COMP1021
Introduction to Computer Science

Functions

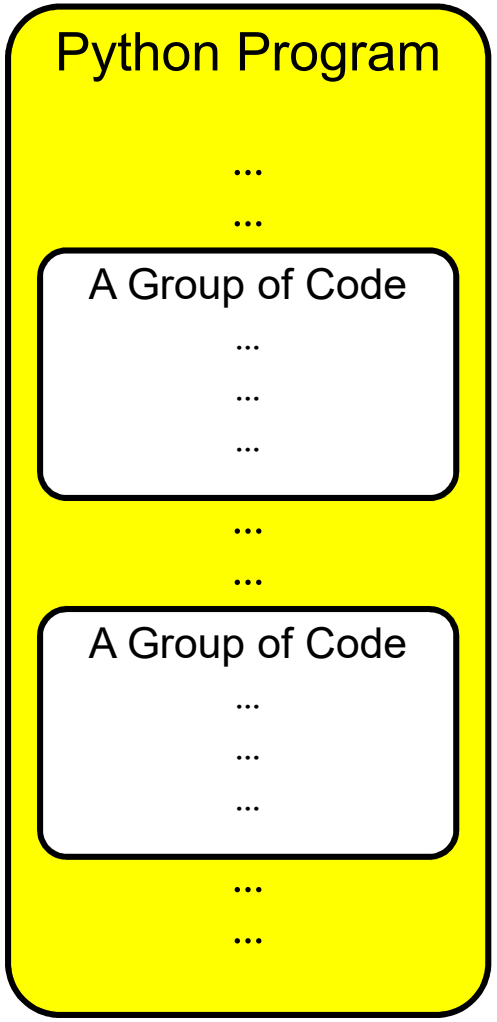
David Rossiter and Gibson Lam

Outcomes

- After completing this presentation, you are expected to be able to:
  1. Define and use a function in Python
  2. Explain the working areas of local variables and global variables
  3. Stop a function and return values from the function using the return command

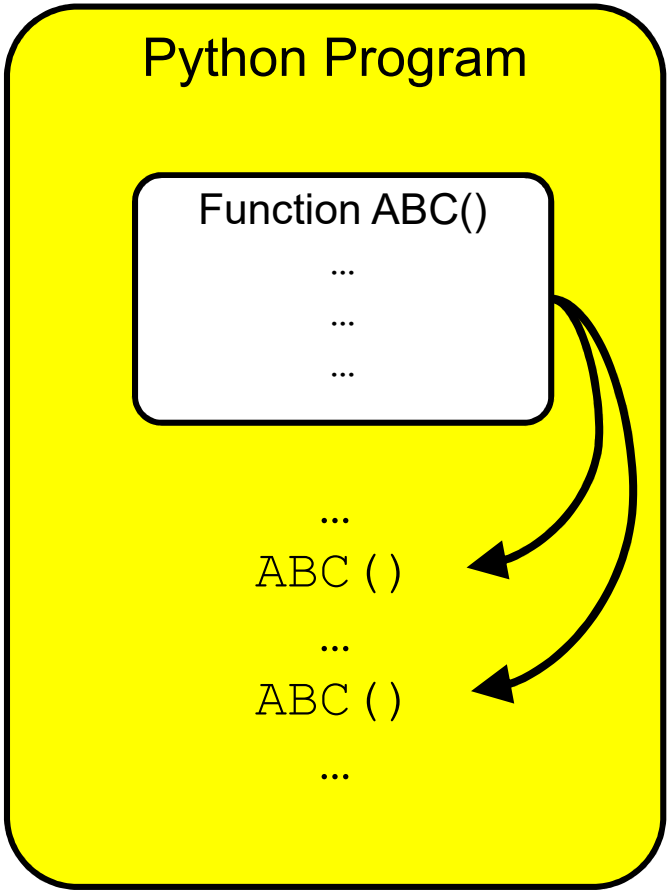
Running a Group of Code

- Sometimes you may want to put the same group of code in different places in your program
- To do that, a straightforward way is to copy and paste the same code into those places inside the program
- However, the program will become very long and contain a lot of duplicated code



