COMP1021
Introduction to Computer Science

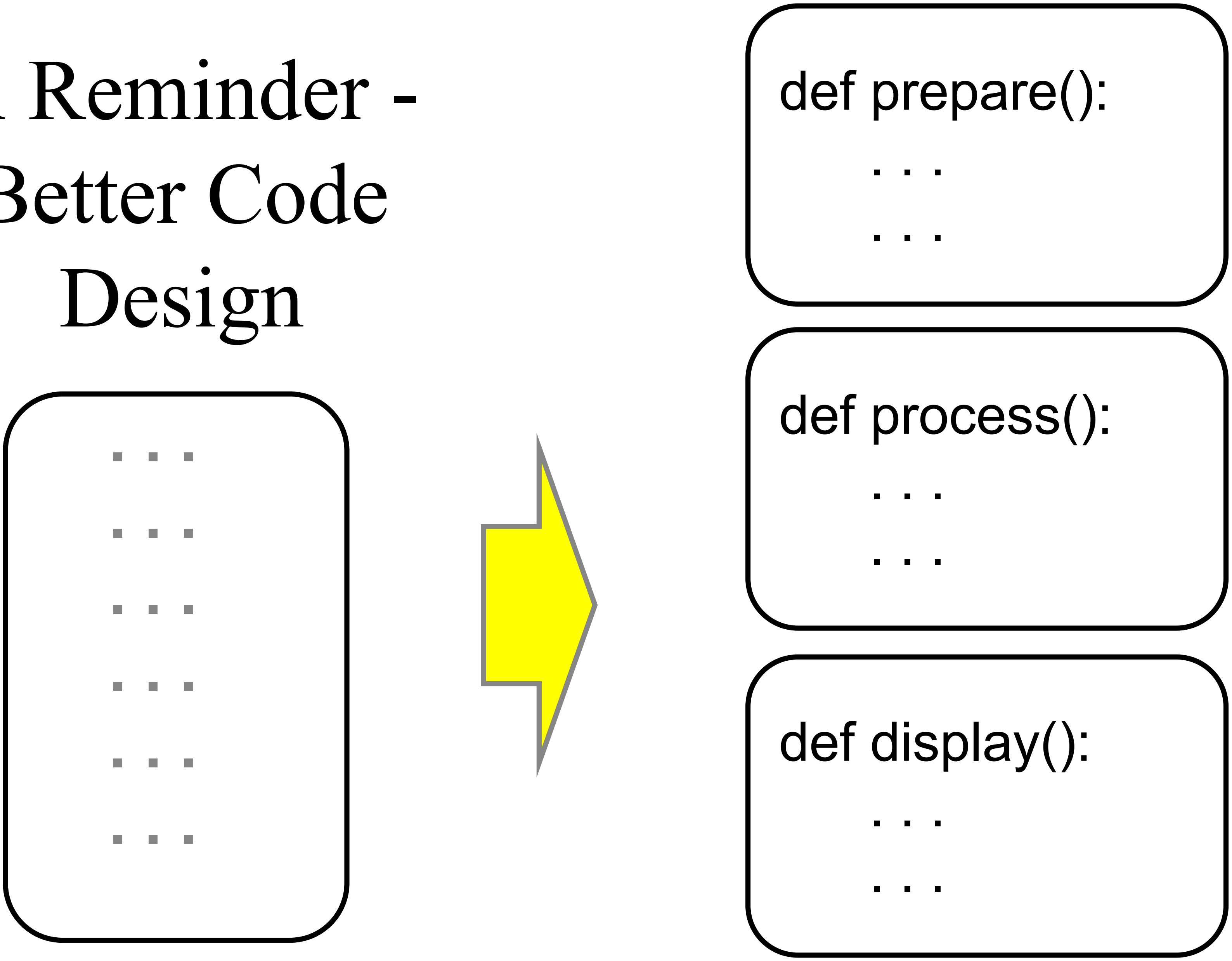
More on Functions

David Rossiter and Gibson Lam

Outcomes

- After completing this presentation, you are expected to be able to:
    1. Return values from a function using *return*
    2. Stop a function by using the return command
    3. Explain the difference between local variables and global variables
    4. Use a global variable to update data between the main part of a program and functions
    5. Pass and return values to functions, to update data between the main part and functions

