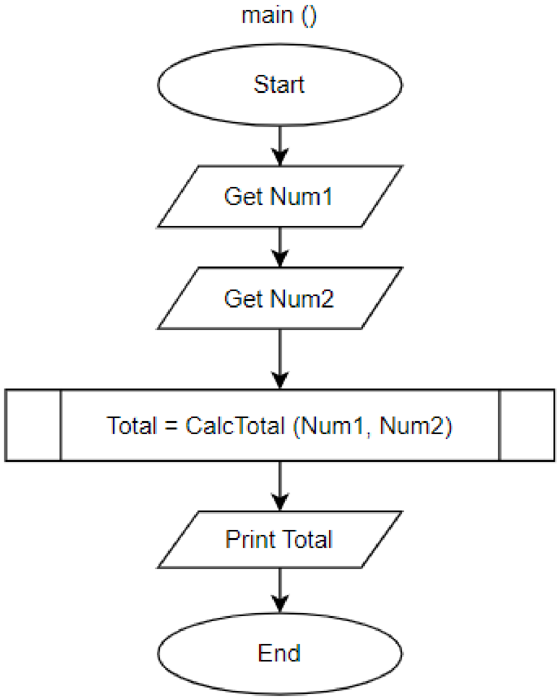
Example 1

1. Create Main Function

```
Start
Get Num1
Get Num2
Total = CalcTotal (Num1, Num2)
Print Total
End
```

2. Create Function CalcTotal

```
Function CalcTotal (Num1, Num2)
    Total = Num1+Num2
    Return Total
Exit
```



main ()

Start

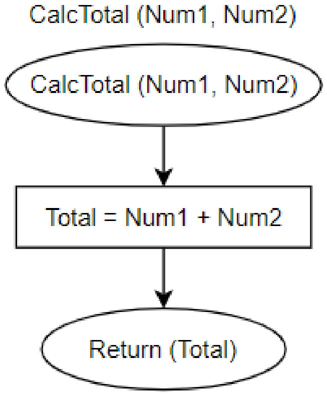
Get Num1

Get Num2

Total = CalcTotal (Num1, Num2)

Print Total

End



CalcTotal (Num1, Num2)

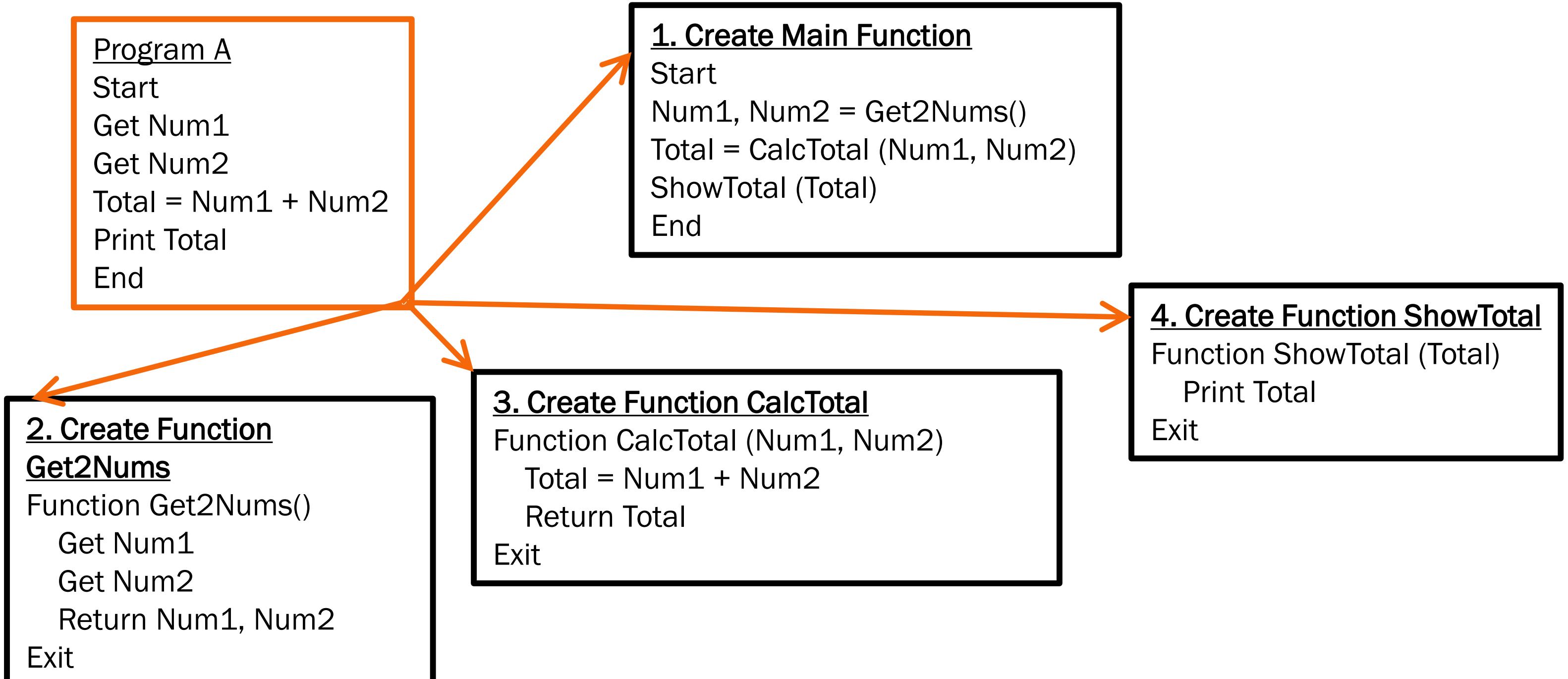
CalcTotal (Num1, Num2)

