# CSC101 – Introduction to ICT
# Lecture 9-10
# Functions

Course Instructors

**Dr. Majid Iqbal Khan** **Dr. Ashfaq Hussain Farooqi** **Mr. Rizwan Rashid** **Ms. Sajida Kalsoom**

# Agenda

- Why use functions?
- What are functions?
  - Example
- The scope of variables
- Passing arguments by reference values
- Modularizing Code

# Why use functions?

```
start, end = 35, 49
sum = 0
for i in range(start, end+1):
        sum += i
print("Sum from", start, "to",
                    end, "is", sum)
```

- Write a program that find the sum of integers from **1** to **10**, **20** to **37**, and **35** to **49**

```
start, end = 1, 10
sum = 0
for i in range(start, end+1):
        sum += i
print("Sum from", start, "to",
                    end, "is", sum)
```

```
start, end = 1, 10
su
fo

pr
```

```
start, end = 20, 37
sum = 0
for i in range(start, end+1):
        sum += i
print("Sum from", start, "to",
                    end, "is", sum)
```

# Why use functions?

- Write a program that find the sum of integers from **1** to **10**, **20** to **37**, and **35** to **49**

```
start, end = 1, 10
sum = 0
for i in range(start, end+1):
        sum += i
print("Sum from", start, "to",
                        end, "is", sum)
```

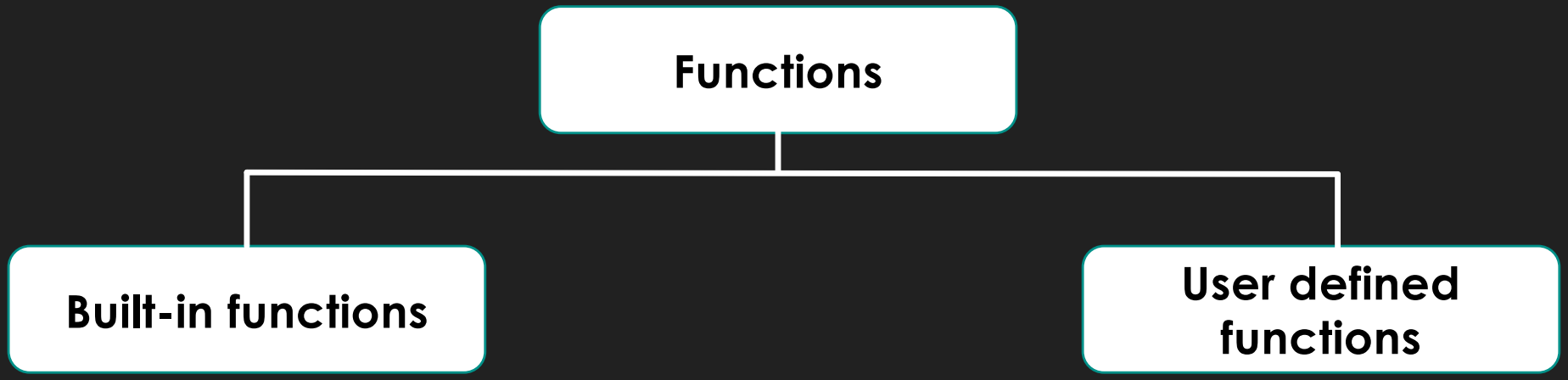
```
# Taking sum of 1st series
sum, start, end = 0, 1, 10
for i in range(start, end+1):
        sum += i
print("Sum from 1 to 10 is", sum)

# Taking sum of 2nd series
sum, start, end = 0, 20, 38
for i in range(start, end+1):
        sum += i
print("Sum from 20 to 37 is", sum)

# Taking sum of 3rd series
sum, start, end = 0, 35, 50
for i in range(start, end+1):
        sum += i
print("Sum from 35 to 49 is", sum)
```

One don't want to repeat the same lines of code.
Specially when you need to add 100 such series?

While writing code follow the concept of **DRY – Don't repeat yourself**
**Functions** are just the right solution because they **are reusable**

# What are functions?

- A ***function*** *is a collection of statements grouped together that performs an operation.*
- *For example;* print(), input(), and eval()



# What are functions?

*A function definition consists of the function's name, parameters, and body.*

- A ***function*** *is a collection of statements grouped together that performs an operation.*
- *For example;* print(), input(), and eval()

## Function definition

**def** keyword is used to define a function, followed by a function name, parenthesis and :

```
def fucntionName(list of parameters/arguments):
    #Function  body
```

- A parameter is like a placeholder: When a function is called / invoked, value is passed to the parameter
- A function may take zero or many parameter
- A function may or may not return a value

