Introduction to Programming

PYTHON

Chapter 3 – Programming components
Variables, Functions, Modules

Presenter: **Dr. Nguyen Dinh Long**

Email: dinhlonghcmut@gmail.com

Google-site: https://sites.google.com/view/long-dinh-nguyen

Oct. 2022

Programming

.

Hamilton was a self-taught programmer, working in the US in the 1960’s. Owing to the success of her previous work, Hamilton was the first programmer to be hired for the Apollo project. She became the Director of Software Engineering at the MIT Instrumentation lab. Her lab developed the on-board flight software for NASA’s Apollo space project, which took humankind to the moon.

The achievement was a monumental task at a time when computer technology was in its infancy: The astronauts had access to only 72 kilobytes of computer memory (a 256-gigabyte cell phone today carries almost a million times more storage space). Programmers had to use paper punch cards to feed information into room-sized computers with no screen interface.

Margaret Hamilton, NASA’s lead software engineer for the Apollo, stands next to the code she wrote by hand that took humanity to the moon in 1969.

Outline

References

Main:

- Maurizio Gabbrielli and Simone Martini, 2010. *Programming Languages: Principles and Paradigms*, Springer.
- Cao Hoàng Trụ, 2004. *Ngôn ngữ lập trình- Các nguyên lý và mô hình*, Nhà xuất bản Đại học Quốc gia Tp. Hồ Chí Minh

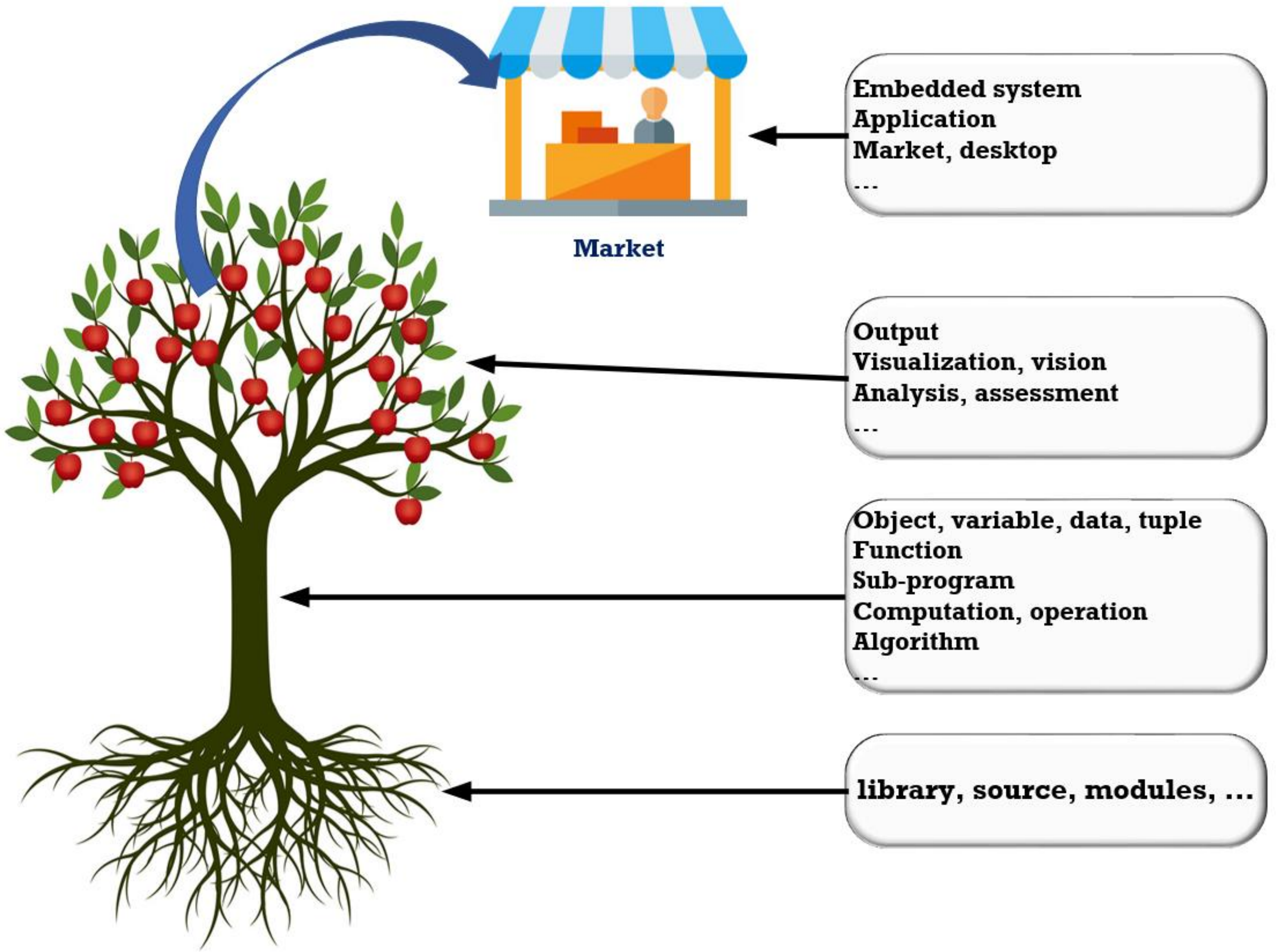
More:

- Wes McKinney, 2013. *Python for Data Analysis*, O'Reilly Media.
- Guido van Rossum, Fred L. Drake, Jr.,, 2012. *The Python Library Reference*, Release 3.2.3.

- Slides here are collected and modified from several sources in Universities and Internet.

