UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FAKULTAS
ILMU
KOMPUTER

Topic 6.1:
Function

CSGE601020 - Dasar-Dasar Pemrograman 1

Lintang Matahari Hasani, S.Kom., M.Kom. | Dr.Eng. Lia Sadita, S.Kom., M.Eng.

Acknowledgement

This slide is an adapted version of **Function** slides used in DDP1 Course (2020/2021) by Hafizh Rafizal Adnan, M.Kom.

Some of the design assets used in these slides were provided by ManyPixels under an nonexclusive, worldwide copyright license to download, copy, modify, distribute, perform, and use the assets provided from ManyPixels for free, including for commercial purposes, without permission from or attributing the creator or ManyPixels.

Some additional contents, illustrations and visual design elements are provided by
Lintang Matahari Hasani, M.Kom. *(lintang.matahari01[at]cs.ui.ac.id)*

In this session, you will learn ...

- What is Function
- Python Function
- Function Flow
- Parameter Passing
- Variables and Scope
- Best Practice

Revisiting the Concept of Mathematical Function

$$\underbrace{\texttt{z}}_{\text{The output}} = \texttt{f}(\underbrace{\texttt{x},\texttt{y}}_{\text{The input}})$$

$$f(x) = x^2$$
$$f(1) = 1$$
$$f(2) = 4$$
$$f(3) = 9$$
$$f(4) = 16$$

and so on...

`f` function operates on the inputs of `x` and `y` **to produce `z`**

- ➔ From Mathematics we know that functions perform some operation and return one value
- ➔ **In programming functions are much more generalized and versatile** than this mathematical definition of function
- ➔ In programming, a function is a self-contained block of code that **encapsulates a specific task or a group of tasks**

https://realpython.com/defining-your-own-python-function/

We have used Python Function Before...

```
print('DDP1 Mantap gan!')
```

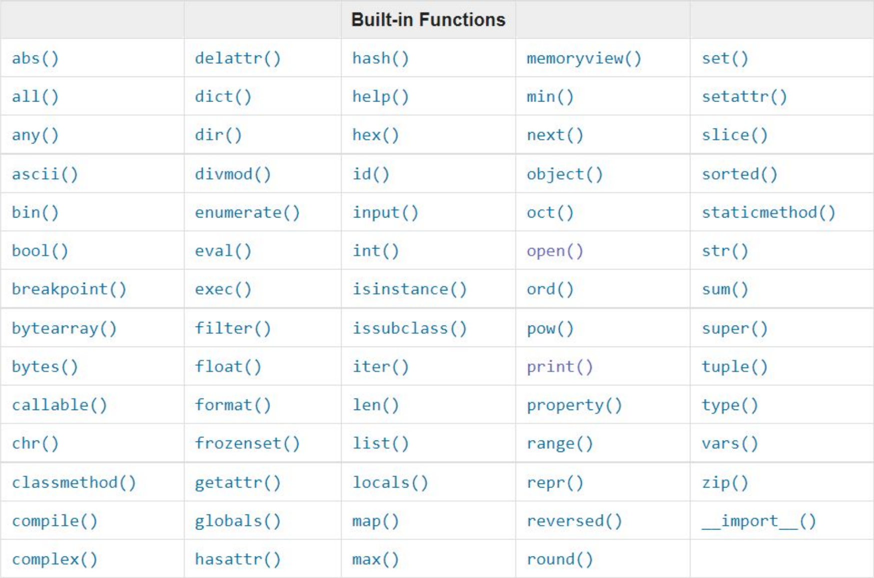
```
a = 'PLUS ULTRA!'
len(a)
```

```
round(12.9231)
```

`print()`, `len()`, and `round()` are Python **Built-In** Functions

Python Built-In Function

https://docs.python.org/3/library/functions.html



		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

```
print('Mantappu Jiwaa!')
```

```
bool('')
```

```
int('123')
```

```
len('Boku no Hero Academia')
```

```
ord('寝')
```

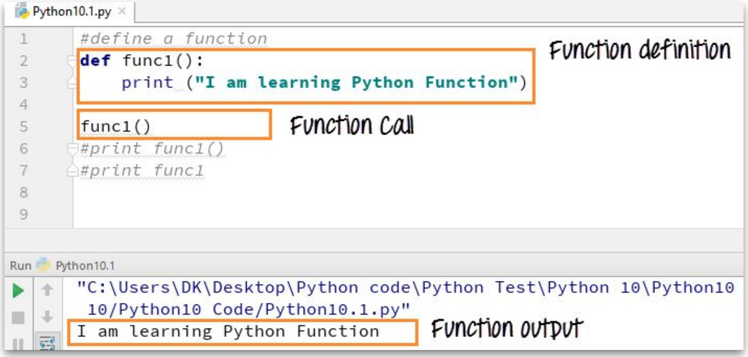
```
chr(65)
```

We can define our own function!

What Is Function?