A Reminder - Better Code Design



```
. . .
. . .
. . .
. . .
. . .
. . .
```

```
def prepare():
    . . .
    . . .
```

```
def process():
    . . .
    . . .
```

```
def display():
    . . .
    . . .
```

One big piece of code is hard to manage

Usually we divide it into several functions, for more efficient handling

A Reminder - Making a Function

- To make a function in Python, we use the def command (**def**ine a function):

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

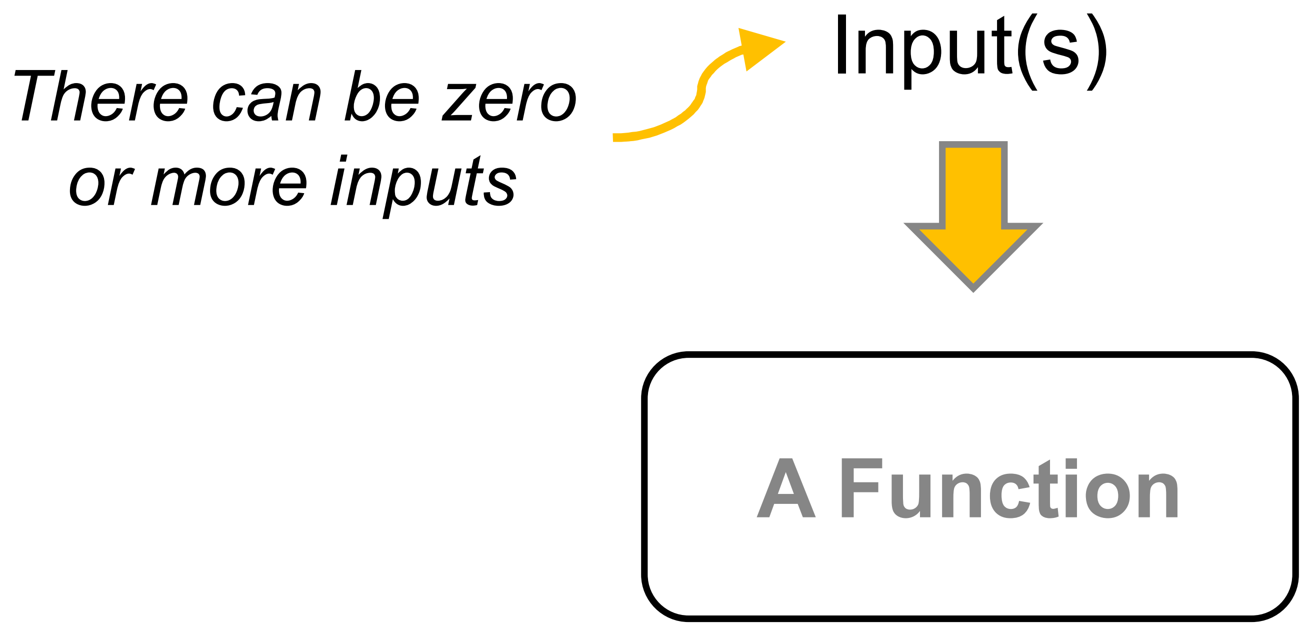
This is the code of the function

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

- Then we can execute the function like this:

```
greeting()
```

A Reminder - Passing Something



A Reminder - Passing Something

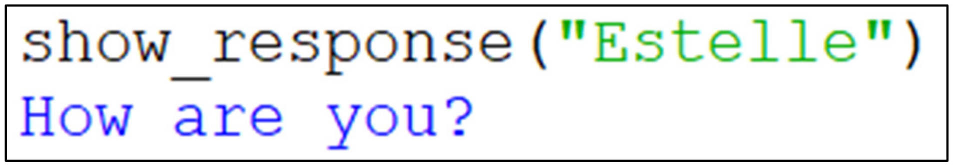
- Sometimes it is useful to give a value to a function, so that it can do different things
- We call that ‘passing values to a function’ in computer science terms
- Here is an example:

In this example, the function expects to receive something, which will be stored in a variable called ‘name’