## Calling a function

*A function is called by its name and passing desired parameters to it*

**print ("num = ", 100)** – passing two parameter of type string and integer

Name = **input ("Enter Name")** – passing one parameter of type string and it also returns value of type string

num = **random.random()** – passing zero parameter and it returns a value of type float

- def sum(x,y):
  - result = x + y
  - print (result)
- def main():
  - sum(2,3)
  - print("this is the main function of the program")
  - print(" here we are again going to call sum function")
  - sum(12,30)
- main()

# Example 1

```
# Taking sum of 1st series
sum, start, end = 0, 1, 10
for i in range(start, end+1):
        sum += i
print("Sum from 1 to 10 is", sum)

# Taking sum of 2nd series
sum, start, end = 0, 20, 38
for i in range(start, end+1):
        sum += i
print("Sum from 20 to 37 is", sum)

# Taking sum of 3rd series
sum, start, end = 0, 35, 50
for i in range(start, end+1):
        sum += i
print("Sum from 35 to 49 is", sum)
```

```
# Definition of the function sum
# Take two parameters and returns a value
def sum(i1, i2):
    result = 0
    for i in range(i1, i2 + 1):
        result += i
    return result  # returning a value

# Definition of the function main
def main():
    # calling the fucntion sum with parameters 1 and 10
    print("Sum from 1 to 10 is", sum(1, 10))
    # calling the fucntion sum with parameters 20 and 37
    print("Sum from 20 to 37 is", sum(20, 37))
    # calling the fucntion sum with parameters 35 and 49
    print("Sum from 35 to 49 is", sum(35, 49))

main() # Call the main function
```

# Example 2

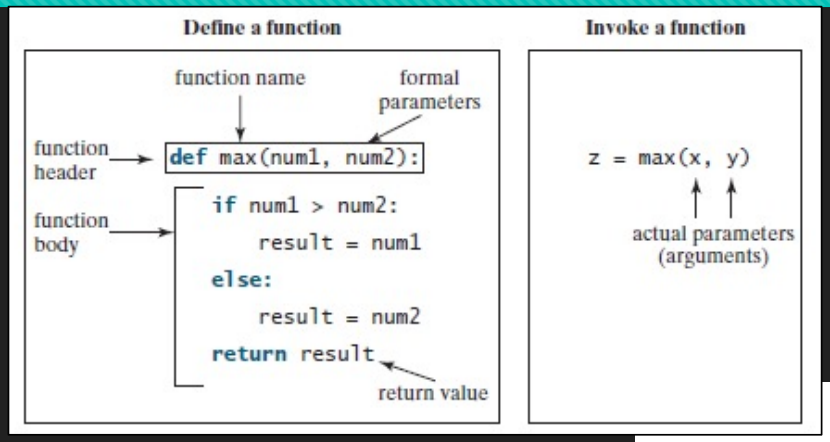
- Create a function that takes two numbers as parameter (arguments) and return the larger number among the two.

```
# definition of max function
# Returns the max of two numbers
def max(num1, num2):
    if num1 > num2:
        result = num1
    else:
        result = num2

    return result

def main():
    x = 5
    y = 2
    z = max(x, y)    # Call the max function
    print("The larger number of", x, "and", y, "is", z)

main() # Call the main function
```

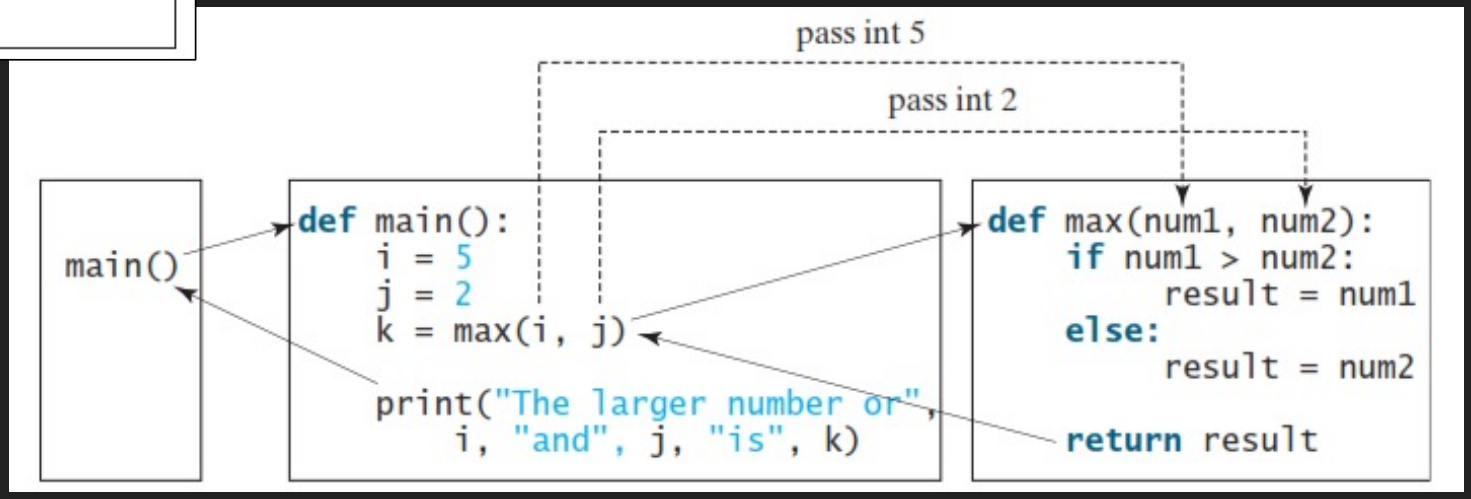
```
The larger number of 5 and 2 is 5
```