Total = Num1 + Num2

Return (Total)

Example 2

- Write the pseudocode using 4 functions for a program that accepts 2 numbers and print the total.



Program A

```
Start
Get Num1
Get Num2
Total = Num1 + Num2
Print Total
End
```

1. Create Main Function

```
Start
Num1, Num2 = Get2Nums()
Total = CalcTotal (Num1, Num2)
ShowTotal (Total)
End
```

2. Create Function Get2Nums

```
Function Get2Nums()
    Get Num1
    Get Num2
    Return Num1, Num2
Exit
```

3. Create Function CalcTotal

```
Function CalcTotal (Num1, Num2)
    Total = Num1 + Num2
    Return Total
Exit
```

4. Create Function ShowTotal

```
Function ShowTotal (Total)
    Print Total
Exit
```

Example 2

1. Create Main Function

```
Start
Num1, Num2 = Get2Nums()
Total = CalcTotal (Num1, Num2)
ShowTotal (Total)
End
```

2. Create Function Get2Nums

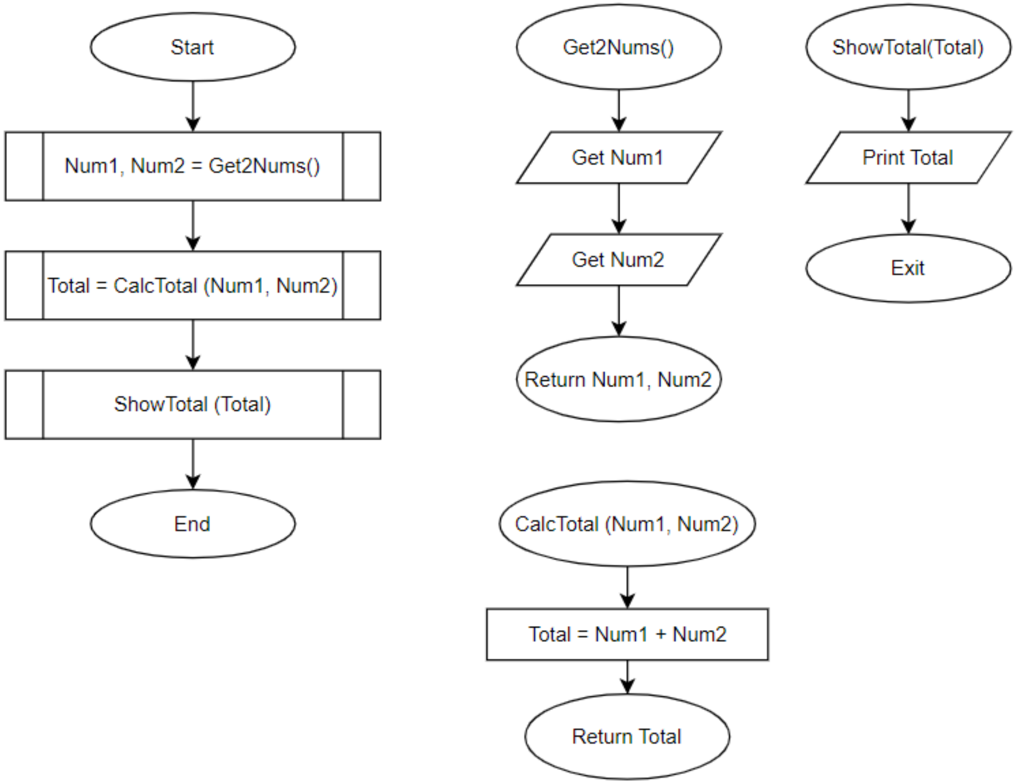
```
Function Get2Nums()
   Get Num1
   Get Num2
   Return Num1, Num2
Exit
```

