

Chapter 4: Functions

Instructor: Dr. Murat Tunc

Lecture 4

Last Week (Summary)



boolean Data Type

- A variable that holds a boolean value is known as a **boolean variable**
- The boolean data type is used to declare boolean variables
- A boolean expression evaluates to **True** or **False**

`b = 1 > 2 # b is assigned the value False`



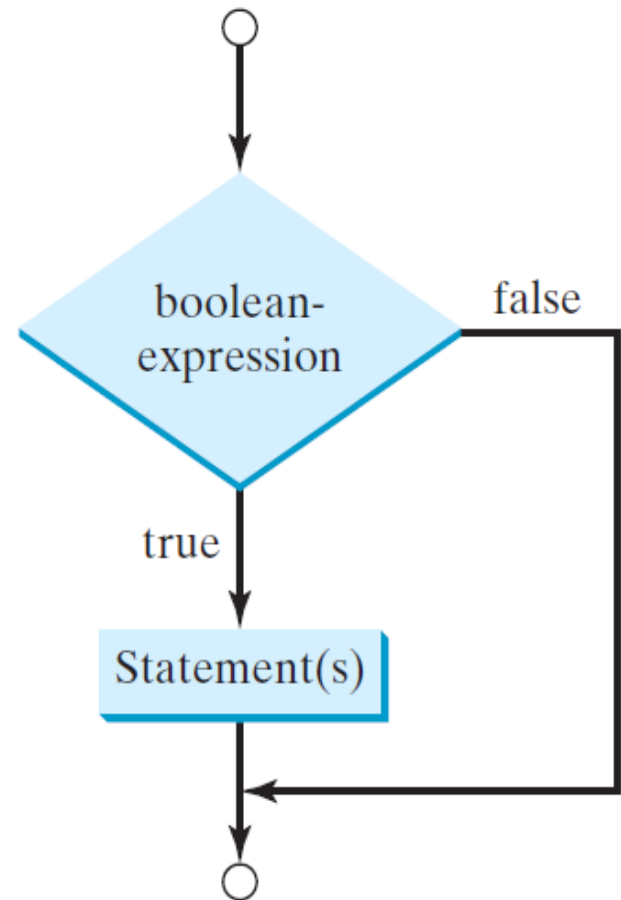
Relational Operators

Operator	Mathematics Symbol	Name	Example (radius is 5)	Result
<	<	less than	<code>radius < 0</code>	<code>false</code>
<=	≤	less than or equal to	<code>radius <= 0</code>	<code>false</code>
>	>	greater than	<code>radius > 0</code>	<code>true</code>
>=	≥	greater than or equal to	<code>radius >= 0</code>	<code>true</code>
==	=	equal to	<code>radius == 0</code>	<code>false</code>
!=	≠	not equal to	<code>radius != 0</code>	<code>true</code>



One-way **if** Statements

if boolean-expression:
statement(s)