Functions

- Instead of copying and pasting the group of code everywhere, the group of code is first put inside a *function*
- You can then use the function as many times as you like in appropriate places inside a program



Functions You Have Already Used

- We have already used a lot of different functions in the course
- For example, `print`, `input` and `turtle.forward` are all functions that we have used before
- These are functions made by others, i.e. the people who made the Python language
- In this presentation, we will look at making our own functions and then use them

Defining a Function

- To make a function in Python, we use the def command (**def**ining a function)
- Here is an example:

This is the name of the function (you need to put parentheses after the name)

This is the code of the function

```
def greeting():
    name = input("What is your name? ")
    print("Welcome " + name + "!")
```

- When we define a function, we need to give it a name
- We will refer to this name when we want to use the function later

Using a Function

- To use the function we have defined in the previous slide, we simply run it using its name, like this:

```
def greeting():
    name = input("What is your name? ")
    print("Welcome " + name + "!")
```

The function we defined before

```
print ("I am going to ask you a question...")
greeting()
```

The function is used here (again, you need to put parentheses after the name)

```
I am going to ask you a question...
What is your name? Gibson
Welcome Gibson!
>>>
```

Defining and then Using Functions

- When you make functions you have to make sure that you define them before you use them
- If you don't, Python will give you an error, e.g.:

```
print ("I am going to ask you a question...")
greeting()

def greeting():
    name = input("What is your name? ")
    print("Welcome " + name + "!")
```

Here the function is used before it is defined

```
I am going to ask you a question...
Traceback (most recent call last):
  File "C:\greeting.py", line 2, in <module>
    greeting()
NameError: name 'greeting' is not defined
>>>
```

Using a Function Multiple Times

- You can run a function as many times as you like
- For example, we can run a function three times in different places:

```
def response():
    print("Very good!")

print("Is it a good course?")
response()
print("Is the instructor good?")
response()
print("Do I look good?")
response()
```

```
Is it a good course?
Very good!
Is the instructor good?
Very good!
Do I look good?
Very good!
>>>
```

Passing a Value into a Function

- Sometimes it is useful to give some values to a function so that it can do different things
- We called that 'passing values into a function' in computer science terms
- Here is an example:

In this example, the function is expected to receive a value, stored in a variable called 'name'

```
def magic_mirror( name ):
    if name == "Gibson":
        print("What a good name!")
    else:
        print("How are you?")
```

Running the Function

- You can ‘pass’ the value directly into the function

```
>>> magic_mirror("Joe")
How are you?
>>>
```

- Or, as in most cases, the value that you pass into the function is likely stored in a variable, like this:

```
name = input("What is your name? ")
magic_mirror(name)
```

```
What is your name? Gibson
What a good name!
>>>
```

Using Variables with the Same Name

- If you look at our previous magic mirror example:

The name variable here is the value passed into the function

The name variable is also used in the main program

```
def magic_mirror(name):
    if name == "Gibson":
        print("What a good name!")
    else:
        print("How are you?")

name = input("What is your name? ")
magic_mirror(name)
```

- It can be quite confusing when variables with the same name appear in different places of the program
- The problem is, even though the variables have the same name, there are in fact two different variables

Local and Global Variables

- Local Variables
  - They are variables created inside a function
  - They work only inside the function where they are created
- Global Variables
  - They are variables created outside of any function
  - They work everywhere, including inside any function
- If a local variable and a global variable have the same name, priority will be given to the local variable

Local and Global Variables in the Example

- Looking at our example again:

The local variable **name** *works in this area*

```
def magic_mirror(name):
    if name == "Gibson":
        print("What a good name!")
    else:
        print("How are you?")

name = input("What is your name? ")
magic_mirror(name)
```

The global variable **name** *works in this area*

Using Different Names

- Since having the same name for local and global variables is very confusing we should try our best to use different names, for example:

```
def magic_mirror(name):
    if name == "Gibson":
        print("What a good name!")
    else:
        print("How are you?")

name_input = input("What is your name? ")
magic_mirror(name_input)
```

`name_input` *is used here, no more confusion!*

Changing Local Variables

- You need to be careful when you change a local variable, like this:

```
def magic_trick(money):
    if money < 1000:
        money = money + 500

money = int(input("How much do you have? "))
magic_trick(money)
print("You have $" + str(money) + " now!")
```