3. Create Function CalcTotal

```
Function CalcTotal (Num1, Num2)
   Total = Num1 + Num2
   Return Total
Exit
```

4. Create Function ShowTotal

```
Function ShowTotal (Total)
   Print Total
Exit
```



Exercise

Write a pseudocode for a program that calculates the area of a circle when the user input is the radius. Your program shall define and call the following function:

1. A function definition that calculates the area of a circle.

Defining Functions

Defining Functions

```
def function_name(parameter list):
    statement(s)
    return statement
```

- The keyword **def** indicates the start of a function definition.
- The function name must follow variable naming rules.
- The parameter list allow data to be passed into the function. These are **optional**.
- One or more Python statements make up the function body, indented relative to function definition.
- The return statement allow data to be passed out of the function. This is **optional**.

Defining Functions

```
def print_msg():
    print(“Hello World”)
```

Example 1:

- Function name: `print_msg`
- No parameters.
- Has only a single statement in the function body.
- No return statement.

Defining Functions

```
def print_msg(times):
    print(“Hello World” * times)
```

Example 2:

- Function name: `print_msg`
- One parameter: `times`
- Has only a single statement in the function body.
- No return statement.

Defining Functions

```
def print_msg(msg, times):
    print(msg * times)
```

Example 3:

- Function name: `print_msg`
- Two parameters: `msg` and `times`
- Has only a single statement in the function body.
- No return statement.

Defining Functions

```
def secret_msg():
    return “There is no secret here”
```

Example 4:

- Function name: `secret_msg`
- No parameters.
- Has a **return statement** in the function body.

Defining Functions

```
def secret_msg():
    secret = “There is no secret here”
    return secret
```

Example 5:

- Function name: `secret_msg`
- No parameters.
- Has two statements in the function body.
- The last statement is a **return statement**.

Defining Functions

```
def square(n):
    return n ** 2
```

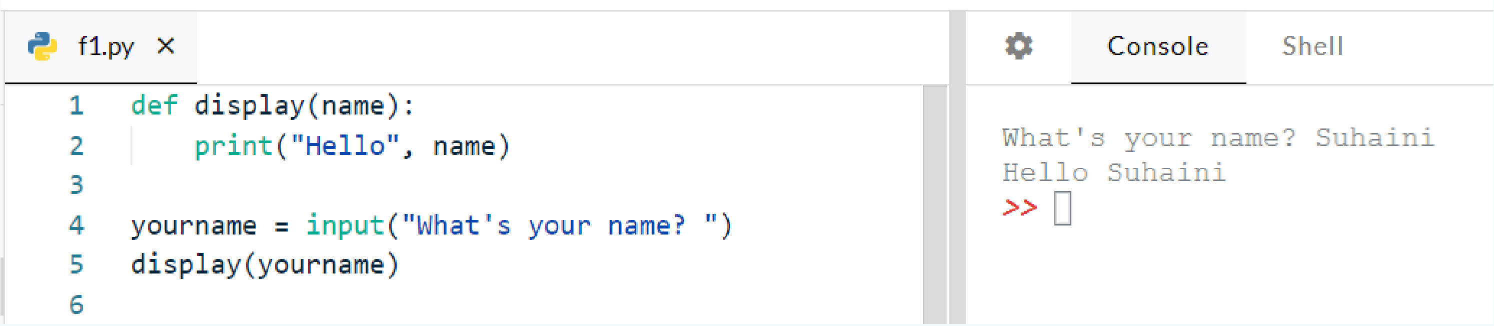
Example 6:

- Function name: `square`
- One parameter: `n`
- Has a **return statement** in the function body.