A function is a **set of instructions** you can call to carry out a specific task

Function "**encapsulate**" the performance of some particular operation, so it can be used by others



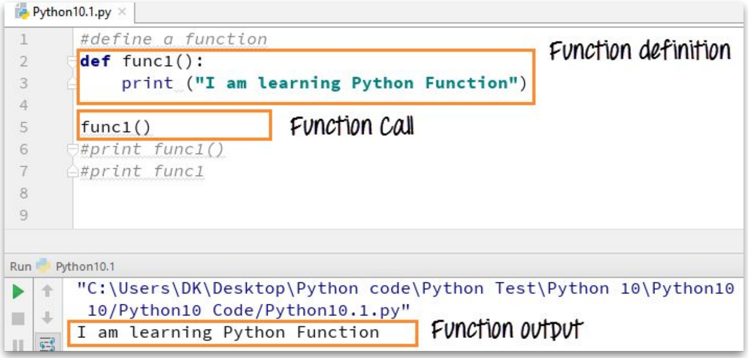
```
#define a function
def func1():
    print ("I am learning Python Function")

func1()
#print func1()
#print func1
```

```
"C:\Users\DK\Desktop\Python code\Python Test\Python 10\Python10
 10/Python10 Code/Python10.1.py"
I am learning Python Function
```

https://www.guru99.com/functions-in-python.html

Why is Function Important

- ➔ **Reusability**. Once written, use again
- ➔ **Sharing**. If tested, others can use
- ➔ **Security**. Well tested, then secure for reuse
- ➔ **Simplify code**.
- ➔ **More readable**.
- ➔ Support divide-and-conquer strategy (will learn this more in recursion topic): Made our **problem solving easier** (solved smaller problems as functions)
- ➔ **Abstraction** of an operation (will learn more in OOP topic)

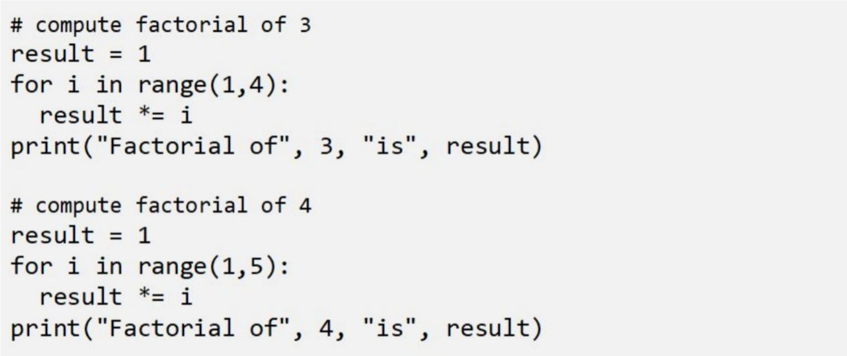


```python
#define a function
def func1():
    print ("I am learning Python Function")

func1()
#print func1()
#print func1
```

```
"C:\Users\DK\Desktop\Python code\Python Test\Python 10\Python10
 10/Python10 Code/Python10.1.py"
I am learning Python Function
```

https://www.guru99.com/functions-in-python.html

Programming Without Functions?



```
# compute factorial of 3
result = 1
for i in range(1,4):
  result *= i
print("Factorial of", 3, "is", result)

# compute factorial of 4
result = 1
for i in range(1,5):
  result *= i
print("Factorial of", 4, "is", result)
```

$$2! = (1! + 0!) \times 1$$
$$3! = (2! + 1!) \times 2$$
$$4! = (3! + 2!) \times 3$$
$$5! = (4! + 3!) \times 4$$
$$6! = (5! + 4!) \times 5$$
$$7! = (6! + 5!) \times 6$$
$$8! = (7! + 6!) \times 7$$
$$9! = (8! + 7!) \times 8$$
$$10! = (9! + 8!) \times 9$$
$$\cdots$$
$$n! = ((n-1)! + (n-2)!) \times (n-1)$$

and so on...

Imagine if you are going to calculate the factorial of all number in range 1-100.
How many lines of code you need?

Surely it will not be efficient and hard to read

Image: https://archimedes-lab.org/2018/05/23/magic-factorials/

Programming With Functions

```
def factorial(n):
      result = 1
      for i in range(1, n+1):
            result *= i

      return result

for i in range(1,10):
      print('Factorial of', i, 'is', factorial(i))
```

With **shorter code**, we can calculate the factorial of all number in range 1-10.