The local variable is changed in this line of code

```
How much do you have? 500
You have $500 now!
>>>
```

The global variable money *is not affected by the change inside the function*

Changing Global Variables inside a Function

- If you want the global variable to be changed by a function you need to tell Python using the global command, for example:

```
def magic_trick():
    global money

    if money < 1000:
        money = money + 500

money = int(input("How much do you have? "))
magic_trick()
print("You have $" + str(money) + " now!")
```

We tell Python that we are going to change the value of the global variable `money`

This line then changes the value of the global variable

Running the Example

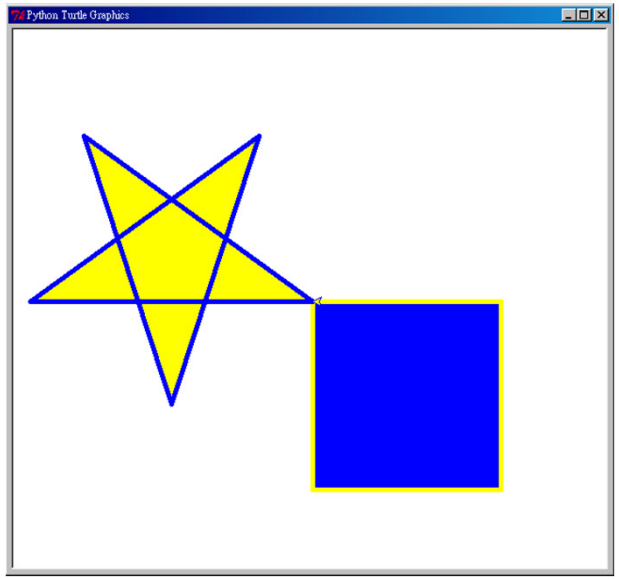
- Here is what we get if we run the example and then enter 500:

```
How much do you have? 500
You have $1000 now!
>>>
```

- If you remove the line '`global money`' and then run the program again, you will get an error like this:

```
How much do you have? 500
Traceback (most recent call last):
  File "C:\global.py", line 6, in <module>
    magic_trick()
  File "C:\global.py", line 2, in magic_trick
    if money < 1000:
UnboundLocalError: local variable 'money' referenced before assignment
>>>
```

A Turtle Shape Example



- In this example, we first define one function:

  `forward_and_turn_right()`

- This function will be used several times inside two other functions:

  `draw_square()` and `draw_star()`

- This is a clever design because the same task, which is needed by two different functions, is written in one place

The Turtle Shape Example: First Function

- The first function is used to draw a line and turn using a certain length and angle

```
def forward_and_turn_right(length, angle):
    turtle.forward(length)
    turtle.right(angle)
```

Two values are passed into the function, separated by a comma

- This function will be used by two other functions, which will be shown in the next slides

The Turtle Shape Example: Drawing a Square Function

- The second function draws a square with a given length of the sides and colours

```
def draw_square(length, line_colour, fill_colour):
    turtle.color(line_colour, fill_colour)

    turtle.begin_fill()
    for _ in range(4):
        forward_and_turn_right(length, 90)
    turtle.end_fill()
```