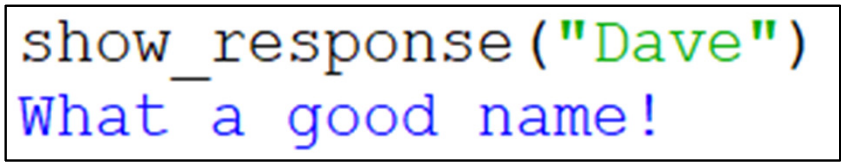
```
def show_response( $\boxed{\text{name}}$ ):
    if name == "Dave":
        print("What a good name!")
    else:
        print("How are you?")
```

A Reminder - Using the Function

- You can pass a value directly to the function, like this:



```
show_response("Estelle")
How are you?
```



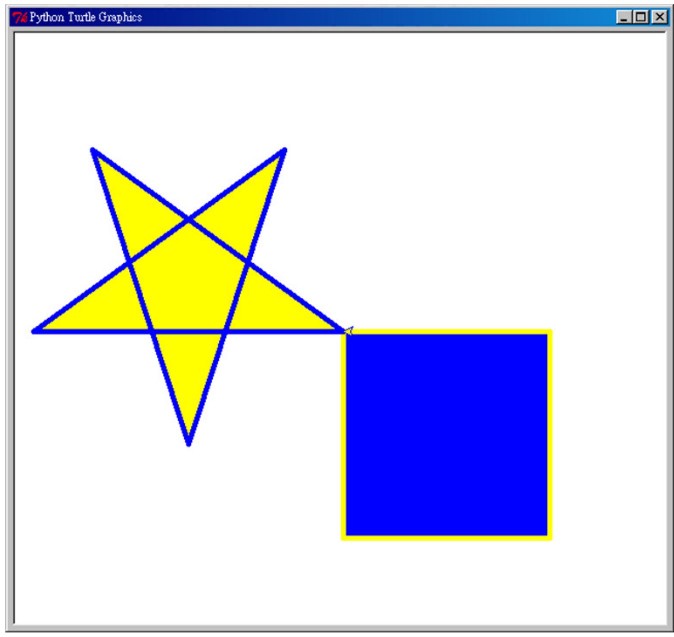
```
show_response("Dave")
What a good name!
```

- Sometimes the value that you pass to the function is first stored in a variable, like this:

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

- Both approaches are common

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 using 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 using a given size and colours