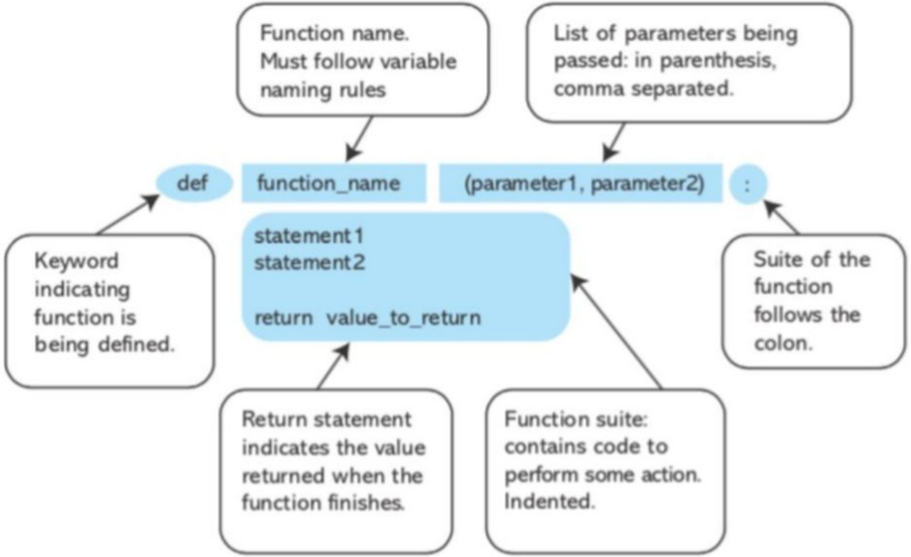
Triggering Question 1

Guess the output...

```
def mystery_1(x, y, z):
      return x + y + z

a, b, c = 10, 20, 30
print(mystery_1(a, b, c))

def mystery_2(x):
      w = x * 3
      return w

a = 'scoobie doobie doo~ '
print(mystery_2(a))
```

Write the answer in the **comment section**

The Anatomy of a Function



Function name. Must follow variable naming rules

List of parameters being passed: in parenthesis, comma separated.

Keyword indicating function is being defined.

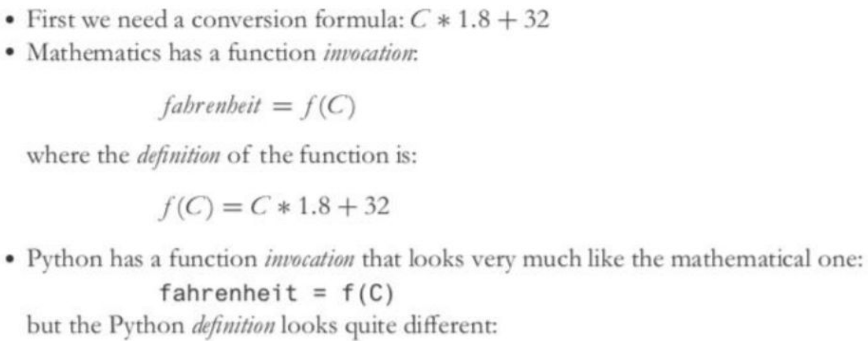
Suite of the function follows the colon.

Return statement indicates the value returned when the function finishes.

Function suite: contains code to perform some action. Indented.

FIGURE 5.1 Function parts.

Example: Converting Celsius temperatures to Fahrenheit



- First we need a conversion formula: $C * 1.8 + 32$
- Mathematics has a function *invocation*:

$$fahrenheit = f(C)$$

where the *definition* of the function is:

$$f(C) = C * 1.8 + 32$$

- Python has a function *invocation* that looks very much like the mathematical one:

```
fahrenheit = f(C)
```

but the Python *definition* looks quite different:

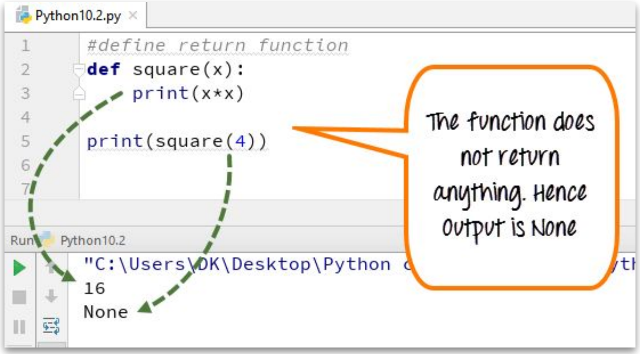
```
def celsius_to_fahrenheit(celsius_float):
      return celsius_float * 1.8 + 32
```

- ➔ C is called an argument of the function.
- ➔ The celsius_float is termed a parameter of the function.