Calling Functions

Calling Functions

- The statements inside a function are not executed when the function is defined.
- To execute the statements inside a function, we need to **call the function**.

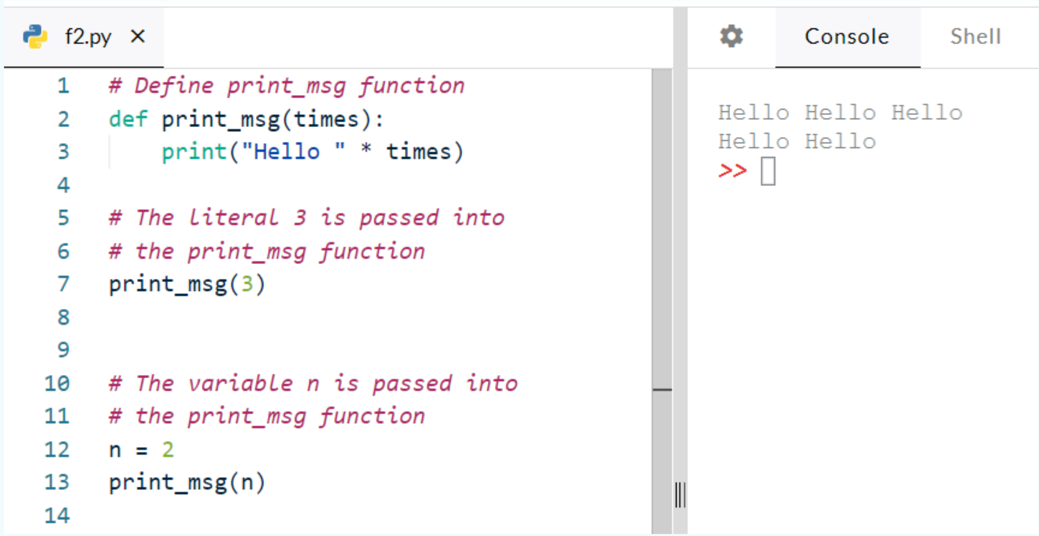


```python
def display(name):
    print("Hello", name)

yourname = input("What's your name? ")
display(yourname)

```

```
What's your name? Suhaini
Hello Suhaini
>>
```

Calling Functions

- When passing data to a function with parameter(s), the data can be **literals** or **value stored in a variable**.
- The variable passed in **need not be the same name** as the function parameter name.

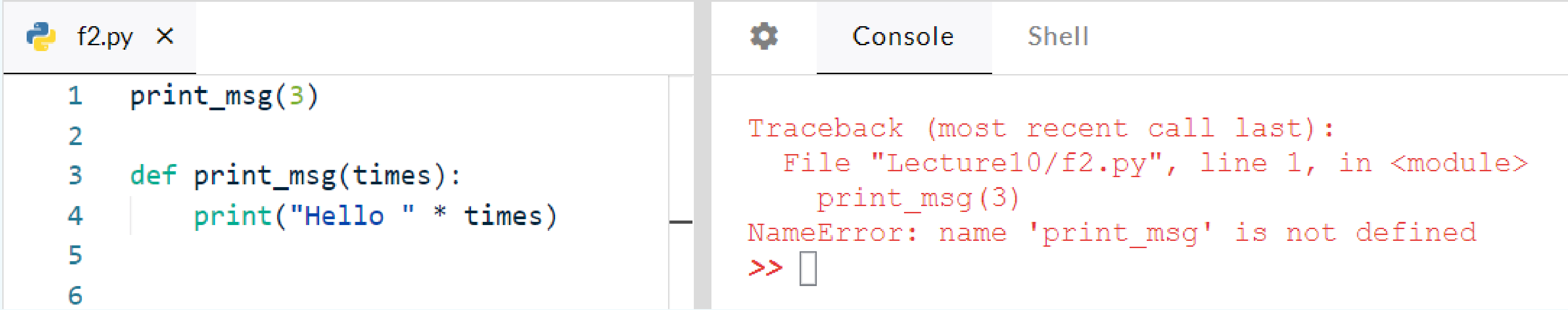


```python
# Define print_msg function
def print_msg(times):
    print("Hello " * times)

# The literal 3 is passed into
# the print_msg function
print_msg(3)


# The variable n is passed into
# the print_msg function
n = 2
print_msg(n)

```

```
Hello Hello Hello
Hello Hello
>>
```

Calling Functions

- Function must be defined **before the first time it is called**.