Computer programs

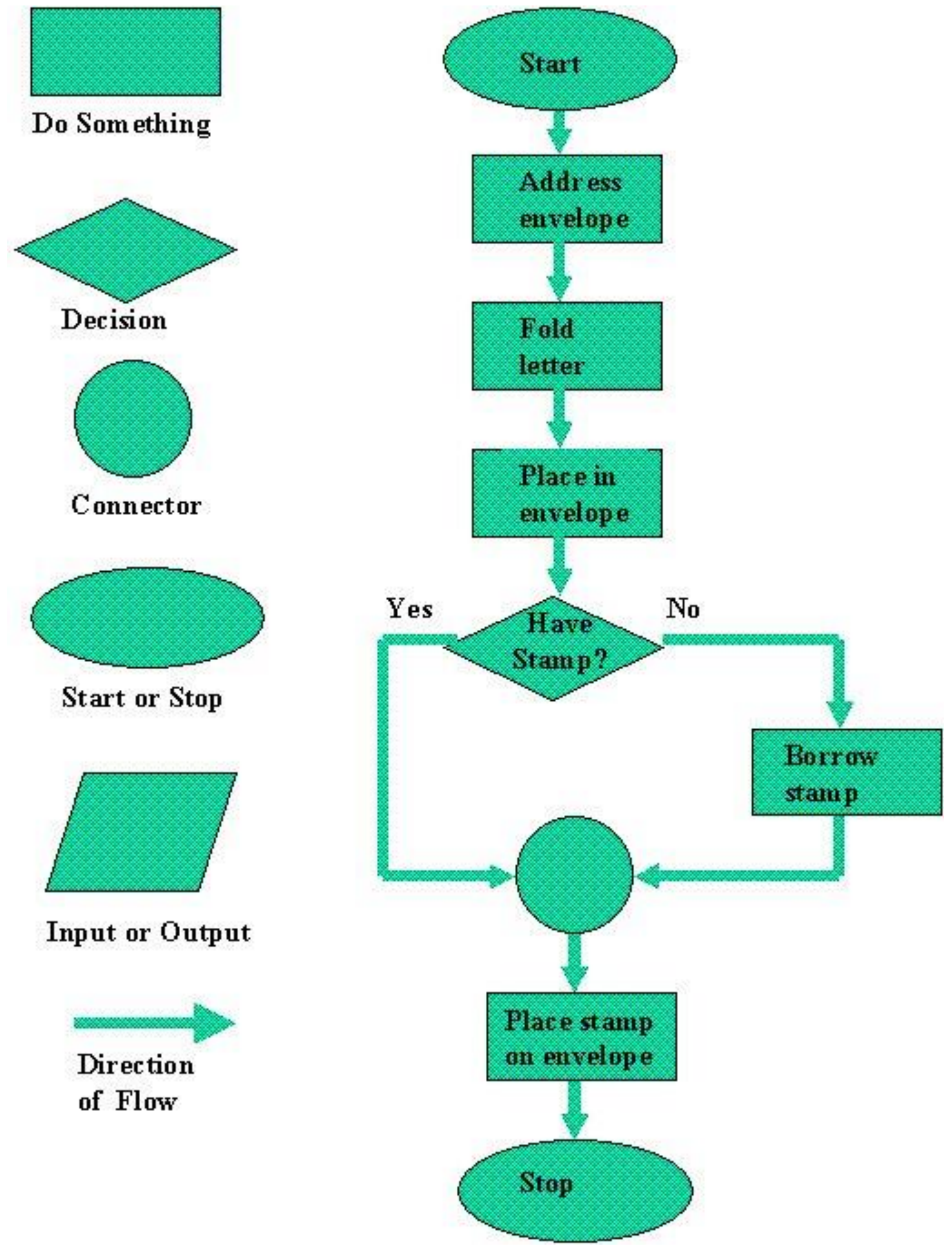
- General structure:



Content of Chapter 3

Structure of Computer programs

- **Computer programming:**
  - Objects
  - Types
  - Variables
  - Methods
  - Tuples



Programming components - Variables

Object Names (Variables):

- Objects can be given a name:

```
message = "CS101 is fantastic"

n = 17
hubo = Robot()
pi = 3.1415926535897931
finished = True
img = load_picture("geowi.jpg")
```

- In the Python zoo, the name is a sign board on the animal’s cage.

Programming components - Variables

❑ **Variables:**

- Names are often called variables, because the meaning of a name is variable: the same name can be assigned to different objects within a program.

```
n = 17
n = "Seventeen"
n = 17.0
```

- The object assigned to a name is called the value of the variable. The value can be changed over time.

- To indicate that a variable is empty, we use the special object None (of class ’NoneType’):

```
n = None
```

Programming components - Variables

- **Name structure of Variables:**

- The rules for variable and function names:
  - A name consists of letters, digits, and the underscore _.
  - The first character of a name should not be a digit.
  - The name should not be a keyword such as def, if, else, or while.
  - Upper case and lower case are different: Pi is not the same as pi.

Good:

- `my_message = "CS101 is fantastic"`
- `a13 = 13.0`

Bad:

- `more@ = "illegal character"`
- `13a = 13.0`
- `def = "Definition 1"`

Programming components - Functions

- **The name function comes from mathematics. A function is a mapping from one set to another set:**

.

$$f : R \rightarrow R$$

$$x \rightarrow \pi \times \frac{x}{180.0}$$

Here, x is the argument of the function, f (x) is the result of the function.

In Python, functions also take arguments and return a result:

```
def to_radians(deg):
    return (deg / 180.0) * math.pi
```

```
>>> a = to_radians(90)
>>> print(a)
1.5707963267948966
```

Programming components - Functions

- **Useful functions:**

Python comes with many built-in functions

```
>>> int("32")
32
>>> int(17.3)
17
>>> float(17)
17.0
>>> float("3.1415")
3.1415
>>> str(17) + " " + str(3.1415)
'17 3.1415'
>>> complex(17)
(17 + 0j)
```

Math functions:
To use math functions, we need to tell Python that we want to use the math module:

```
import math
degrees = 45
radians = degrees / 360.0 * 2 * math.pi
print(math.sin(radians))
print(math.sqrt(2) / 2)
```

When using math functions often, we can use shorter names:

```
import math
sin = math.sin
pi = math.pi
radians = degrees / 360.0 * 2 * pi
print(sin(radians))
```

Programming components - Functions

- **Functions are reusable pieces of programs:**

- They allow us to give a name to a block of statements. We can execute that block of statements by just using that name anywhere in our program and any number of times.
- Functions are defined using the def keyword. This is followed by an identifier name for the function.

```
def sayHello():
    print 'Hello World!'  # A new block
    # End of the function
sayHello() # call the function
```

Programming components - Functions with parameters

- **Defining functions:**

- Are values we supply to the function to perform any task. Specified within the pair of parentheses () in the function definition, separated by commas.
- When we call the function, we supply the values in the same way and order.
  - the names given in the function definition are called parameters.
  - the values we supply in the function call are called arguments.
- Arguments are passed using call by value
  (where the value is always an object reference, not the value of the object)

```
# Demonstrating Function Parameters

def printMax(a, b):
    if a > b:
        print a, 'is maximum'
    else:
        print b, 'is maximum'

printMax(3, 4) # Directly give literal values

x = -5

y = -7

printMax(x, y) # Give variables as arguments
```

Programming components - Functions

❑ Calling functions:

When a function is called, the arguments of the function call are assigned to the parameters:

```
def print_twice(text):
    print(text)
    print(text)
```

Parameter

The number of arguments in the function call must be the same as the number of parameters.