The first function is used here to draw a line and turn 90 degrees to the right

The Turtle Shape Example: Drawing a Star Function

- The third function draws a star with a given size and colours

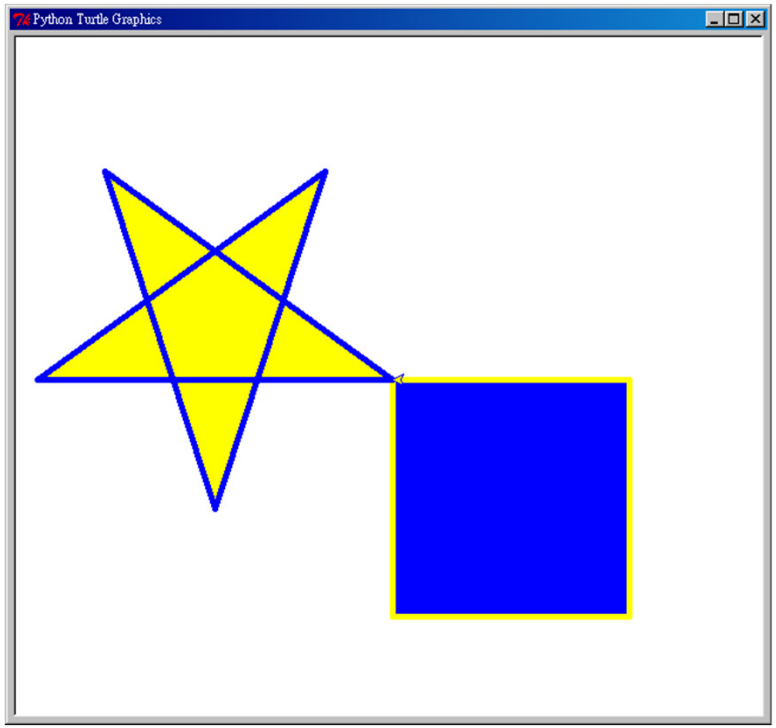
```
def draw_star(length, line_colour, fill_colour):
    turtle.color(line_colour, fill_colour)

    turtle.begin_fill()
    for _ in range(5):
        forward_and_turn_right(length, 144)
    turtle.end_fill()
```

The first function again is used to draw a line but the turtle turns 144 degrees this time

The Shape Example: The Main Part

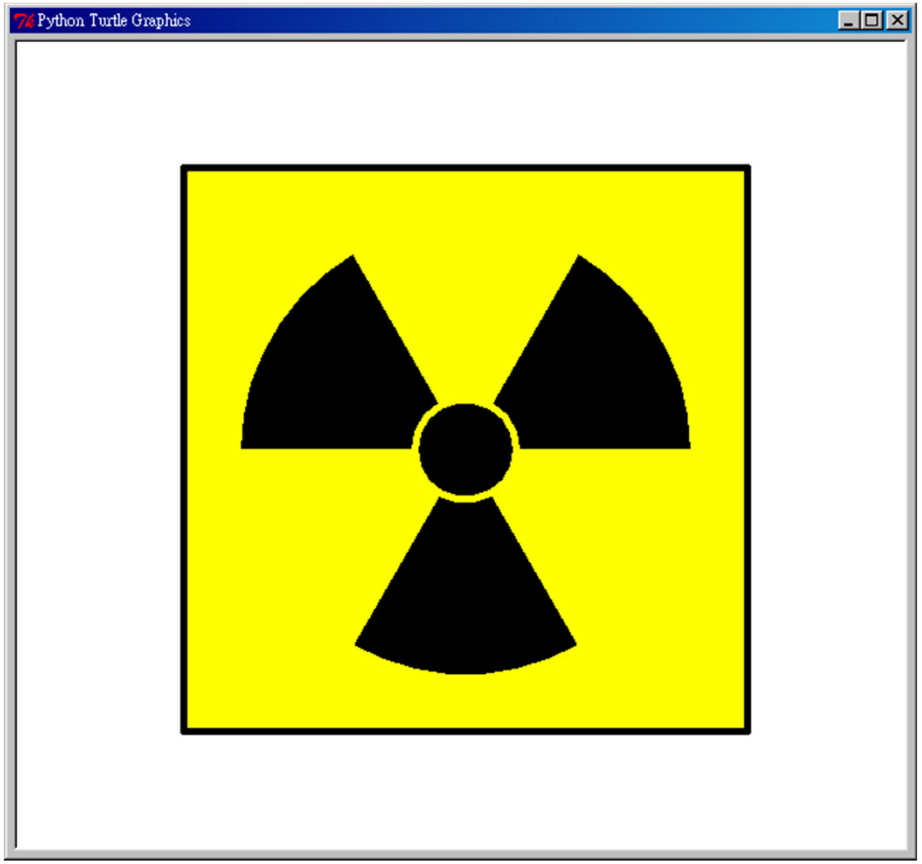
- The main part of the program then uses the `draw_square()` and `draw_star()` functions to draw the two shapes in the turtle window, like this:

```
draw_square(200, \
    "yellow", "blue")

turtle.right(180)

draw_star(300, \
    "blue", "yellow")
```