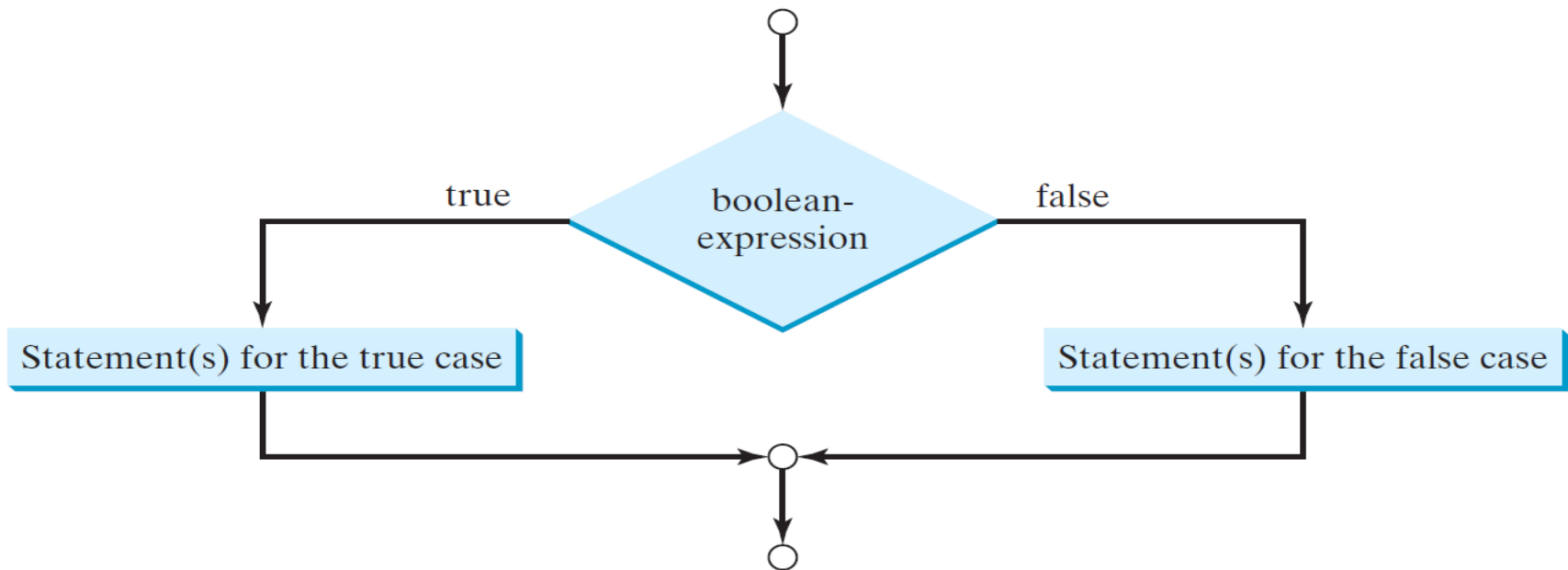
Two-way **if-else** statements

if boolean-expression:

statement(s)-for-the-true-case

else:

statement(s)-for-the-false-case



Logical Operators

Operator	Description
not	logical negation
and	logical conjunction
or	logical disjunction



Truth Table for Operator **and**

p_1	p_2	p_1 and p_2	Example (assume age = 24, weight = 140)
false	false	false	age \leq 18 and weight $<$ 140 is false
false	true	false	age \leq 18 and weight $==$ 140 is false
true	false	false	age $>$ 18 and weight $>$ 140 is false
true	true	true	age $>$ 18 and weight \geq 140 is true



Truth Table for Operator **or**

p_1	p_2	p_1 or p_2	Example (assume age = 24, weight = 140)
false	false	false	age < 18 or weight \geq 150 is false
false	true	true	age < 18 or weight \geq 130 is true
true	false	true	age > 18 or weight \geq 150 is true
true	true	true	age > 18 or weight \geq 130 is true



Practice Exercise 1

Write a program that

- 1) **prompts** the user to enter a movie's IMDB rating (0 to 10 – may include decimal, like 3.5) and Metascore (0 to 100 - integer), and
- 2) **checks** whether the movie is recommended to watch

Hint: Recommend if $\text{rating} > 7.0$ & $\text{Metascore} > 60$



Practice Exercise 1 - Answer

```
# Practice Exercise 1

# Step 1: Read in the IMDB rating and metascore

IMDBRating = float(input("Please input IMDB rating of a movie (0 to 10) and press Enter:"))

metascore = int(input("Please input IMDB rating of a movie (0 to 100 - integer) and press Enter:"))

# Step 2: Recommend the movie if the rating > 7.0 and metascore > 60

if IMDBRating > 7.0 and metascore > 60:
    print("This movie is recommended to watch!")
else:
    print("This movie is not recommended to watch!")
```

Please input IMDB rating of a movie (0 to 10) and press Enter: 7.6

Please input IMDB rating of a movie (0 to 100 - integer) and press Enter: 63

This movie is recommended to watch!



Practice Exercise 2

Write a program that

- 1) **prompts** the user to enter the day, month and year he/she was born, and
- 2) **displays** whether he/she can legally purchase beer in US
 - Give me a beer, please.
 - Can I see an ID? 30.06.1990
 - Here is your beer.



Practice Exercise 2 - Answer

```
# Practice Exercise 2

# Step 1: Read in the day, month and year

print("Give me a beer please!")

print("Can I see an ID?")

dayOfBirth = int(input("Please input the day of your birthday and press Enter:"))

monthOfBirth = int(input("Please input the month of your birthday and press Enter:"))

yearOfBirth = int(input("Please input the year of your birthday and press Enter:"))

# Step 2: Check whether the user can legally purchase beer in US

if yearOfBirth < 2000 or (yearOfBirth == 2000 and monthOfBirth < 11) or \
    (yearOfBirth == 2000 and monthOfBirth == 11 and dayOfBirth <= 30):
    print("Here is your beer!")
else:
    print("I'm sorry, I cannot sell you a beer!")
```

Give me a beer please!

Can I see an ID?