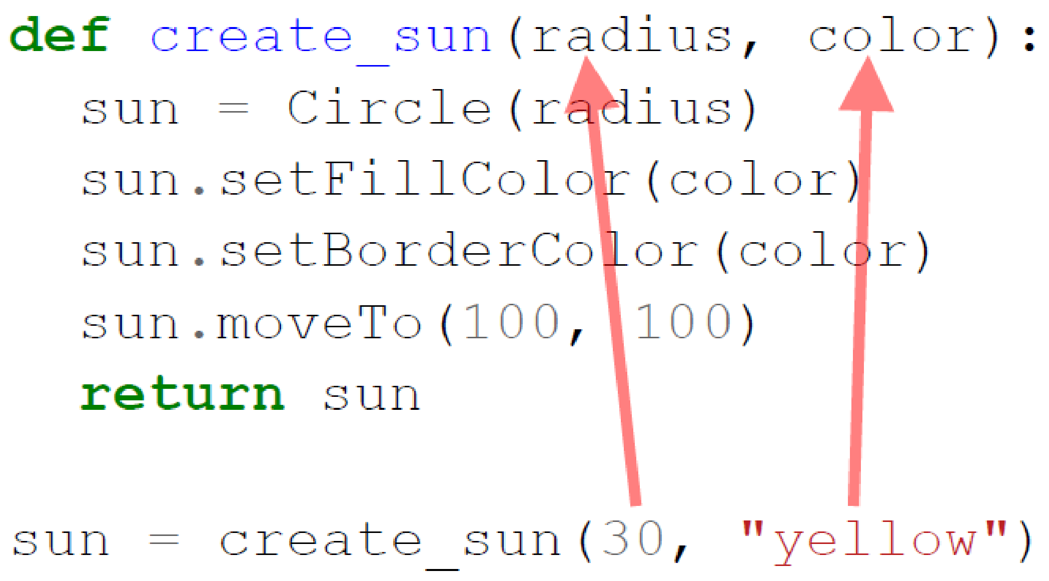
```
>>> print_twice("I love CS101")
I love CS101
I love CS101
>>> print_twice(math.pi)
3.14159265359
3.14159265359
```

(*)Remember: A parameter is a name for an object. The name can only be used inside the function.

Programming components – Functions with arguments

❑ **Function arguments:**

We have learned about parameters and function arguments:



```
def create_sun(radius, color):
    sun = Circle(radius)
    sun.setFillColor(color)
    sun.setBorderColor(color)
    sun.moveTo(100, 100)
    return sun

sun = create_sun(30, "yellow")
```

Arguments are mapped to parameters one-by-one, left-to-right.

Programming components – Functions with arguments

❑ **Function arguments:**

What does this code print?

```
def swap(a, b):
    a, b = b, a

x, y = 123, 456
swap(x, y)
print (x, y)
```

"a" is a new name for the object "123", not for the name "x"!

Programming components - Functions with parameters

❑ **Defining functions with parameters:**

The function definition uses names for the arguments of the function. These names are called parameters:

```
def compute_interest(amount, rate, years):
```

Inside the function, the parameter is just a name:

```
value = amount * (1 + rate/100.0) ** years
```

When we have computed the result of the function, we return it from the function.

The function ends at this point, and the result object is given back:

```
return value
```

We can now call the function with different argument values:

```
>>> s1 = compute_interest(200, 7, 1)
>>> s2 = compute_interest(500, 1, 20)
```

Programming components - Functions with parameters

❑ **Default parameters:**

We can provide default parameters:

```
def create_sun(radius = 30, color = "yellow"):
    sun = Circle(radius)
    sun.setFillColor(color)
    sun.setBorderColor(color)
    sun.moveTo(100, 100)
    return sun
```

?

Now we can call it like this:

```
sun = create_sun()
```

```
star = create_sun(2)
moon = create_sun(28, "silver")
```

But not like this:

```
moon = create_sun("silver")
```

Programming components - Functions with parameters

❑ **Normal and default parameters:**

Default parameters have to follow normal parameters:

```
def avg(data, start = 0, end = None):
    if not end:
        end = len(data)
    return sum(data[start:end]) / float(end-start)
```

```
>>> d = [ 1, 2, 3, 4, 5 ]
>>> avg(d)
3.0
>>> avg(d, 2)
4.0
>>> avg(d, 1, 4)
3.0
```

Programming components - Functions with parameters

- **Named parameters:**

We can include the name of the parameter in the function call to make the code clearer. Then the order of arguments does not matter:

```
moon = create_sun(color = "silver")
moon = create_sun(color = "silver", radius = 28)
```

```
>>> avg(d, end=3)
2.0
>>> avg(data=d, end=3)
2.0
>>> avg(end=3, data=d)
2.0
>>> avg(end=3, d)
SyntaxError: non-keyword arg after keyword arg
```

Programming components – Functions with the return

- **Returning a boolean:**

A function that tests a condition and returns either True or False is often called a predicate:

```
# Is integer a divisible by b?
def is_divisible(a, b):
  if a % b == 0:
    return True
  else:
    return False
```

A predicate (function) can be used directly in an if or while statement:

```
if is_divisible(x, y):
  print('x is divisible by y')
```

Easier:

```
def is_divisible(a, b):
  return a % b == 0
```

Programming components - Functions with the return

- **More than one return in a function:**

Compute the absolute value (like built-in function abs):

```
def absolute(x):
  if x < 0:
    return -x
  else:
    return x
```

The same function can be written like this:

```
def absolute(x):
  if x < 0:
    return -x
  return x
```

But not like this:

```
def absolute(x):
  if x < 0:
    return -x
  if x > 0:
    return x
```

Programming components - Functions with the return

- **Returning more than one value:**

A function can only return one value.

But this value can be a tuple, and functions can return arbitrarily many values by returning them as a tuple:

```
def student():
    name = "Hong"
    id = 20101234
    return name, id
```

Often function results are unpacked immediately:

```
name, id = student()
```

Programming components - Functions with the return

- **Functions without results:**

We have seen many functions that do not use return:

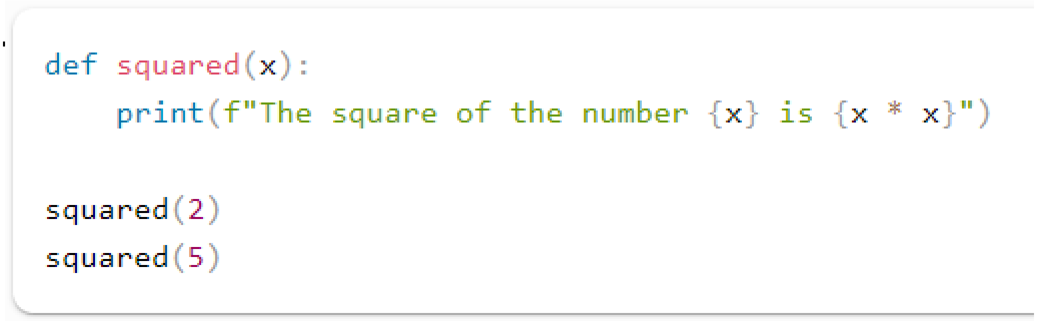
```
def turn_right():
  for i in range(3):
    hubo.turn_left()
```