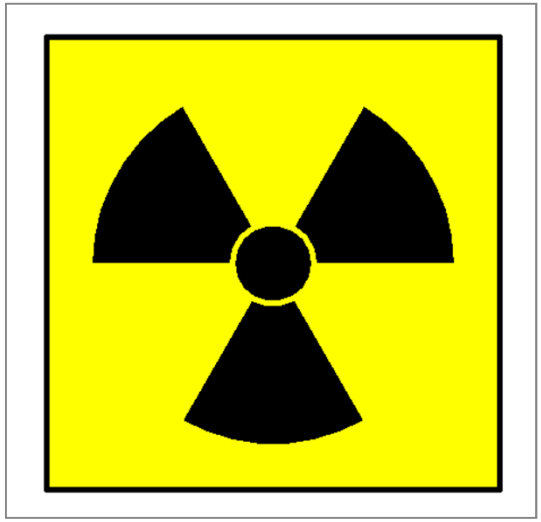


Radioactive Symbol Example

- In the following larger example, we use functions to help create the warning symbol for radioactivity



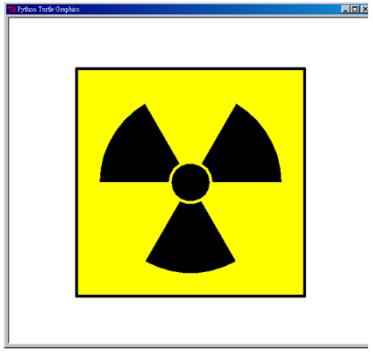
Radioactive Symbol 1/3



```
def square(length):
    # Draw a square of length pixels
    for i in range(4):
        turtle.forward(length)
        turtle.left(90)

def sector(radius, angle):
    # Draw part of a circle
    turtle.forward(radius)
    turtle.left(90)
    turtle.circle(radius, angle)
    turtle.left(90)
    turtle.forward(radius)
    turtle.left(180-angle)
```

```
def move(x, y):
    # Move forward and left
    turtle.up()
    turtle.forward(x)
    turtle.left(90)
    turtle.forward(y)
    turtle.right(90)
    turtle.down()
```

Radioactive Symbol 2/3

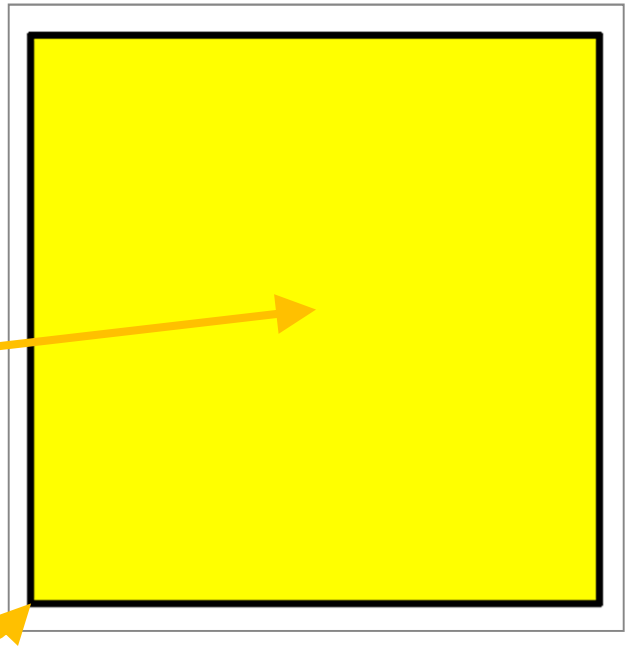


- Remember that, by default, (0, 0) is the middle of the screen

```
def draw_symbol(large_radius, small_radius, side):
    move(-(side/2), -(side/2))

    turtle.color("black", "yellow")
    # Draw outer yellow square
    turtle.begin_fill()
    turtle.width(5)
    square(side)
    turtle.end_fill()

    move(side/2, side/2)

    # Draw the complete symbol
    turtle.color("yellow", "black")
    turtle.width(1)
```