`return` Statement

The `return` statement indicates **the value that is returned** by the function

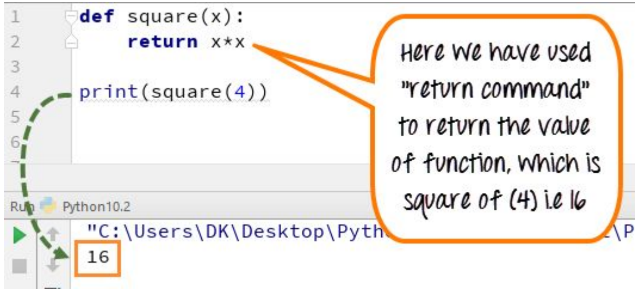
The statement is **optional** (the function can return nothing). If no `return` is stated inside, a function is often called a procedure



```python
#define return function
def square(x):
    print(x*x)

print(square(4))
```

```
"C:\Users\DK\Desktop\Python c
16
None
```

The function does not return anything. Hence output is None



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

print(square(4))
```

```
"C:\Users\DK\Desktop\Pyth
16
```

Here we have used "return command" to return the value of function, which is square of (4) i.e 16

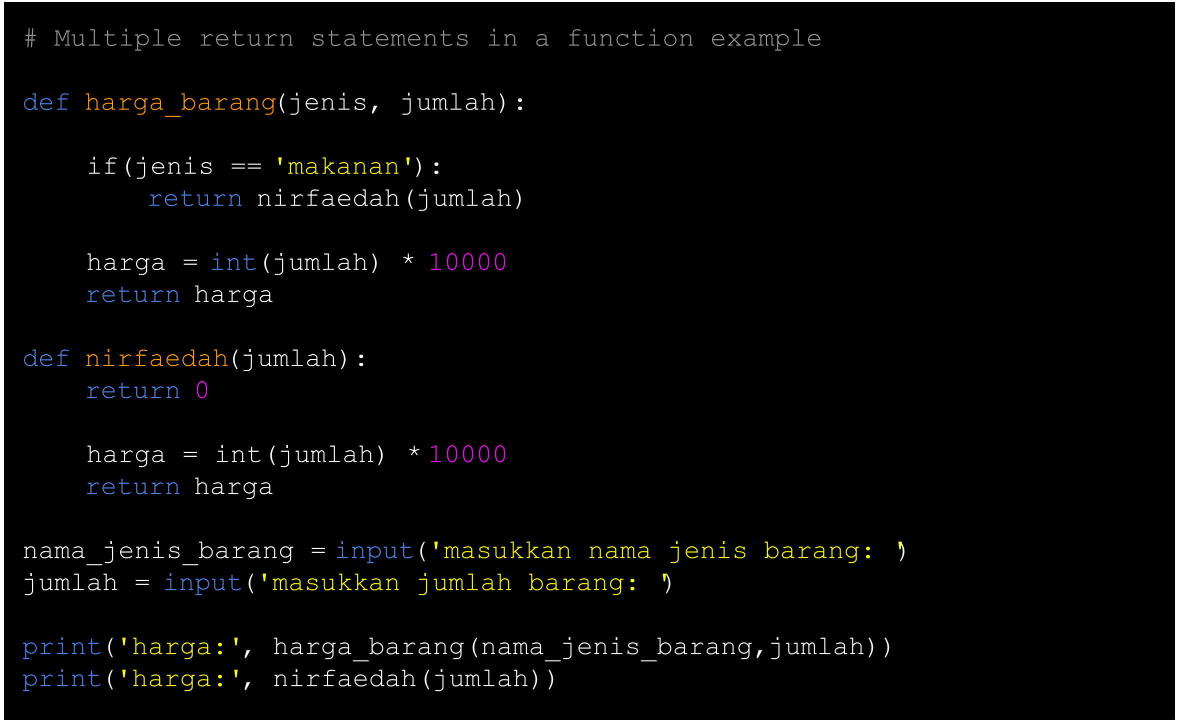
https://www.guru99.com/functions-in-python.html

Multiple `return` Statement

A function can have **multiple** `return` **statements**.

Remember, when the **first** `return` **statement** executed, it **ends** the function.

Multiple `return` can be confusing to the reader and should be used judiciously.



```
# Multiple return statements in a function example

def harga_barang(jenis, jumlah):

    if(jenis == 'makanan'):
        return nirfaedah(jumlah)

    harga = int(jumlah) * 10000
    return harga

def nirfaedah(jumlah):
    return 0

    harga = int(jumlah) *10000
    return harga

nama_jenis_barang = input('masukkan nama jenis barang: ')
jumlah = input('masukkan jumlah barang: ')

print('harga:', harga_barang(nama_jenis_barang,jumlah))
print('harga:', nirfaedah(jumlah))
```

Procedures

Functions that have **no `return` statements** are often called procedures.

Procedures are used to perform some duty (print output, store a file, etc.)

Remember, `return` is not required.

```
def sapa(nama):
      print("Selamat pagi,", nama + "!")

sapa("Midoriya-shounen")
print(sapa("Midoriya-shounen"))
```

Triple Quoted String in Function

A triple quoted string just after the `def` is called a **docstring**

docstring is **documentation of the function's purpose**, to be used by other tools to tell the user what the function is used for.