In fact, a function that does not call return automatically returns None:

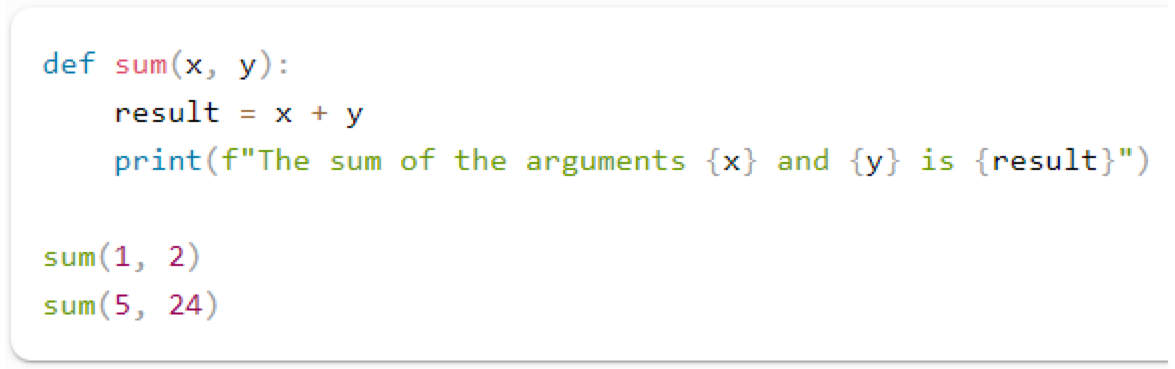
```
>>> s = turn_right()
>>> print(s)
None
```

Programming components - Functions with practice

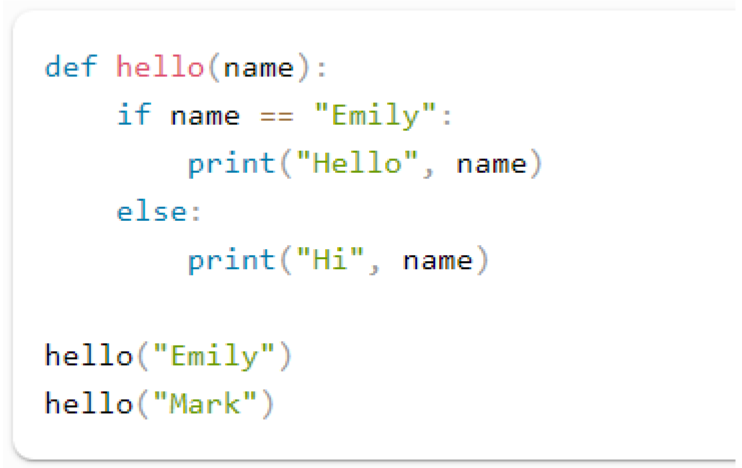
- **More examples:**



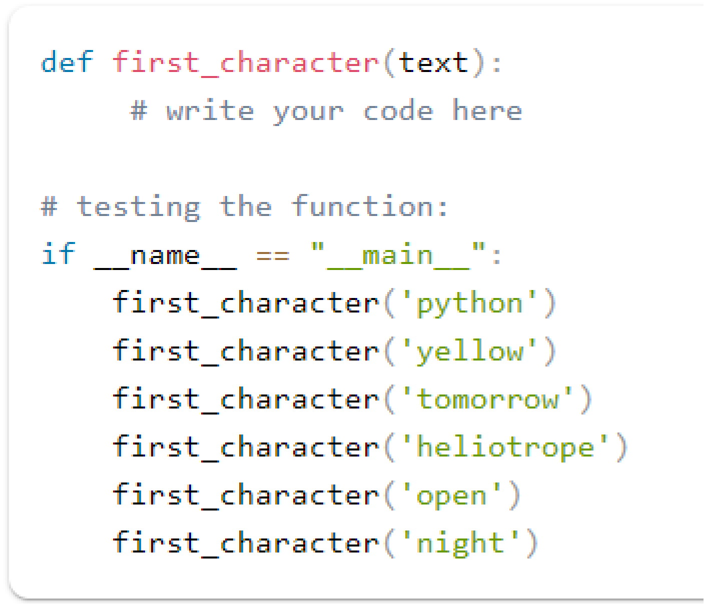
```python
def squared(x):
    print(f"The square of the number {x} is {x * x}")

squared(2)
squared(5)
```



```python
def sum(x, y):
    result = x + y
    print(f"The sum of the arguments {x} and {y} is {result}")

sum(1, 2)
sum(5, 24)
```



```python
def hello(name):
    if name == "Emily":
        print("Hello", name)
    else:
        print("Hi", name)

hello("Emily")
hello("Mark")
```

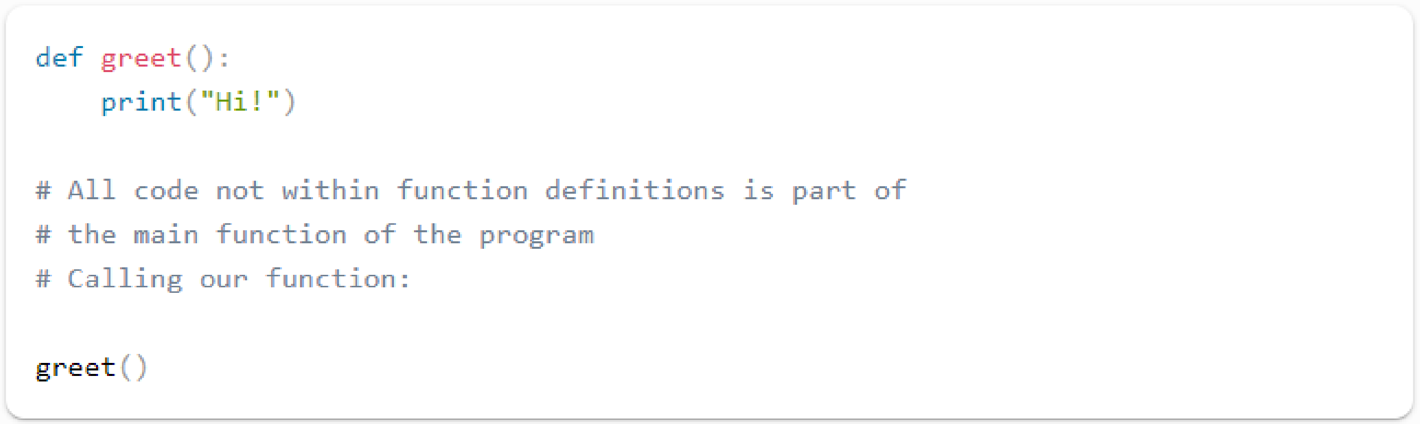


```python
def first_character(text):
     # write your code here

# testing the function:
if __name__ == "__main__":
    first_character('python')
    first_character('yellow')
    first_character('tomorrow')
    first_character('heliotrope')
    first_character('open')
    first_character('night')
```

