

# Functions in Python

---

Distributed Network and Systems Laboratory,  
Chonnam National University  
Presenter : Shivani Kolekar



- ✓ **Introduction to List**
- ✓ **Types of Elements**
- ✓ **Accessing and Modifying List Elements**
- ✓ **Introduction to Other Data Structures (Tuples etc.)**
- ✓ **Q & A**

# *Today's Goals*

- ✓ **Introduction to Functions**
- ✓ **Types of Functions**
- ✓ **Defining and calling Functions**
- ✓ **Local Variables**
- ✓ **Passing Arguments**
- ✓ **Global Variables**
- ✓ **Q & A**

# *Introduction to Functions*

**CONCEPT:** A **function** is a group of statements that exist within a program for the purpose of performing a specific task.

- Most programs perform tasks that are large enough to be broken down into several subtasks.
- We can **break down a program into small manageable pieces** known as functions.

***Divide and conquer!***

# *Introduction to Functions*

- When using functions in a program, you generally isolate each task within the program in its own function.
- A program that has been written with each task in its own function is called a modularized program.

## Using Functions to divide and conquer a large task

This program is one long, complex sequence of statements.

[illegible]

In this program the task has been divided into smaller tasks, each of which is performed by a separate function.

```
def function1():
    statement
    statement
    statement
```

```
def function2():
    statement
    statement
    statement
```

```
def function3():
    statement
    statement
    statement
```

```
def function4():
    statement
    statement
    statement
```

# *Void Functions and Value Returning Functions*

The benefits of using functions include:

- Simpler code
  - Code reuse
  - write the code once and call it multiple times
- 
- ✓ **Better testing and debugging**
    - Can test and debug each function individually
- 
- ✓ **Faster development**
- 
- ✓ **Easier facilitation of teamwork**
    - Different team members can write different functions.

# ***Void Functions and Value Returning Functions***

- **A void function:**  
Simply executes the statements it contains and then terminates.
- **A value-returning function:**  
Executes the statements it contains, and then it returns a value back to the statement that called it.  
The **input, int, and float functions** are examples of **value-returning functions**.



# Functions Example

## Problem to solve:

Lets consider that John is hungry and wants to make himself a smoothie and a sandwich. Write a program to give John recipe instructions to make a smoothie and a sandwich.

### Functions needed for making breakfast:

- 1) Function for instructions to **make a smoothie** .
- 2) Function for instructions to **make a sandwich**.
- 3) Function for **user input**, whether he wants recipe for smoothie or sandwich.

```
def function_name() :  
    statement  
    statement
```

- Functions are given names.

- **Function naming rules:**

1. Cannot use key words as a function name
2. Cannot contain spaces
3. First character must be a letter or underscore
4. All other characters must be a letter, number or underscore
5. Uppercase and lowercase characters are distinct

```
def function_name() :  
    statement  
    statement
```

# Function Example and explanation

## function\_demo.py

```
1 # This program demonstrates a function.
2 # First, we define a function named message.
3 def message():
4     print('I am Arthur,')
5     print('King of the Britons.')
6
7 # Call the message function.
8 message()
```

### Program Output

```
I am Arthur,
King of the Britons.
```

### The function definition and the function call

These statements cause  
the message function to  
be created.

```
# This program demonstrates a function.
# First, we define a function named message.
def message():
    print('I an Arthur,')
    print('King of the Britons.')
```

```
# Call the message function.
message()
```

This statement calls  
the message function,  
causing it to execute.

# Two functions in one program

## Two\_functions.py

```
# This program has two functions. First we
# define the main function.
def main():
    print('I have a message for you.')
    message()
    print('Goodbye!')

# Next we define the message function.
def message():
    print('I am Arthur,')
    print('King of the Britons.')

# Call the main function.
main()
```

### Program Output

```
I have a message for you.
I am Arthur,
King of the Britons.
Goodbye!
```



### Calling the main function

The interpreter jumps to the main function and begins executing the statements in its block.

```
# This program has two functions. First we
# define the main function.
def main():
    print('I have a message for you.')
    message()
    print('Goodbye!')

# Next we define the message function.
def message():
    print('I am Arthur,')
    print('King of the Britons.')

# Call the main function.
main()
```

# Two functions in one program (continued..)

## Two\_functions.py

```
# This program has two functions. First we
# define the main function.
def main():
    print('I have a message for you.')
    message()
    print('Goodbye!')

# Next we define the message function.
def message():
    print('I am Arthur,')
    print('King of the Britons.')

# Call the main function.
main()
```

### Program Output

```
I have a message for you.
I am Arthur,
King of the Britons.
Goodbye!
```



### Calling the message function

```
# This program has two functions. First we
# define the main function.
def main():
    print('I have a message for you.')
    message()
    print('Goodbye!')

# Next we define the message function.
def message():
    print('I am Arthur,')
    print('King of the Britons.')

# Call the main function.
main()
```

The interpreter jumps to the message function and begins executing the statements in its block.