Defined in the previous slide

Defined in the previous slide



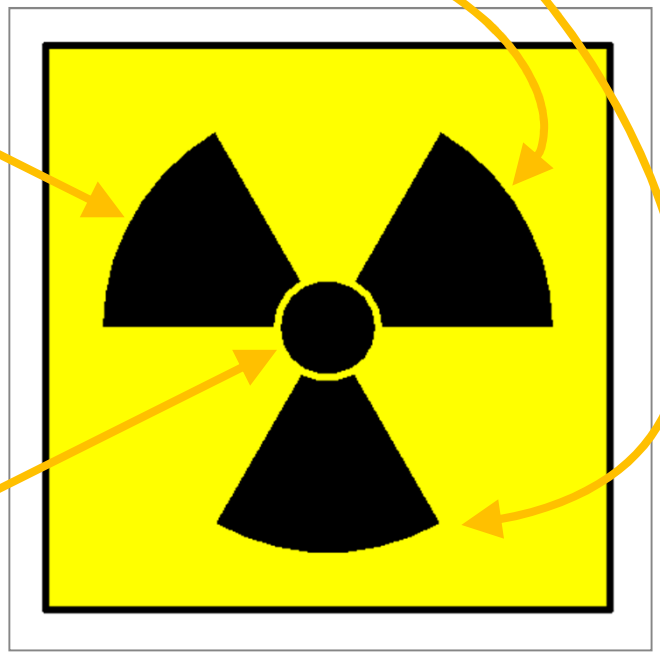
Radioactive Symbol 3/3

function draw_symbol() continued:

```
# Draw three sections
for i in range(3):
    turtle.begin_fill()
    sector(large_radius, 60)
    turtle.left(120)
    turtle.end_fill()

turtle.forward(small_radius)
turtle.left(90)

# Draw centre circle
turtle.width(5)
turtle.begin_fill()
turtle.circle(small_radius)
turtle.end_fill()
```

Defined previously

function draw_symbol() ends here



Defined last/this slide

```
# Main part of program
turtle.reset()
draw_symbol(160, 36, 400)
turtle.hideturtle()
turtle.done()
```

Using the Return Command

- We can use the return command to tell a function to immediately stop running
- For example, we stop the following function from running when the value passed into the function is not valid in the example

```
def donate(money):
    if money <= 0:
        return
```

Stop the function if `money` *is not positive*

```
    print("Thank you! You are so generous!")
```

Stopping a Function Using Return

- Here is the entire program:

```
def donate(money):
    if money <= 0:
        return

    print("Thank you! You are so generous!")

donation = int(input("How much do you donate? "))
donate(donation)
```

```
How much do you donate? -5000
>>>
```

If the `return` *command is executed then the program will immediately continue from this point, i.e. finish the program*

Returning Values from a Function

- Another use of the return command is to return some values from a function, to the place where the function is run
- Usually this is used to return some results from a function
- For example, we can make a square function to calculate and return the square of a number

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

Calculating the Square of a Number

- We can use the square function like this:

```
input_number = \
    int(input("Please give me a number: "))

print("The square of the number is: ", end="")
print(square(input_number))
```

Run the function and print the result

- This is what we get if we enter 25 in the program:

```
Please give me a number: 25
The square of the number is: 625
>>>
```

Returning Multiple Values

- We can also return multiple values by simply separating the values using commas
- The following function returns two values:

```
def get_info(current_year, year_of_birth):
    chinese_zodiac = [
        "Rat", "Ox", "Tiger", "Rabbit",
        "Dragon", "Snake", "Horse", "Sheep",
        "Monkey", "Rooster", "Dog", "Pig"
    ]

    age = current_year - year_of_birth
    zodiac = chinese_zodiac[(year_of_birth - \
             1960) % 12]

    return age, zodiac
```

Two values are returned in this example

Getting Multiple Returned Values

- To get the values from this function, we use a separated list of variables like this:

```
year = int(input("Hi, what is the current year? "))
birthyear = int(input("When is your year of birth? "))

age, zodiac = get_info(year, birthyear)

print("You are", age, "and your zodiac is", zodiac)
```

```
Hi, what is the current year? 2017
When is your year of birth? 1997
You are 20 and your zodiac is Ox
```