Programming components - Functions with practice

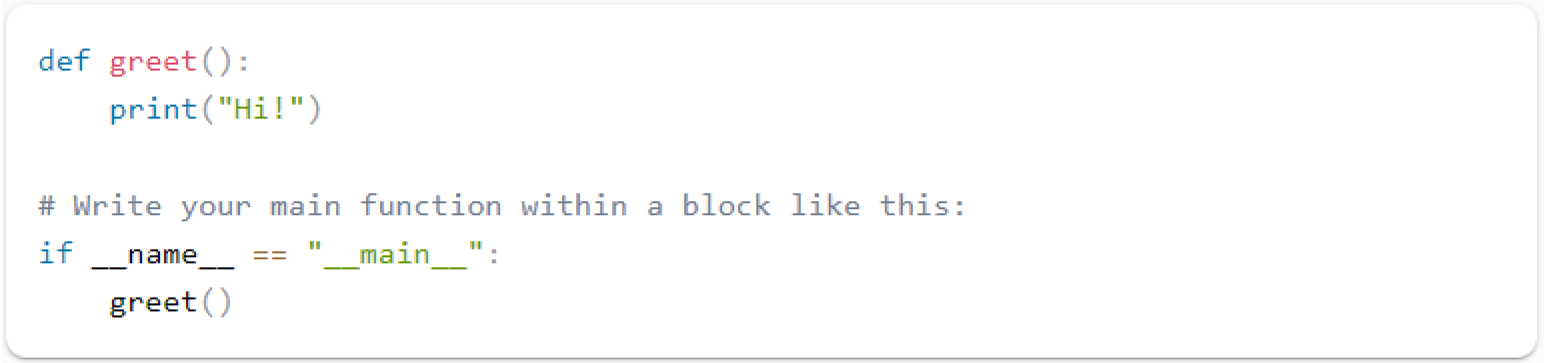
❑ Testing your own functions:

The "main" program below the function should contain appropriate function calls, so that the program can be tested. So, lets add a function call:



```python
def greet():
    print("Hi!")

# All code not within function definitions is part of
# the main function of the program
# Calling our function:

greet()
```

<u>Important:</u> No commands should be left in the main function of your solution. That is, any code that you yourself use for testing must be contained in a specially defined if block:



```python
def greet():
    print("Hi!")

# Write your main function within a block like this:
if __name__ == "__main__":
    greet()
```

It is worth noting that the tests will not execute any code from within the if __name__ == "__main__" block

Programming components - Functions with practice

❑ **Python Main Function:**

Starting point of any program. When the program is run, the python interpreter runs the code sequentially. Main function is executed only when it is run as a Python program. It will not run the main function if it imported as a module:

```
def main():
     print ("Hello World!")
print ("Guru99")
```

```
def main():
    print("Hello World!")

if __name__ == "__main__":
    main()

print("Guru99")
```

- When Python interpreter reads a source file, it will execute all the code found in it.
- When Python runs the "source file" as the main program, it sets the special variable (__name__) to have a value ("__main__").
- When you execute the main function in python, it will then read the "if" statement and checks whether __name__ does equal to __main__.
- In Python "if__name__== "__main__" allows you to run the Python files either as reusable modules or standalone programs.

Programming components – Functions with Variables

Local variables:

A function to evaluate the quadratic function: $ax^2 + bx + c$

```
def quadratic(a, b, c, x):
  quad_term = a * x ** 2
  lin_term = b * x
  return quad_term + lin_term + c

result = quadratic(2, 4, 5, 3)
```

The names “quad_term” and “lin_term” exist only during the execution of the function quadratic. They are called local variables.

A function’s parameters are also local variables. When the function is called, the arguments in the function call are assigned to them.

Local variables are names that only exist during the execution of the function, defined inside the function.

Programming components - Functions with Variables

❑ **Why local variables:**

Humans are not good at remembering too many things at the same time.

To use the function quadratic, we only want to remember this:

```
def quadratic(a, b, c, x):
  # implemented somehow
```

Modularization means that software consists of parts that are developed and tested separately. To use a part, you do not need to understand how it is implemented.

cs1robots is a module that implements the object type *Robot*. You can use Robot easily without understanding how it is implemented.

→ Object-oriented Programming

Programming components - Functions with Variables

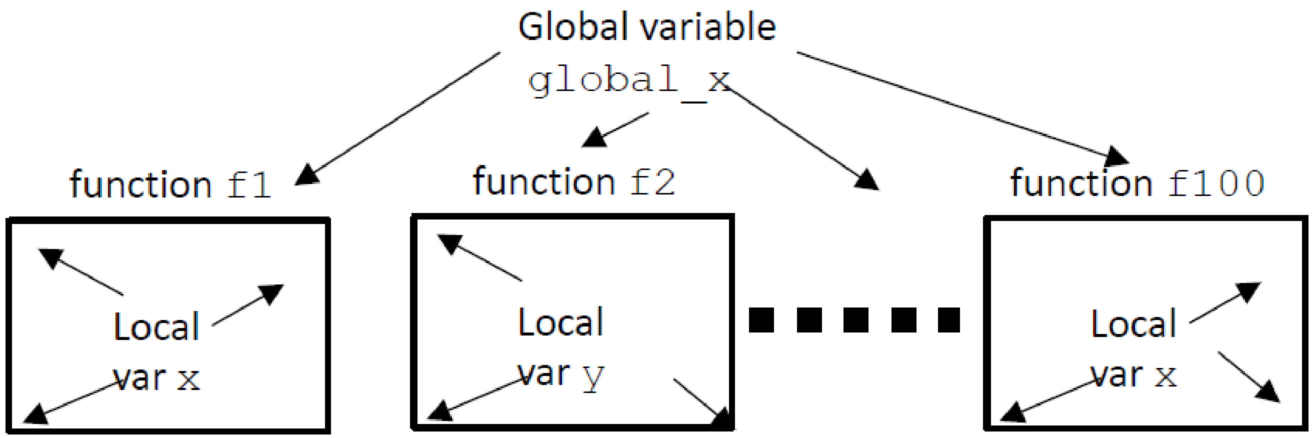
- **Global variables:**

Variables defined *outside of a function* are called global variables.

Global variables can be used inside a function:

```
hubo = Robot()          ← global variable hubo
def turn_right():
  for i in range(3):
    hubo.turn_left()    ← using global variable
```

In large programs, using global variables is dangerous, as they can be accessed (by mistake) by all functions of the program.