```
def celsius_to_fahrenheit(celsius):
      '''The function celsius_to_fahrenheit(celsius) takes a
      float (degree Celsius) and converts it to Fahrenheit'''

      return celsius * 1.8 + 32

# print the returned value of celsius_to_fahrenheit function
print(celsius_to_fahrenheit(30.8))

# print the docstring of celsius_to_fahrenheit function
print(celsius_to_fahrenheit.__doc__)
```

Triggering Question 2

Guess the output...

```
a, b, c = 10, 20, 30
print(mystery_1(a, b, c))

def mystery_1(x, y, z):
      return x + y + z
```

Write the answer in the **comment section**

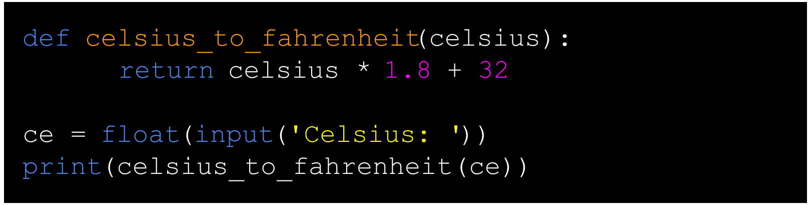
How Function Works (1)

```
ce = float(input('Celsius: '))
far = celsius_to_fahrenheit(ce)
print(far)

def celsius_to_fahrenheit(celsius):
      return celsius * 1.8 + 32
```

```
def celsius_to_fahrenheit(celsius):
      return celsius * 1.8 + 32

ce = float(input('Celsius: '))
far = celsius_to_fahrenheit(ce)
print(far)
```

How Function Works (2): Flow of Control

- ➔ For every program, there is usually one **“main” part** where execution begins.
- ➔ After that the flow of control is **based on the order** of both statements and functions.
- ➔ For functions, operation of a function is determined by **when it is invoked**, not when it is defined
- ➔ Functions can be defined anywhere in the program file, as long as they are **defined before they are invoked.**
- ➔ Functions **must be defined before use** because the function name must be placed in the namespace before it can be called.



```
def celsius_to_fahrenheit(celsius):
      return celsius * 1.8 + 32

ce = float(input('Celsius: '))
print(celsius_to_fahrenheit(ce))
```

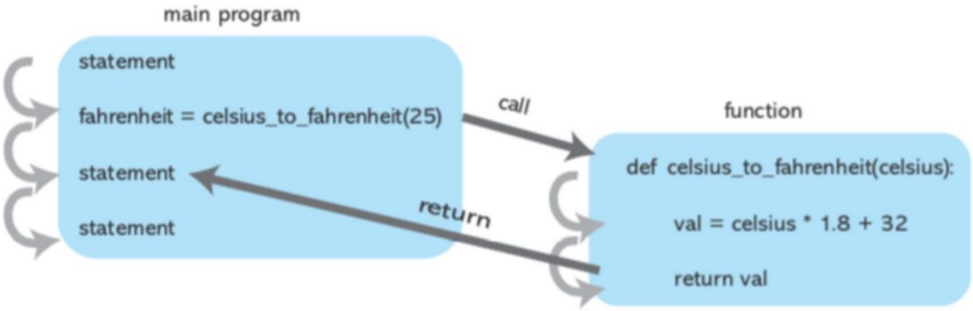
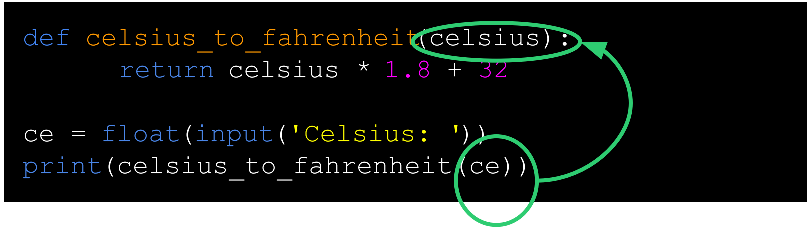


FIGURE 5.2 Function flow of control.

Parameter Passing

- Parameter passing is the **passing of values** from a calling program **to a function**, so that the function can perform its operation
- Parameter passing is the passing of values **from argument to parameter**
- Argument values are typically passed to parameter names in the order they are listed.
- The names of the corresponding argument and parameter need not match.
- **The number of arguments and parameters must match.**



```
def celsius_to_fahrenheit(celsius):
        return celsius * 1.8 + 32

ce = float(input('Celsius: '))
print(celsius_to_fahrenheit(ce))
```

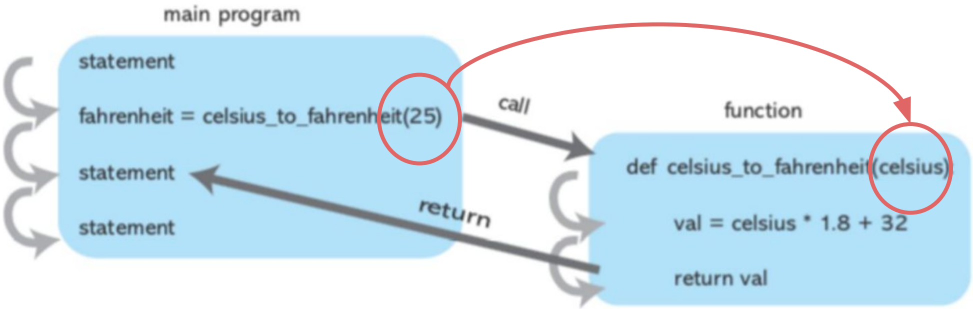


FIGURE 5.2 Function flow of control.

Default Parameter Values

A default parameter value: a value assigned to a function parameter **by default** in the event that the **user did not provide a value**.

However, when the user does provide a **value that provided value always overrides** the default value.