# Two functions in one program (continued.)

## Two\_functions.py

- That is the end of the main function, so the function returns as shown here. There are no more statements to execute, so the program ends.



### The message function returns

When the message function ends, the interpreter jumps back to the part of the program that called it and resumes execution from that point.

```
# This program has two functions. First we
# define the main function.
def main():
    print('I have a message for you.')
    message()
    print('Goodbye!')

# Next we define the message function.
def message():
    print('I am Arthur,')
    print('King of the Britons.')

# Call the main function.
main()
```

### The main function returns

When the main function ends, the interpreter jumps back to the part of the program that called it. There are no more statements, so the program ends.

```
# This program has two functions. First we
# define the main function.
def main():
    print('I have a message for you.')
    message()
    print('Goodbye!')

# Next we define the message function.
def message():
    print('I am Arthur,')
    print('King of the Britons.')

# Call the main function.
main()
```

# Indentation in Python

- In Python, each line in a block must be indented.

All of the statements in a block are indented

The last indented line is  
the last line in the block.

```
def greeting():  
    print('Good morning!')  
    print('Today we will learn about functions.')
```

These statements  
are not in the block.

```
print('I will call the greeting function.')
```

```
greeting()
```

# POINTS TO REMEMBER

- ✓ **Two Parts of a Function Definition:**

A function definition has a header and a body.

- ✓ **Calling a Function:**

Means telling the program to execute the set of actions defined within the function.

It's done by using the function name followed by parentheses, which may include arguments if the function requires them.

- ✓ **End of a Function's Block:**

When the end of the function's block is reached during execution, the function finishes, and the program execution returns to the point where the function was called.

- ✓ **Indenting Statements in a Block:**

You must indent the statements in a block to define the scope of the block in Python.



# Local Variables

**CONCEPT:** A **local variable** is created inside a function and cannot be accessed by statements that are outside the function.

- ✓ Different functions can have local variables with the same names because the functions cannot see each other's local variables.
- ✓ An error will occur if a statement in one function tries to access a local variable that belongs to another function.

# Scope and Local Variables

- A variable's scope → part of a program in which the variable may be accessed.
- A variable is visible only to statements in the variable's scope.

```
# This program demonstrates two functions that
# have local variables with the same name.

def main():
    # Call the texas function.
    texas()
    # Call the california function.
    california()

# Definition of the texas function. It creates
# a local variable named birds.
def texas():
    birds = 5000
    print('texas has', birds, 'birds.')

# Definition of the california function. It also
# creates a local variable named birds.
def california():
    birds = 8000
    print('california has', birds, 'birds.')

# Call the main function.
main()

texas has 5000 birds.
california has 8000 birds.
```

- Each function has its own birds variable

```
def texas():
    birds = 5000
    print('texas has', birds, 'birds.')
```

birds → 5000

```
def california():
    birds = 8000
    print('california has', birds, 'birds.')
```

birds → 8000

# Passing Arguments to Functions

## Argument :

- ✓ Piece of data that is sent into a function.
- ✓ Function can use argument in calculations.
- ✓ When calling the function, the argument is placed in parentheses following the function name.

```
1  # This program demonstrates an argument being
2  # passed to a function.
3
4  def main():
5      value = 5
6      show_double(value)
7
8  # The show_double function accepts an argument
9  # and displays double its value.
10 def show_double(number):
11     result = number * 2
12     print(result)
13
14 # Call the main function.
15 main()
```

### Program Output

10

# Passing Arguments to Functions: Example and explanation

```
1 # This program demonstrates an argument being
2 # passed to a function.
3
4 def main():
5     value = 5
6     show_double(value)
7
8 # The show_double function accepts an argument
9 # and displays double its value.
10 def show_double(number):
11     result = number * 2
12     print(result)
13
14 # Call the main function.
15 main()
```

## Program Output

10

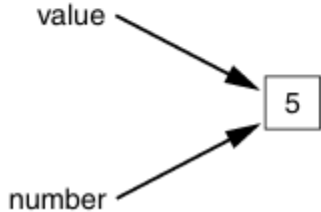
## The value variable is passed as an argument

```
def main():
    value = 5
    show_double(value)
    ↓
def show_double(number):
    result = number * 2
    print(result)
```

## The value variable and the number parameter reference the same value

```
def main():
    value = 5
    show_double(value)

def show_double(number):
    result = number * 2
    print(result)
```



## Passing Arguments to Functions: Example 2

### ❑ Problem to solve:

Your friend Michael runs a catering company.