Programming components - Functions with Variables

- **Local vs. global:**

If a name is only used inside a function, it is global:

```
def f1():
    return 3 * a + 5
```

If a name is assigned to in a function, it is local:

```
def f2(x):
    a = 3 * x + 17
    return a * 3 + 5 * a
```

What does this *test* function print?

```
a = 17
def test():
    print(a)
    a = 13
    print(a)
test()
```

Error!
A is a local variable in *test* function because of the assignment, but has no value inside the first print statement.

Programming components - Functions with Variables

- **Local vs. global:**

Example

```
a = "Letter a"

def f(a):
    print("A = ", a)
```

```
def g():
    a = 7
    f(a + 1)
    print("A = ", a)

print("A = ", a)
f(3.14)
print("A = ", a)
g()
print("A = ", a)
```

```
A = Letter a
A = 3.14
A = Letter a
A = 8
A = 7
A = Letter a
```

Programming components - Functions with Variables

❑ **Assigning to global variables:**

Sometimes we want to change the value of a global variable inside a function:

```
hubo = Robot()
hubo_direction = 0
```

```
def turn_left():
  hubo.turn_left()
  global hubo_direction
  hubo_direction += 90
```

```
def turn_right():
  for i in range(3):
    hubo.turn_left()
  global hubo_direction
  hubo_direction -= 90
```

Programming components - Functions with Multiple calls

- **Multiple Function calls:**

Sometimes we want to have functions calling other functions.

f(g(4))

- In this case, we use the 'inside out' rule, that is we apply “g” first, and then we apply “f” to the result.
- If the functions can have local variables, this can get complicated.

- **Designing Functions:**

- Need to choose parameters.
  - Ask “what does the function need to know”.
  - Everything it needs to know should be passed as a parameter.
  - Do not rely on global parameters.
- Need to choose whether to return or not to return.

➢ Functions that return information to code should return, those that show something to the user shouldn't (print, media.show(), ...).

Programming components - Modules

- A Python module is a collection of functions that are grouped together in a file. Python comes with a large number of useful modules. We can also create our own modules**.**

- math for mathematical functions
- random for random numbers and shuffling
- sys and os for accessing the operating system
- urllibto download files from the web
- graphics for graphics
- media for processing photos

Programming components - Modules

❑ **Importing modules:**

Before you can use a module you have to import it:

```
import math
print(math.sin(math.pi / 4))
```

Sometimes it is useful to be able to use the functions from a module without the module name:

```
from math import *
print(sin(pi / 4))      # OK
print(math.pi)          # NameError: name 'math'
```

Or only import the functions you need:

```
from math import sin, pi
print(sin(pi / 4))    # OK
print(cos(pi / 4))     # NameError: name 'cos'
print(math.cos(pi/4)) # NameError: name 'math'
```

Programming components – Functions with adding delay

❑ Python time.sleep(): Add Delay to Your Code

A function used to delay the execution of code for the number of seconds given as input to sleep(). The sleep() command is a part of the time module. You can use the sleep() function to temporarily halt the execution of your code.

```
import time
time.sleep(seconds)
```

```
import time
print("Welcome to guru99 Python Tutorials")
time.sleep(5)
print("This message will be printed after a wait of 5 seconds")
```

```
import time

print('Code Execution Started')

def display():
    print('Welcome to Guru99 Tutorials')
    time.sleep(5)

display()
print('Function Execution Delayed')
```

Programming components – Functions with adding delay

❑ **Python time.sleep(): Add Delay to Your Code**

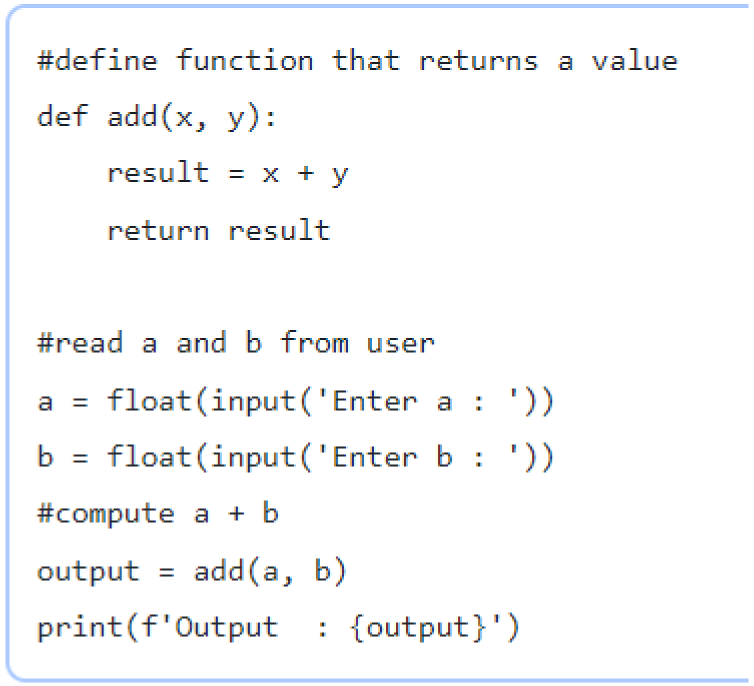
Using Event().wait

```
from threading import Event

print('Code Execution Started')

def display():
    print('Welcome to Guru99 Tutorials')
Event().wait(5)
display()
```

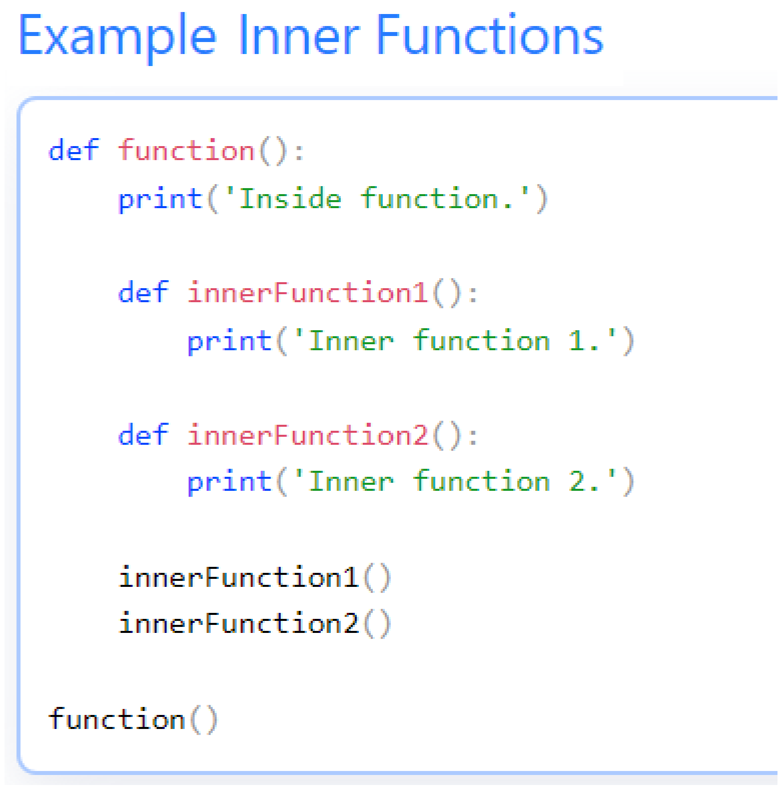
Using Timer

```
from threading import Timer

print('Code Execution Started')

def display():
    print('Welcome to Guru99 Tutorials')

t = Timer(5, display)
t.start()
```