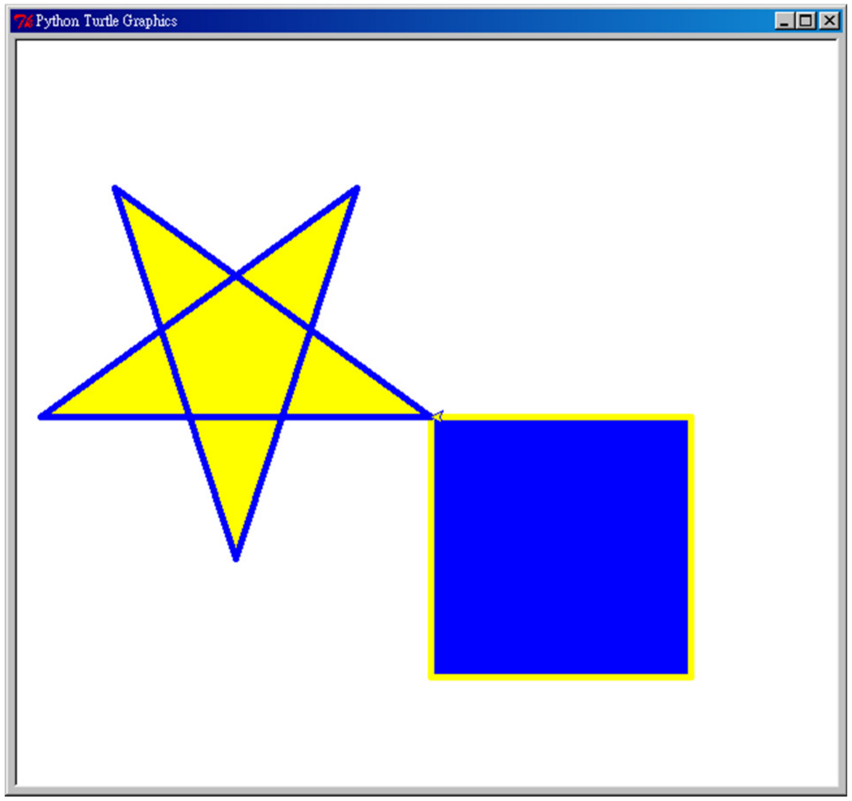
```
def draw_star(length, line_colour, fill_colour):
    turtle.color(line_colour, fill_colour)

    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

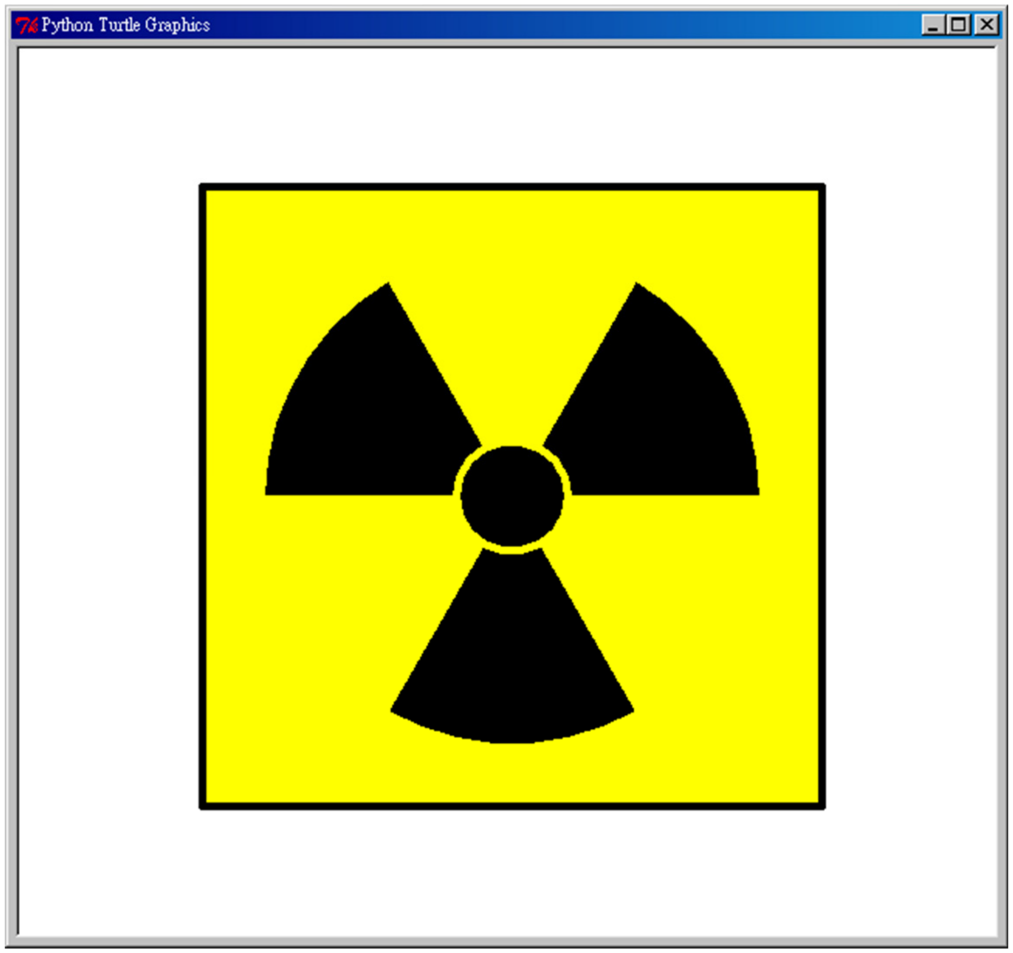
- The main part of the program then uses the `draw_square()` and `draw_star()` functions to draw the two shapes in the turtle window:

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

turtle.right(180)

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

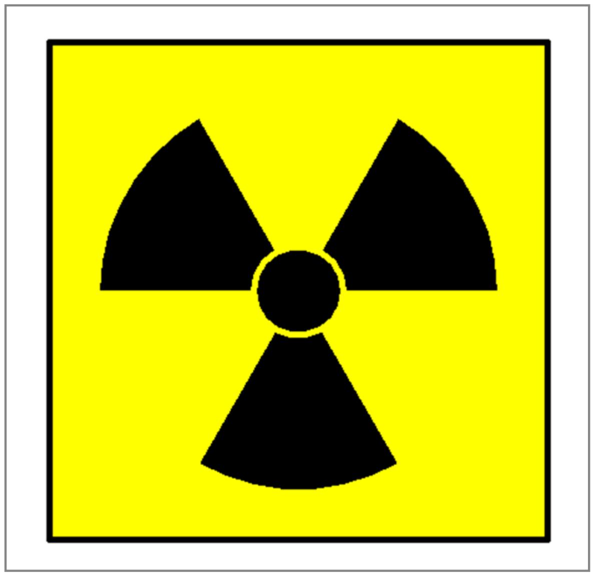


Radioactive Symbol Example

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



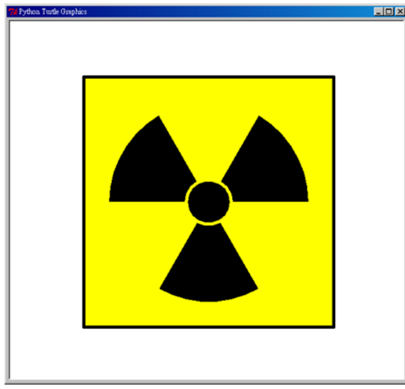
Radioactive Symbol 1/3