# Discussion

```
# definition of max function
# Returns the max of two numbers
def max(num1, num2):
    if num1 > num2:
        result = num1
    else:
        result = num2

    return result

def main():
    x = 5
    y = 2
    z = max(x, y)    # Call the max function
    print("The larger number of", x, "and", y, "is", z)

main() # Call the main function
```




The program that calls the function is called a *caller. In this example main() is caller function.*



# Example 3: Functions with/without return values

- *A program that prints the grade corresponding to the scores obtained by the student.*

```python
# Print grade for the score
def printGrade(score):
    if score >= 90.0:
        print('A')
    elif score >= 80.0:
        print('B')
    elif score >= 70.0:
        print('C')
    elif score >= 60.0:
        print('D')
    else:
        print('F')

def main():
    score = eval(input("Enter a score: "))
    print("The grade is ", end = " ")
    printGrade(score)

main() # Call the main function
```

```python
# Return the grade for the score
def getGrade(score):
    if score >= 90.0:
        return 'A'
    elif score >= 80.0:
        return 'B'
    elif score >= 70.0:
        return 'C'
    elif score >= 60.0:
        return 'D'
    else:
        return 'F'

def main():
    score = eval(input("Enter a score: "))
    print("The grade is", getGrade(score))

main() # Call the main function
```

# Example 4: Functions with multiple return values

- *The Python return statement can return multiple values.*

```
def sort(number1, number2):
    if number1 < number2:
        return number1, number2
    else:
        return number2, number1

def main():
    n1, n2 = sort(13, 2)
    print("n1 is", n1)
    print("n2 is", n2)

main()
```

# Exercise

- Write a program that takes a number as input from the user and provide following options to the user. Your code should have following functions
  - isEven (num) - returns bool value true/false
  - isPositive (num) - returns bool value true/false
  - Factorial (num) - calculate and returns the factorial of the number

# CSC101 – Introduction to ICT
# Lecture 9-10
# Functions

Course Instructors

**Dr. Majid Iqbal Khan** **Dr. Ashfaq Hussain Farooqi** **Mr. Rizwan Rashid** **Ms. Sajida Kalsoom**

# Agenda

- Why use functions?
- What are functions?
  - Example
- The scope of variables
- Passing arguments by reference values
- Modularizing code
- Summary

# The Scope of Variables

- A variable created inside a function is referred to as a *local variable*.
  - Local variables can only be accessed within a function.
  - The scope of a local variable starts from its creation and continues to the end of the function that contains the variable.
- In Python, you can also use *global variables*. They are created outside all functions and are accessible to all functions in their scope.

# The Scope of Variables

```
globalVar = 1 # global variable defined outside of functions

def func():
    localVar = 2     # local variable in func()
    print ("Inside func - Global variable = ", globalVar)
    print ("Inside func - Local variable = ", localVar)

def main():
    func()
    print ("Inside main - Global variable = ", globalVar)
#   Following line will generate an error
#   print ("Inside main - Local variable = ", localVar)

main()
```