Programming components – Examples

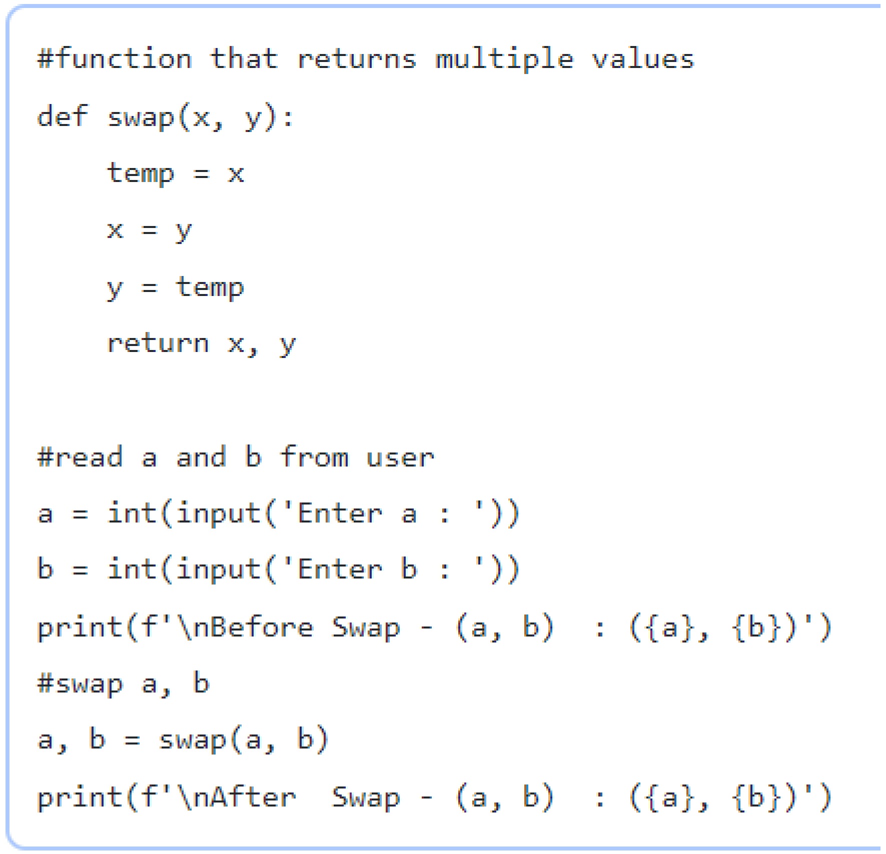


```
#define function that returns a value
def add(x, y):
    result = x + y
    return result

#read a and b from user
a = float(input('Enter a : '))
b = float(input('Enter b : '))
#compute a + b
output = add(a, b)
print(f'Output  : {output}')
```



Example Inner Functions

```
def function():
    print('Inside function.')

    def innerFunction1():
        print('Inner function 1.')

    def innerFunction2():
        print('Inner function 2.')

    innerFunction1()
    innerFunction2()

function()
```

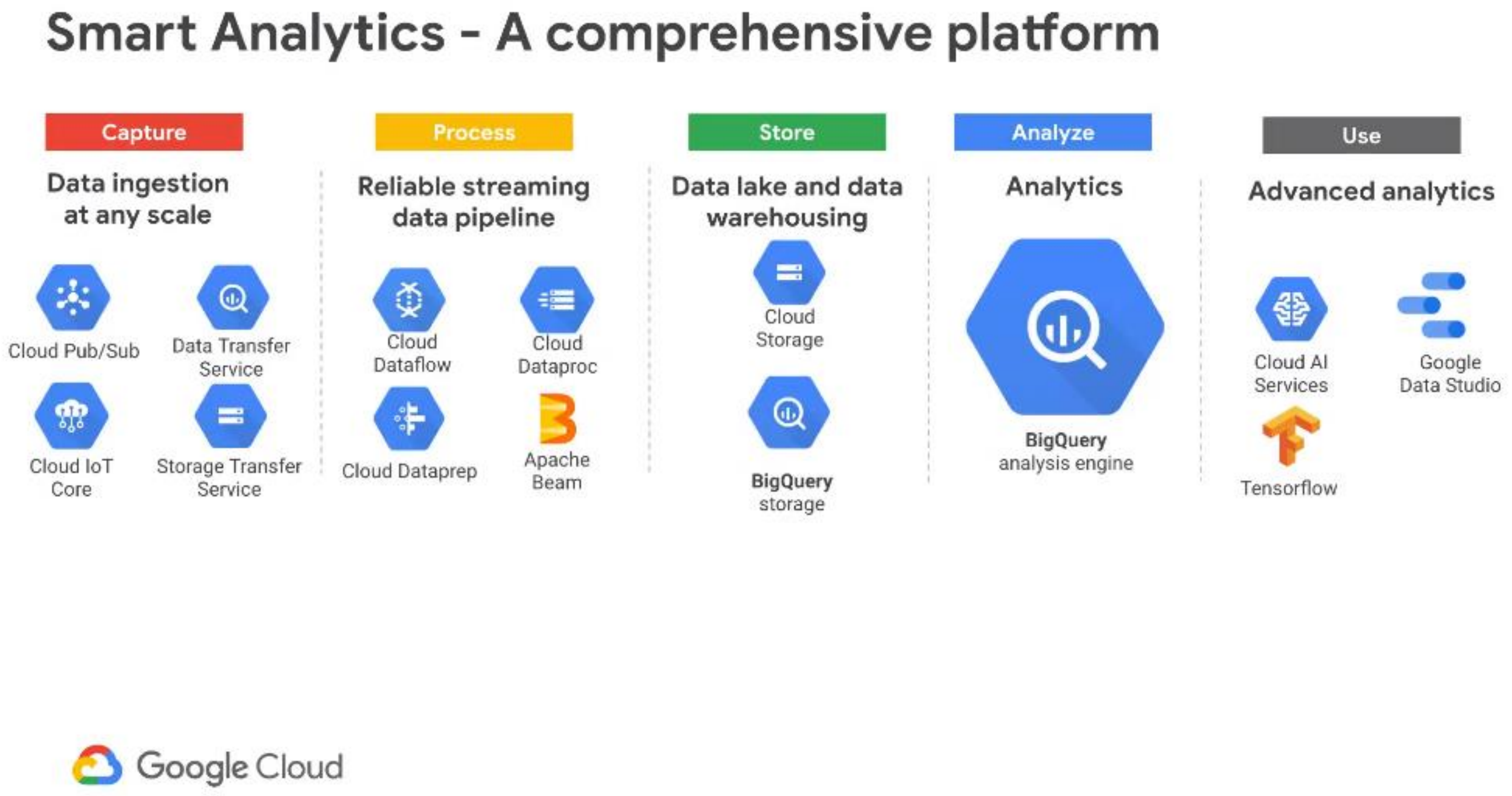


```
#function that returns multiple values
def swap(x, y):
    temp = x
    x = y
    y = temp
    return x, y

#read a and b from user
a = int(input('Enter a : '))
b = int(input('Enter b : '))
print(f'\nBefore Swap - (a, b)  : ({a}, {b})')
#swap a, b
a, b = swap(a, b)
print(f'\nAfter  Swap - (a, b)  : ({a}, {b})')
```