```
def func1(param_required, param_default = 2):
     print(param_required, param_default)
```

Arguments are mapped to parameters in a left-to-right positional matching.

If there are more parameters then arguments, any unmatched parameters get their default values. Therefore, default values can only be used on the rightmost parameters.

Parameter as Keywords

Python also allows you to **use the names of the parameters as keywords** in the function invocation.

Use of parameter names as keywords in a function invocation is particularly useful when there are many parameters and many have default values.

```
def func1(param_required, param_default = 2):
     print(param_required, param_default)

# arguments order does not matter, passing by name

func1(param_required = 3)
func1(param_default = 4, param_required = 3)
```

Output:
3 4
3 2

Triggering Question 3

Guess the output...

```
def length(a_str):
      count = 0
      for char in a_str:
            count += 1
      return count

x_str = "DDP1"
print(length(x_str))
```

1

```
def length(a_str):
      count = 0
      for char in a_str:
            count += 1
      return count

x_str = "DDP1"
print(length(x_str, "DDP2"))
```

2

Write the answer in the **comment section**

Variable Scoping

Not all variables are accessible from all parts of our program!
We call the part of a program where a variable is accessible its scope

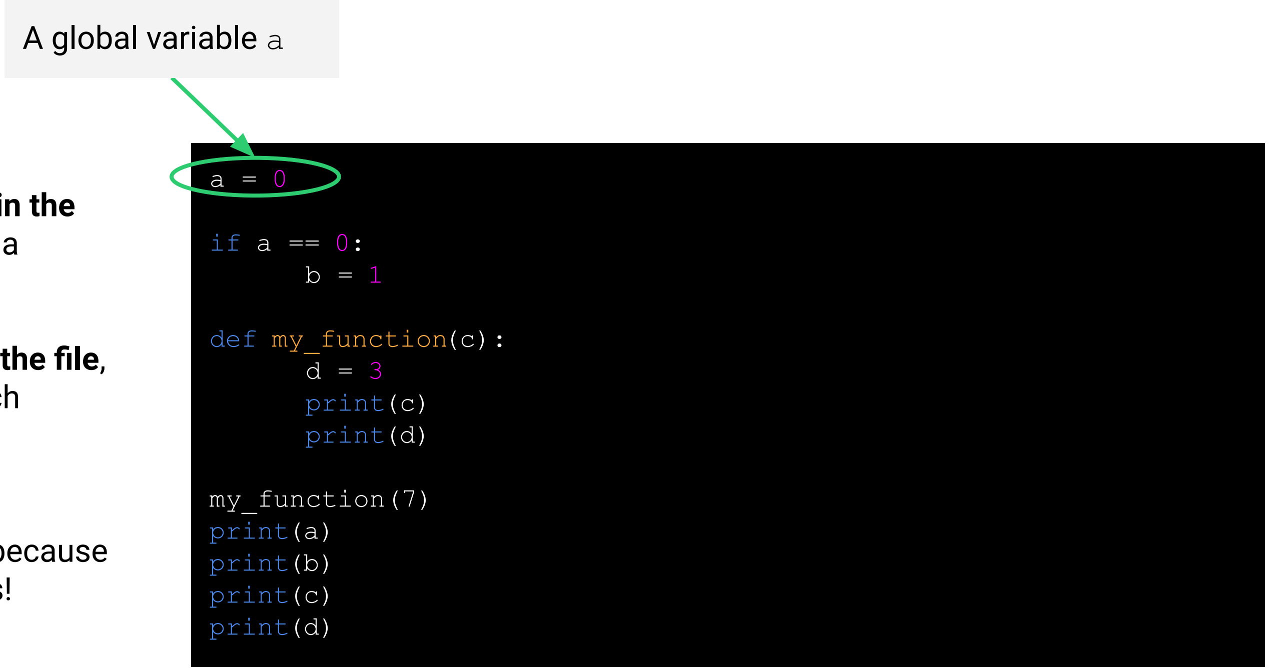
Scope:
The set of program statements over which a variable exists, that is, can be referred to.