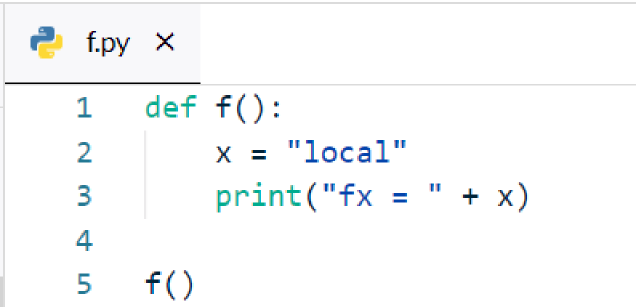
```python
print_msg(3)

def print_msg(times):
    print("Hello " * times)


```

```
Traceback (most recent call last):
  File "Lecture10/f2.py", line 1, in <module>
    print_msg(3)
NameError: name 'print_msg' is not defined
>> 
```

Variable Scope and Lifetime

Variable Scope and Lifetime

Can you guess the output of this program?

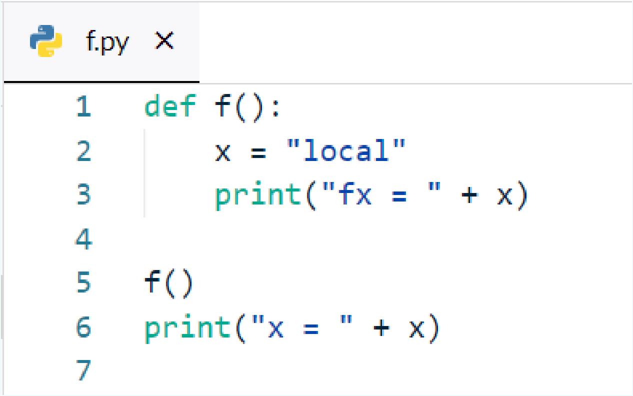


```python
def f():
    x = "local"
    print("fx = " + x)

f()
```

A	fx = local
B	fx = x
C	NameError: name 'x' is not defined
D	NameError: name 'f' is not defined

Variable Scope and Lifetime

Can you guess the output of this program?



```python
def f():
    x = "local"
    print("fx = " + x)

f()
print("x = " + x)
```

A	`fx = local`
	`x = local`

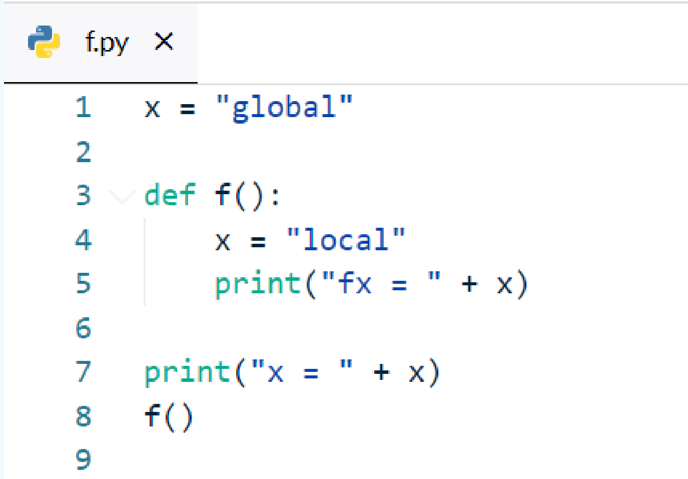
B	`NameError: name 'f' is not defined`
	`NameError: name 'x' is not defined`

C	`fx = local`
	`NameError: name 'x' is not defined`

D	`x = local`
	`fx = local`

Variable Scope and Lifetime

Can you guess the output of this program?



```python
x = "global"

def f():
    x = "local"
    print("fx = " + x)

print("x = " + x)
f()

```

A	fx = local
	x = global

B	NameError: name 'f' is not defined NameError: name 'x' is not defined
C	fx = local NameError: name 'x' is not defined

D	x = global
	fx = local

Variable Scope and Lifetime

In Python:

- We create variables using assignment statements.
- Variables created outside of any function is called ***global variables***.
- In the codes below, the variable x is a global variable.