globalVar = 1

Scope of the func()
localVar = 2

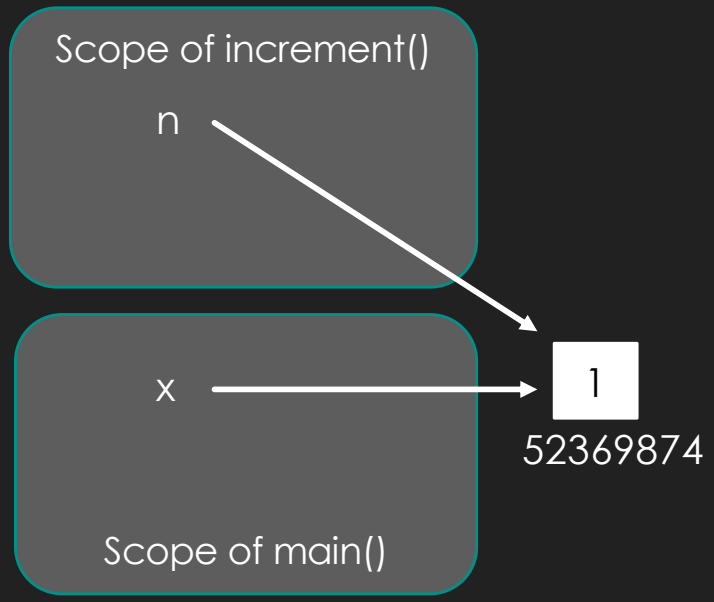
Scope of the main()
localVar is out of scope for main()

# Passing Arguments by Reference Values

- *When you invoke a function with arguments, each argument's reference is passed by value to the parameter in the function.*

```
def increment(n):
    n += 1
    print("Inside the increment function n = ", n)

def main():
    x = 1
    print("Before the call, x is", x)
    increment(x)
    print("After the call, x is", x)

main() # Call the main function
```

Before the call, x is 1

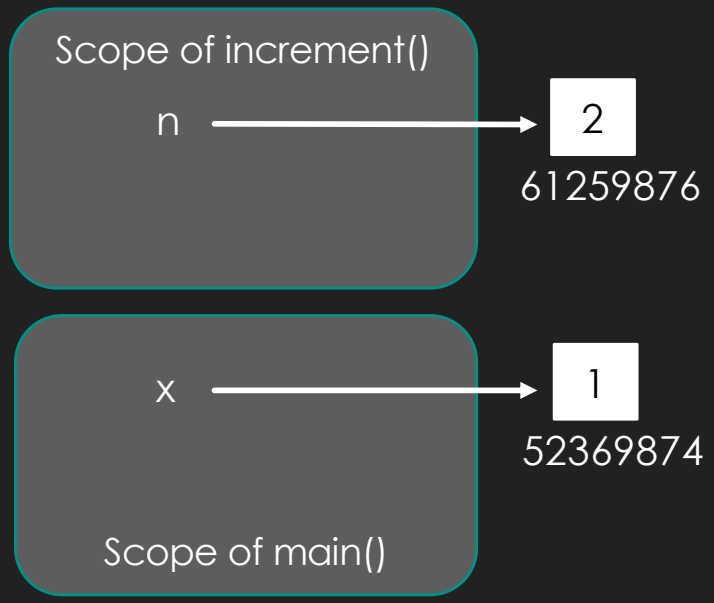


# Passing Arguments by Reference Values

- *When you invoke a function with arguments, each argument's reference is passed by value to the parameter in the function.*

```
def increment(n):
    n += 1
    print("Inside the increment function n = ", n)

def main():
    x = 1
    print("Before the call, x is", x)
    increment(x)
    print("After the call, x is", x)

main() # Call the main function
```

Before the call, x is 1
Inside the increment function n = 2
After the call, x is 1



# Modularizing Code

*Modularizing makes code easy to maintain and debug, and enables the code to be reused.*

- Functions can be used to reduce redundant code and enable code reuse.
- Functions can also be used to modularize code and improve a program's quality.
- In Python, you can place the function definition into a file called *module* with the file-name extension **.py**.
  - The module can be later imported into a program for reuse.
  - The module file should be placed in the same directory with your other programs.
  - A module can contain more than one function.
    - Each function in a module must have a different name.
- **random** and **math** are the modules defined in the Python library, and thus they can be imported into any Python program.

GDCFunction.py

```
# Return the gcd of two integers
def gcd(n1, n2):
    gcd = 1 # Initial gcd is 1
    k = 2 # Possible gcd

    while k <= n1 and k <= n2:
        if n1 % k == 0 and n2 % k == 0:
            gcd = k # Update gcd
        k += 1

    return gcd      # Return gcd
```

GDCTest.py

```
from GDCFunction import gcd  # import the gcd function

# Prompt the user to enter two integers
n1 = eval(input("Enter the first integer: "))
n2 = eval(input("Enter the second integer: "))
print("The greatest common divisor for", n1, "and", n2, "is", gcd(n1, n2))
```

Output

```
Enter the first integer: 25
Enter the second integer: 20
The greatest common divisor for 25 and 20 is 5
```

# Summary

- Functions are very effective in making your code reusable.
- We have discussed how to create a user defined function with
  - Zero or multiple arguments
  - Void and value returning functions
  - Returning multiple values from a function
- Scope of the variable
- Modularization of the code