```
a = 0

if a == 0:
      b = 1

def my_function(c):
      d = 3
      print(c)
      print(d)

my_function(7)
print(a)
print(b)
print(c)
print(d)
```

What are the outputs?

Variable Scoping: Global Variable

- ➔ A variable which is **defined in the main body** of a file is called a global variable.
- ➔ It will be **visible throughout the file**, and also inside any file which imports that file.
- ➔ Global variables can have unintended consequences because of their wide ranging effects!



A global variable a

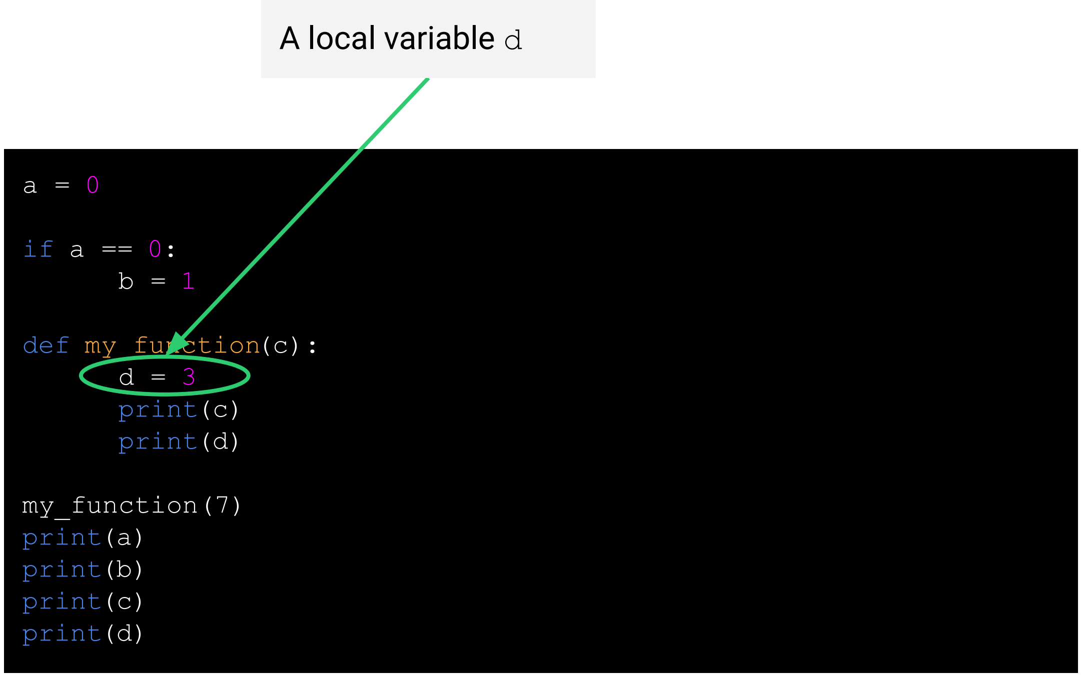
```
a = 0

if a == 0:
      b = 1

def my_function(c):
      d = 3
      print(c)
      print(d)

my_function(7)
print(a)
print(b)
print(c)
print(d)
```

Variable Scoping: Local Variable

- ➔ A variable which is defined **inside a function** is local to that function.

- ➔ It is accessible **from the point at which it is defined** until the end of the function.

- ➔ The parameter names in the function definition behave like local variables, but they contain the values that we pass into the function when we call it.



A local variable d

```
a = 0

if a == 0:
      b = 1

def my_function(c):
      d = 3
      print(c)
      print(d)

my_function(7)
print(a)
print(b)
print(c)
print(d)
```

Variable Scoping: Example (1)

```python
a = 0 # this is a global variable

if a == 0:
    b = 1 # this is also a global variable

def my_function(c): # c behaves like a local variable
    d = 3  # this is a local variable
    print(c)
    print(d)

my_function(7) # call the function, pass the value 7
print(a) # a still exists
print(b) # b too
print(c) # c is inaccessible. It is a local variable inside my_function. ERROR
print(d) # c is inaccessible. ERROR in print(c). SKIPPED
```

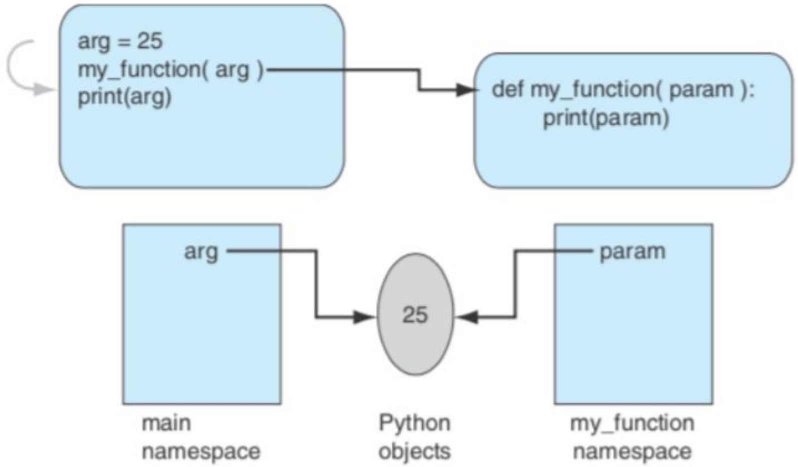
Variable Scoping: Example (2)

```python
# GLOBAL VARIABLES ARE REALLY GLOBAL

def f():
     print(s)

s = "Can't wait for Monday's quiz!"
f()
```

What are the outputs?