Python Code:

```
x = 1
```

Python Code:

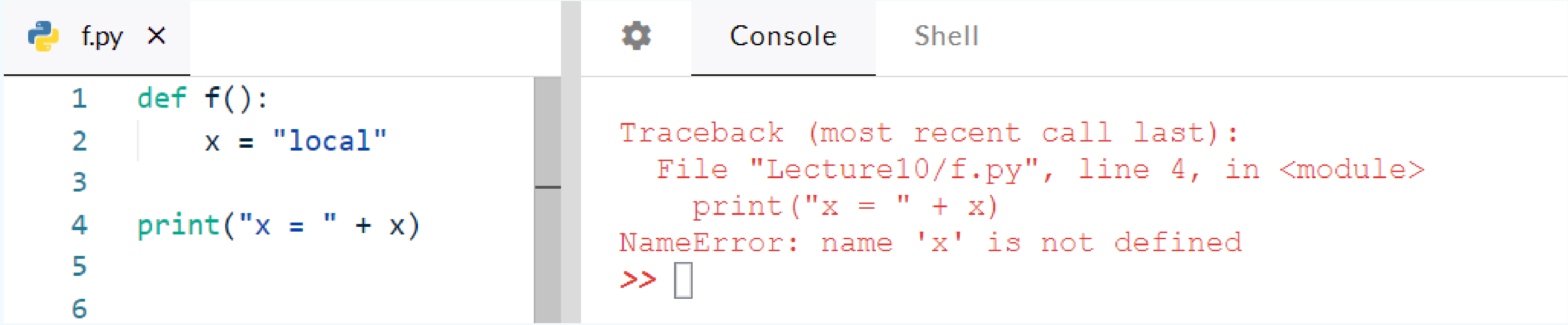
```
if True:
    x = 1
```

Python Code:

```
while True:
    x = 1
```

Variable Scope and Lifetime

- Variables created within a function are called ***local variables***:
- Local variables are:
  - Scoped only to that function
  - Not accessible outside of that function.



```python
def f():
    x = "local"

print("x = " + x)
```

```
Traceback (most recent call last):
  File "Lecture10/f.py", line 4, in <module>
    print("x = " + x)
NameError: name 'x' is not defined
>> 
```

The **global** keyword

Attempting to **access** the value of a global variable within a function.

Python Code:

```
x = “global”

def f():
    x = “local”
    print(x)

f()
```

Output:

```
local
```

The **global** keyword

Attempting to **change** the value of a global variable within a function.

Python Code:

```
x = “global”

def f():
    x = x + “local”
    print(x)

f()
```

Output:

```
Traceback (most recent call last):
  File "def.py", line 7, in <module>
    f()
  File "def.py", line 4, in f
    x = x + 'local'
UnboundLocalError: local variable
'x' referenced before assignment
```

The **global** keyword

To tell Python we want to reference a global variable, we need to use the keyword **global** with the name of the variable.

Python Code:

```
x = “global”

def f():
    global x
    x = x + “local”
    print(x)

f()
```

Output:

```
globallocal
```

Default Argument

Parameter v.s. Argument

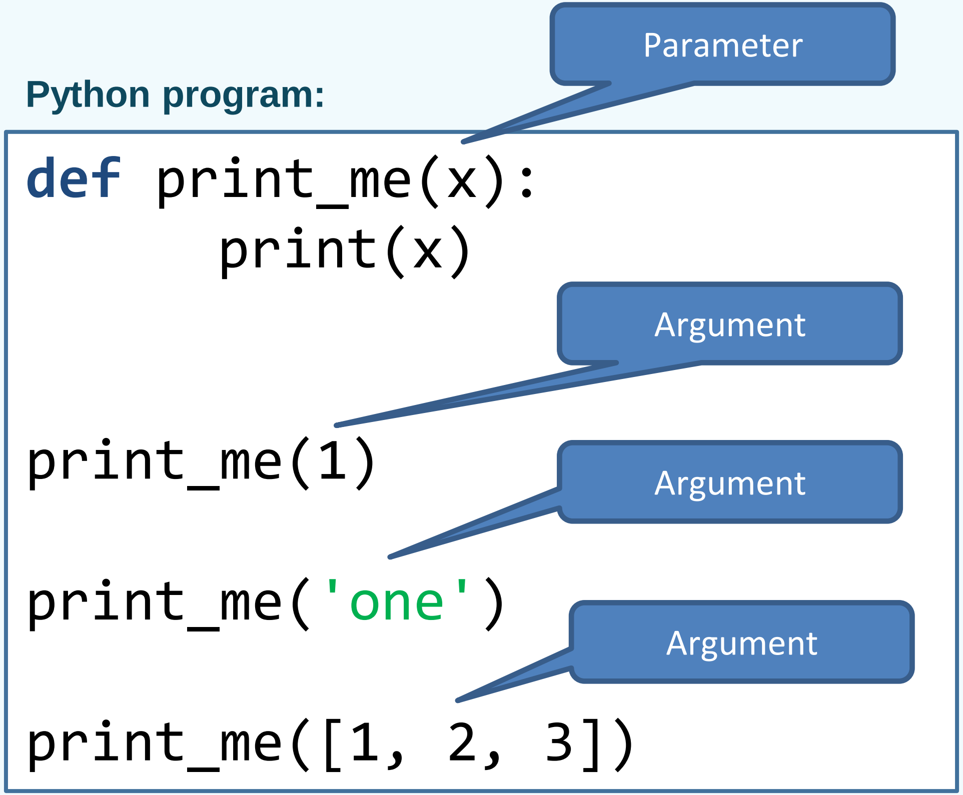
Parameter:

- A variable defined by a function that receives a value when the function is called.

Argument:

- A value passed to a function when it is involved.

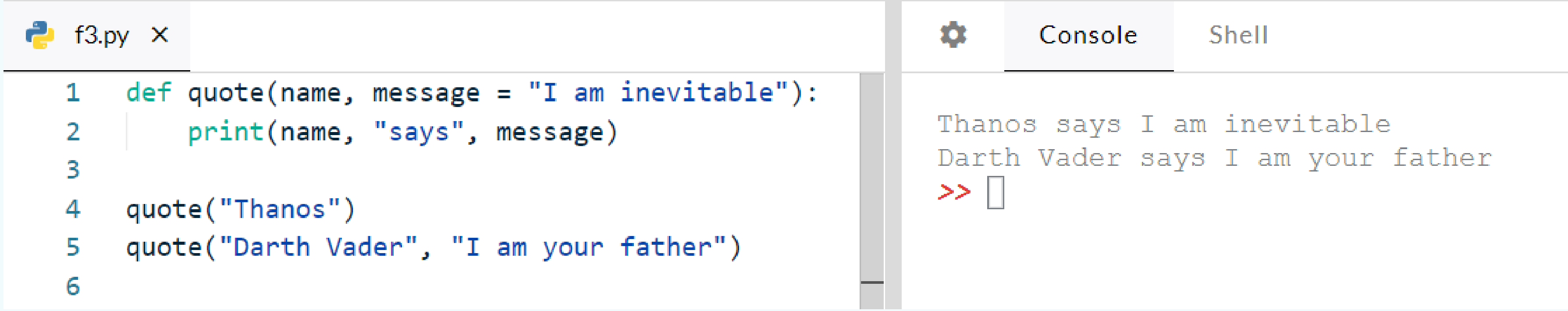
Python program:



```
def print_me(x):
      print(x)


print_me(1)

print_me('one')

print_me([1, 2, 3])
```

Default Argument

- Function parameters can be set with default arguments.
- The default value is declared in the function header along with the parameter name.
- If a value is supplied for the parameter, it will override the default.
- If no value is supplied when the function is called, the default value will be used.