Some of the ingredients that his recipes require are measured in cups.

When he goes to the grocery store to buy those ingredients, however, they are sold only by the fluid ounce.

✓ He has asked you to write a simple program that converts cups to fluid ounces.

▪ *formula* : 1 cup = 8 fluid ounces

# Passing Arguments to Functions: Example 2

## cups\_to\_ounces.py

```
# This program converts cups to fluid ounces.

def main():
    intro() #calling intro function
    cups_needed = int(input('Enter the number of cups: '))
    cups_to_ounces(cups_needed) #convert cups to ounces

# The intro function displays an introductory screen.
def intro():
    print('This program converts measurements in cups to fluid ounces.')
    print('For your reference the formula is:')
    print('1 cup = 8 fluid ounces')

# The cups_to_ounces function accepts a number of
# cups and displays the equivalent number of ounces.
def cups_to_ounces(cups):
    ounces = cups * 8
    print('That converts to', ounces, 'ounces.')

# Call the main function.
main()
```

```
This program converts measurements in cups to fluid ounces.
For your reference the formula is:
1 cup = 8 fluid ounces
Enter the number of cups: 2
That converts to 16 ounces.
```

## Output

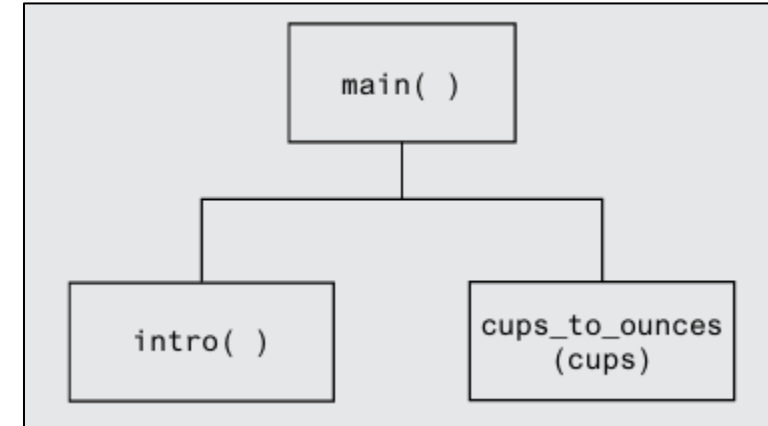


Fig: hierarchy chart of functions in this example

Functions created are:

- ✓ **intro()**
- ✓ **cups\_to\_ounces()**
- ✓ **main()**

# Passing Multiple Arguments

- ✓ Often it's useful to write functions that can accept multiple arguments.
- ✓ Program Add\_numbers.py shows a function named add\_numbers, that accepts two arguments.
- ✓ The function adds the two arguments and displays their addition.

## Add\_numbers.py

```
def main():  
    num1 = int(input('Add first number for addition: '))  
    num2 = int(input('Add second number for addition: '))  
    add_numbers(num1, num2)  
  
def add_numbers(n1,n2):  
    result = n1 + n2  
    print ('Addition is:', result)  
  
main()  
  
Add first number for addition: 80  
Add second number for addition: 20  
Addition is: 100
```



# Passing String Arguments

- ✓ Program Add\_numbers.py shows a function named reverse\_name, that accepts two string arguments.

## String\_Arguments.py

```
# This program demonstrates passing two string
# arguments to a function.

def main():
    first_name = input('Enter your first name: ')
    last_name = input('Enter your last name: ')
    print('Your name reversed is:')
    reverse_name(first_name, last_name)

def reverse_name(first, last):
    print(last, first)
# Call the main function.
main()
```

```
Enter your first name: Harry
Enter your last name: Potter
Your name reversed is:
Potter Harry
```

# Global Variables

# global Variables

- A global variable is accessible to all the functions in a program file.
- When a variable is created by an assignment statement that is written outside all the functions in a program file, the **variable is global**.

## Global\_Var\_example1.py

```
# This program demonstrates use of global variables

#create a global variable
my_value = 10

# The show_value function prints
# the value of the global variable.
def show_value():
    print(my_value)

# Call the show_value function.
show_value()

10
```

- ✓ The `global number` statement tells the interpreter that the main function intends to assign a value to the global number variable.
- ✓ The value entered by the user is assigned to number .

## Global\_Var\_example2.py

```
# Create a global variable.  
  
number = 0  
def main():  
    global number  
    number = int(input('Enter a number: '))  
    show_number()  
  
def show_number():  
    print('The number you entered is', number)  
  
# Call the main function.  
main()  
  
Enter a number: 55  
The number you entered is 55
```

**Lets Code!**