```
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

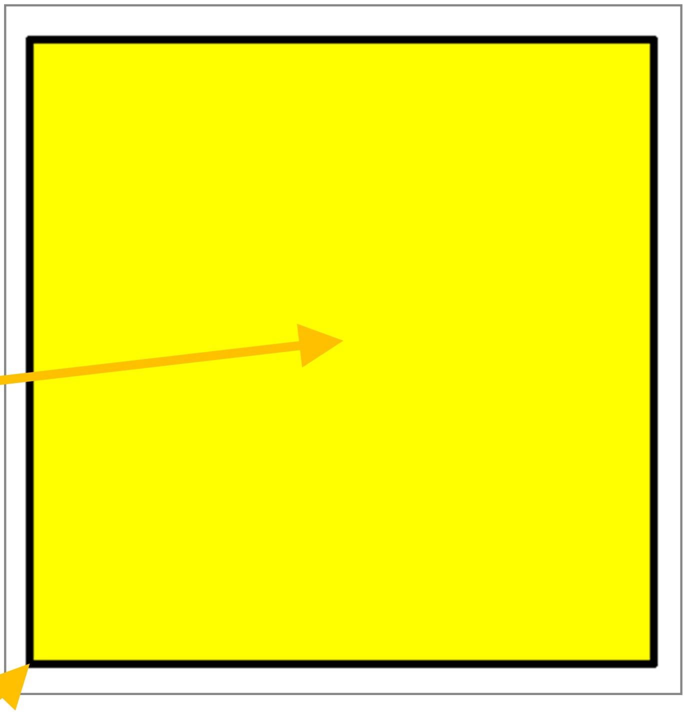
```
def draw_symbol(large_radius, small_radius, side):
    move(-(side/2), -(side/2))
```

} *Defined in the previous slide*

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

} *Defined in the previous slide*

```
    move(side/2, side/2)

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



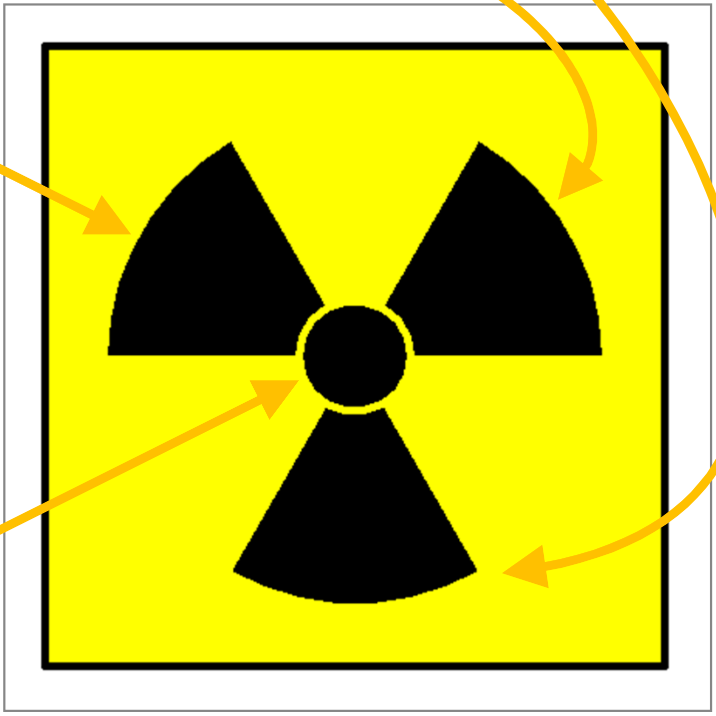
Radioactive Symbol 3/3

function draw_symbol() continued:

Defined previously