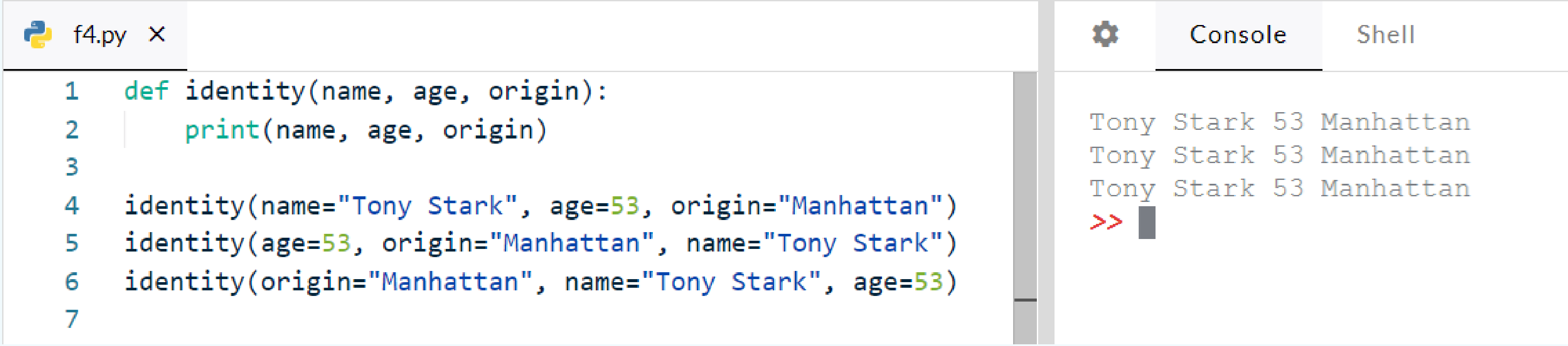
```python
def quote(name, message = "I am inevitable"):
    print(name, "says", message)

quote("Thanos")
quote("Darth Vader", "I am your father")
```

```
Thanos says I am inevitable
Darth Vader says I am your father
>>
```

Named Argument

Named Argument

- So far, the position of an argument in a function is used to determine which parameter that argument is assigned to.
- Using named argument, we can associate an argument with the parameter’s name instead of position.



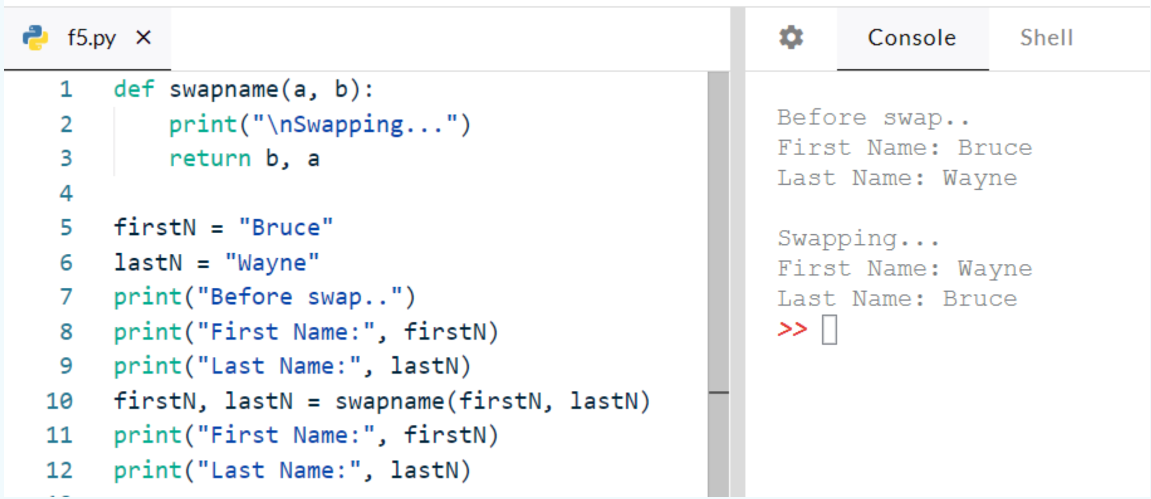
```python
def identity(name, age, origin):
    print(name, age, origin)

identity(name="Tony Stark", age=53, origin="Manhattan")
identity(age=53, origin="Manhattan", name="Tony Stark")
identity(origin="Manhattan", name="Tony Stark", age=53)

```

```
Tony Stark 53 Manhattan
Tony Stark 53 Manhattan
Tony Stark 53 Manhattan
>>
```

Returning Multiple Values

Returning Multiple Values

- It is possible to return multiple values from a function.
- In this swapname() function, the order in which the parameters are supplied is swapped when they are returned.
- When swapname() function is called, the returned values are assigned to variables.



```python
def swapname(a, b):
    print("\nSwapping...")
    return b, a

firstN = "Bruce"
lastN = "Wayne"
print("Before swap..")
print("First Name:", firstN)
print("Last Name:", lastN)
firstN, lastN = swapname(firstN, lastN)
print("First Name:", firstN)
print("Last Name:", lastN)
```

```
Before swap..
First Name: Bruce
Last Name: Wayne

Swapping...
First Name: Wayne
Last Name: Bruce
>> 
```