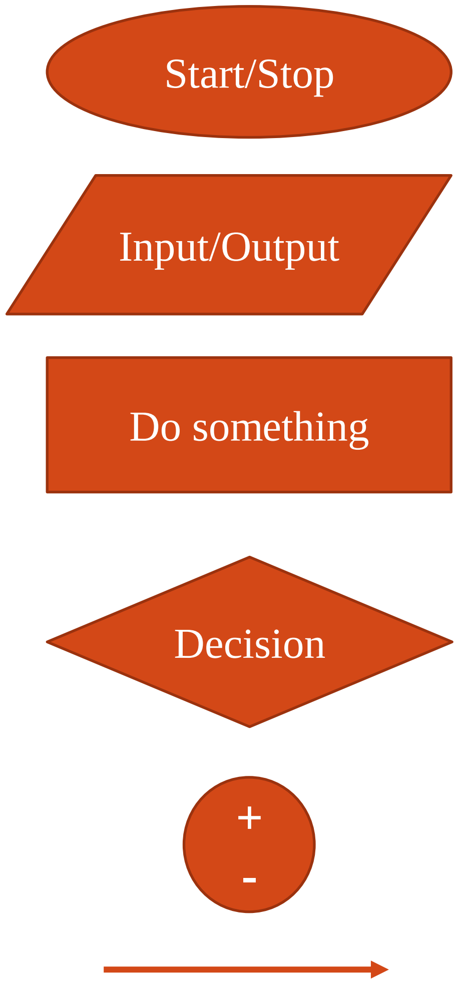
Programming for Data Analysis

❑ Data Analysis is the technique to collect, transform, and organize data to make future predictions, and make informed data-driven decisions. It also helps to find possible solutions for a business problem. There are six steps for Data Analysis.

1. Ask or Specify Data Requirements
2. Prepare or Collect Data, Read data
3. Clean and Process
4. Analyze, Compute
5. Share, Visualization
6. Act or Report, prediction …



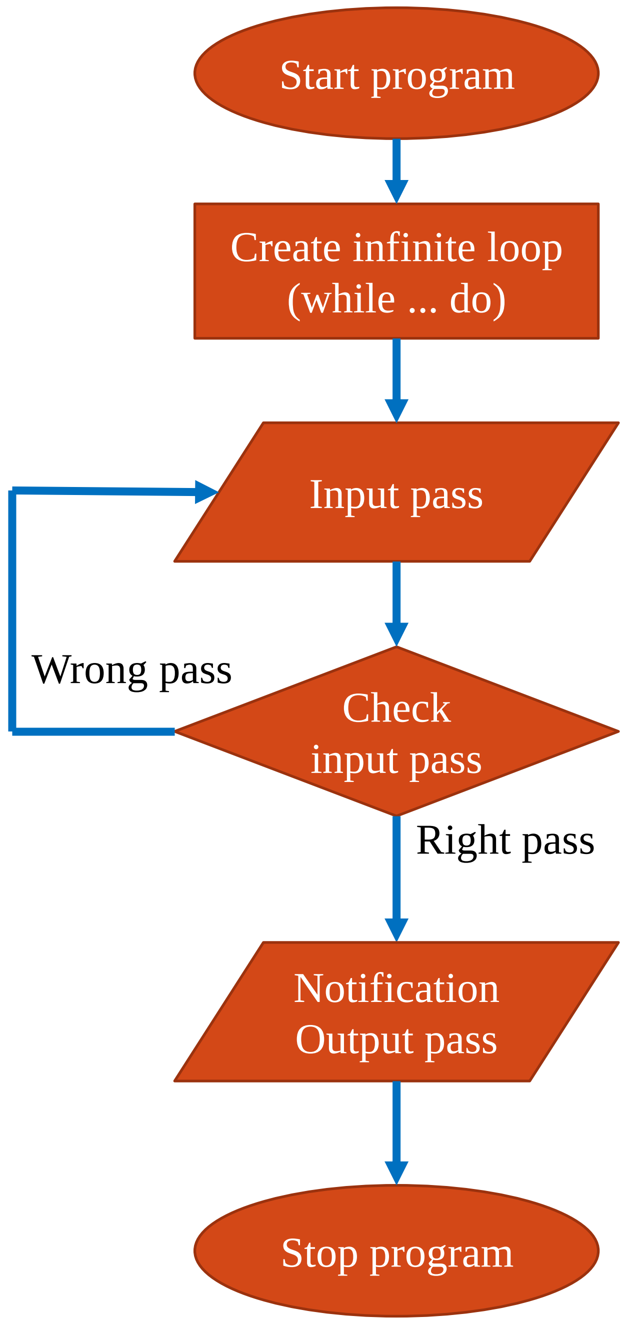
Bài tập thực hành



1. Viết một chương trình (giao diện) bảo mật, yêu cầu nhập đúng mật mã mới tiến hành các hoạt động tiếp theo (cho phép tiếp tục code).
   Nếu nhập không đúng mật mã thì sẽ không cho thực hiện hoạt động nào.

Bài tập thực hành

- **Xây dựng flow chart (algorithm):**
  - Xây dựng môi trường (giao diện) tương tác
    → tạo vòng lặp vô tận
  - Yêu cầu nhập mật mã (giả sử only text)
  - Ra quyết định:
    - Đúng mật mã thì thoát khỏi vòng lặp, thông báo và chuyển sang thực hiện code tiếp
    - Nếu mật mã không đúng, giữ nguyên vòng lặp, thông báo mật mã sai quay trở lại vòng lặp nhập mật mã
  - Nếu mật mã đúng, xuất thông báo và hiển thị mật mã đúng
  - Kết thúc


Start program
Create infinite loop (while ... do)
Input pass
Wrong pass
Check input pass
Right pass
Notification Output pass
Stop program