```python
# OVERRIDING GLOBAL VARIABLES

def f():
     s = "Oh no!"
     print(s)

s = "Can't wait for Monday's quiz!"
f()
```

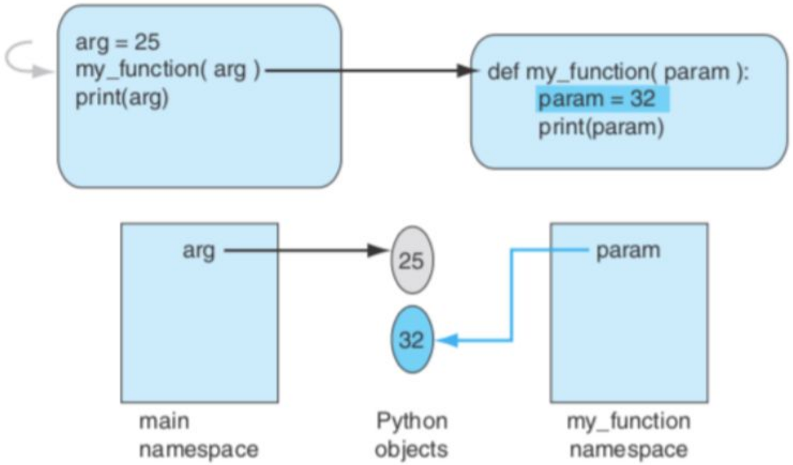
Arguments, Parameters, and Namespaces



```
arg = 25
my_function( arg )
print(arg)
```

```
def my_function( param ):
    print(param)
```

FIGURE 8.1 Function namespace: at function start.



```
arg = 25
my_function( arg )
print(arg)
```

```
def my_function( param ):
    param = 32
    print(param)
```

FIGURE 8.2 Function namespace modified.

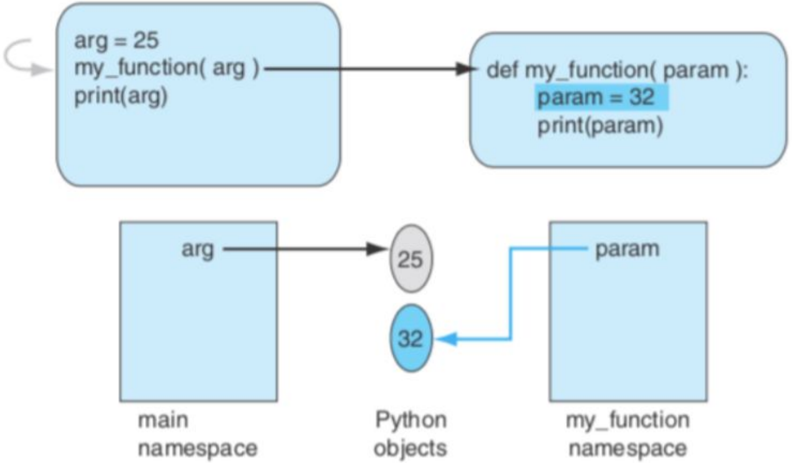
Pass “by Value” or “by Reference”?

It is useful to know that Python programmers do not need to think in those terms.

The best answer is to say that Python **passes object references**.

Everything in Python is an object, so every value passed is a reference to an object. Therefore, an object is not copied in this process.

- ➔ If the object is mutable, then a change made in the function is reflected in the outer scope of the object, for example, where function was called.
- ➔ If it is not mutable, a new object is referenced when the reference is updated.



```
arg = 25
my_function( arg )
print(arg)
```

```
def my_function( param ):
    param = 32
    print(param)
```

FIGURE 8.2 Function namespace modified.

Some Best Practices

A function ...

...does one thing. If it does too many things, it should be broken down into multiple functions (refactored)

...is not too long. Kind of synonymous with do one thing. Use it as a measure of doing too much.

...is readable. How often should we say this? If you write it, it should be readable

…is reusable. If it does one thing well, then when a similar situation (in another program) occurs, use it there as well.

...is complete. A function should check for all the cases where it might be invoked. Check for potential errors.

Review Questions

Explain what a function is and how to implement it.

Explain how the following function works.

```
def my_function(a_str):
      a_variable = 0
      for char in a_str:
            if(char == "D"):
                  a_variable += 1
      return a_variable

x_str = "DDP1, DDP2, SDA"
print(my_function(x_str))
```

Mention some of the best practices in implementing a function in Python?

Q&A Session