```
    # 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()
```

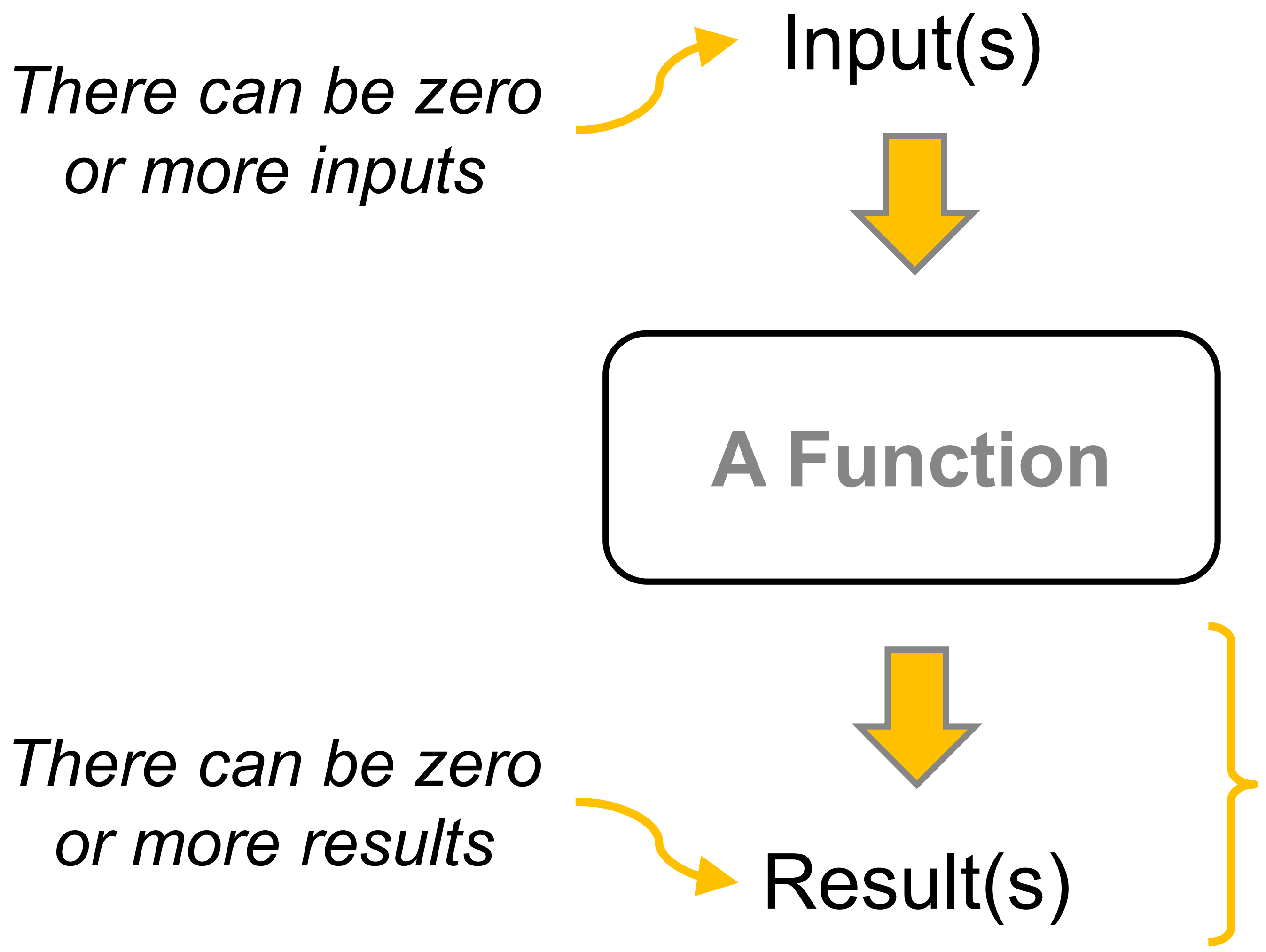
function draw_symbol() ends here



Defined last/this slide

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

A Python Function


There can be zero or more inputs
Input(s)
A Function
There can be zero or more results
Result(s)

- Now we will consider this, which uses the `return` command

Returning Values from a Function

- The `return` command is usually used to return one or more values from a function
- The value(s) go from the function to the place where the function was executed
- For example, we can make a square function to calculate and return the square of a number

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

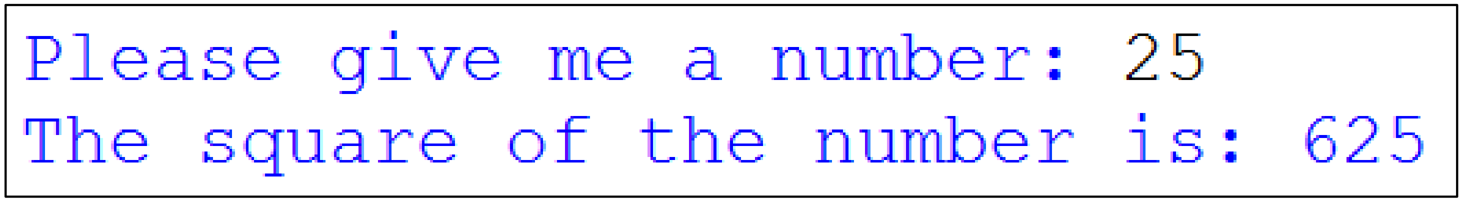
Calculating the Square of a Number

- Then 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:



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

Returning Multiple Things

- We can return more than one thing
- E.g. the following function returns two values:

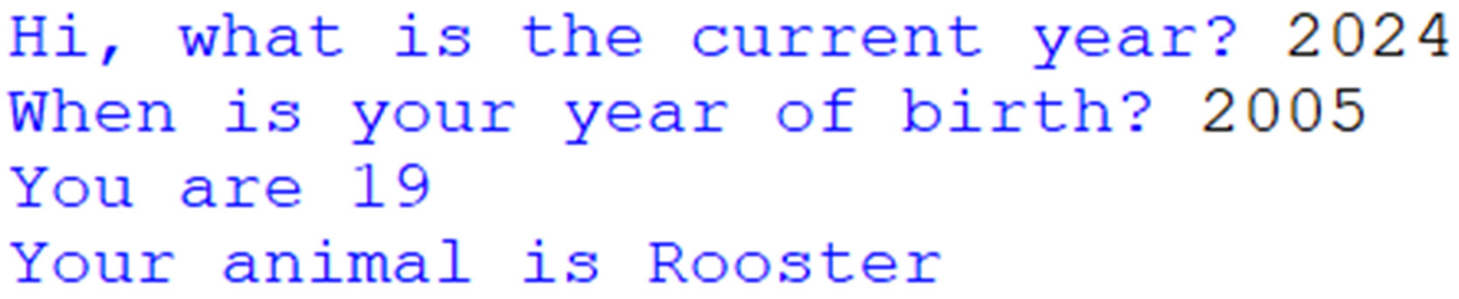
```
def get_info(current_year, year_of_birth):
    chinese_zodiac = [
        "Rat", "Ox", "Tiger", "Rabbit",
        "Dragon", "Snake", "Horse", "Sheep",
        "Monkey", "Rooster", "Dog", "Pig"
    ]

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

    return age, animal
```

Two values are returned in this example

Getting Multiple Results

- To get the two results from the function we use two variables, like this:

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

yourage, youranimal = get_info(year, birthyear)

print("You are", yourage)
print("Your animal is", youranimal)
```



```
Hi, what is the current year? 2024
When is your year of birth? 2005
You are 19
Your animal is Rooster
```

Using the Return Command

- Whenever the `return` command is used the function will immediately stop running
- `return` doesn't actually have to return anything
- For example, here we stop the function when the value passed to the function is not appropriate:

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

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

If `money` *is not positive then stop the function here*

Stopping a Function Using Return

The complete program:

```
How much do you donate? -5000
Finished!
```

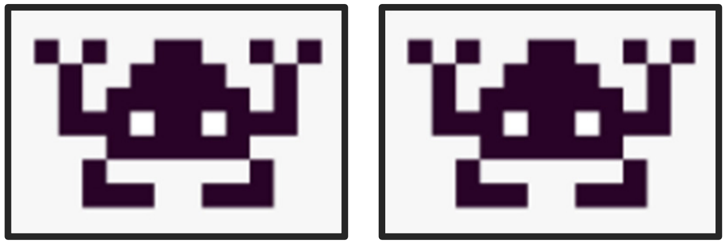
```
How much do you donate? 100
Thank you! You are so generous!
Finished!
```

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


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


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


print("Finished!")
```

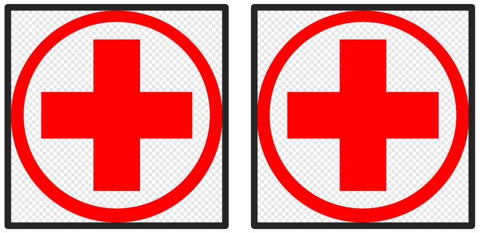
If the `return` *command is executed then the function immediately finishes, and Python continues with any code under the place where the function was executed*

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 is given to the local variable



A Game



- Let’s imagine you are developing a game
- The user has to shoot monsters,
  but cannot shoot boxes of medicine
- If a monster is shot, the player gets 100 points
- But if a box of medicine is shot,
  the player loses 500 points
- We need to make sure that the score is updated correctly

Sharing Data

```
def shoot_monster():
    . . .
    # Increase score by 100
    . . .
```

```
def shoot_medicine():
    . . .
    # Decrease score by 500
    . . .
```

```
# Main part of program
. . .
# Set score to zero
. . .
```

- The score needs to be changed in the functions and also the main part
- How can we handle it?
- Let's look at 2 approaches

Approach 1

```
# Main part of program
. . .
score = 0
. . .
shoot_monster()
. . .
shoot_medicine()
. . .
```

```
def shoot_monster():
    global score
    . . .
    score = score + 100
    . . .
```

```
def shoot_medicine():
    global score
    . . .
    score = score - 500
    . . .
```

- In the approach shown here the variable *score* is shared by the functions and the main part

Approach 1

```
# Main part of program
. . .
reset_score()
. . .
shoot_monster()
. . .
shoot_medicine()
. . .
```

- The main part of the program doesn’t actually have to refer to the variable in any way
- Even if it doesn’t, this approach will still work

```
def reset_score():
    global score
    score = 0
```

```
def shoot_monster():
    global score
    . . .
    score = score + 100
    . . .
```

```
def shoot_medicine():
    global score
    . . .
    score = score - 500
    . . .
```

Approach 2

```
# Main part of program
. . .
score = 0
. . .
score = shoot_monster(score)
. . .
score = shoot_medicine(score)
. . .
```

```
def shoot_monster(sc):
    . . .
    sc = sc + 100
    . . .
    return sc
```

```
def shoot_medicine(sc):
    . . .
    sc = sc - 500
    . . .
    return sc
```

- Here we pass the current value to the function, then the function changes the value and returns it, and the returned value goes back into the variable

Using Variables with the Same Name

- Let’s consider this example:

The `name` *variable here contains the value passed to the function*

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

This `name` *is a different variable, even though it looks the same*

```
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
- Even though the variables have the same name, in this example they are **two different variables**

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 == "Dave":
        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

- Having the same name for local and global variables is very confusing - we should use different names, for example:

```
def magic_mirror(name):
    if name == "Dave":
        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:

```
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 `money` *is changed in this line of code*

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

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

Changing Global Variables inside a Function

- If you want a 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 when we refer to `money` *in the function, it means the global variable* `money`

This line changes the value of the global variable

Running the Example

- This 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
```