Please input the day of your birthday and press Enter: 30

Please input the month of your birthday and press Enter: 06

Please input the year of your birthday and press Enter: 1990

Here is your beer!



Chapter 4: Functions

Instructor: Dr. Murat Tunc

Lecture 4

Functions – Motivating Problem

- Find the absolute difference between two numbers
 - 3 and 10
 - 22 and 65
 - 83 and 99



A Tedious Way

```
num1 = 3
num2 = 10
if num1 > num2:
    absoluteDifference = num1 - num2
else:
    absoluteDifference = num2 - num1
print("The absolute difference is", absoluteDifference)
```

```
num1 = 22
num2 = 65
if num1 > num2:
    absoluteDifference = num1 - num2
else:
    absoluteDifference = num2 - num1
print("The absolute difference is", absoluteDifference)
```

```
num1 = 83
num2 = 99
if num1 > num2:
    absoluteDifference = num1 - num2
else:
    absoluteDifference = num2 - num1
print("The absolute difference is", absoluteDifference)
```


Observation

- Computing the absolute difference in all the 3 cases is very similar **except** that the **starting and ending integers** are different
- It would be nice if we could write a **common code** once and **reuse** it
- We can do this by defining a **‘function’**



Solution - Functions

```
myDifference(x, y)
```

A **function** that returns the absolute difference between two numbers (x and y)

```
print("The absolute difference between 3 and 10 is",myDifference(3, 10))  
print("The absolute difference between 22 and 65 is",myDifference(22,65))  
print("The absolute difference between 83 and 99 is",myDifference(83,99))
```



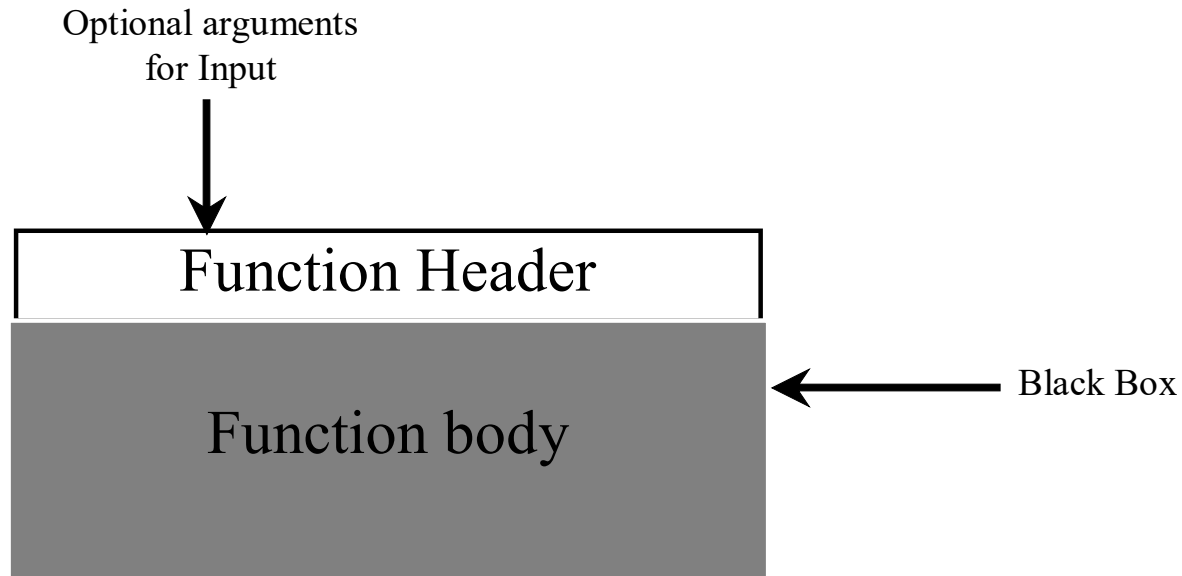
Solution - Functions

```
def myDifference(num1, num2):  
    if num1 > num2:  
        absoluteDifference = num1 - num2  
    else:  
        absoluteDifference = num2 - num1  
    return absoluteDifference  
  
print( myDifference( num1, num2 ) )
```



Function Abstraction

- You can think of the function body as a **black box** that contains the detailed implementation for the function



Introduction to Functions

- Functions can be used to define **reusable** code, **organize**, and **simplify** coding
- Some built-in **function** (defined in the Python library) that we have already used:
 - `print()`
 - `input()`
 - `pow(,)`
- How do we create **our own function**?



Defining a function

Invoking a function



Defining a Function

- A function definition consists of
 - function **name**
 - **parameters**
 - body

- Syntax:

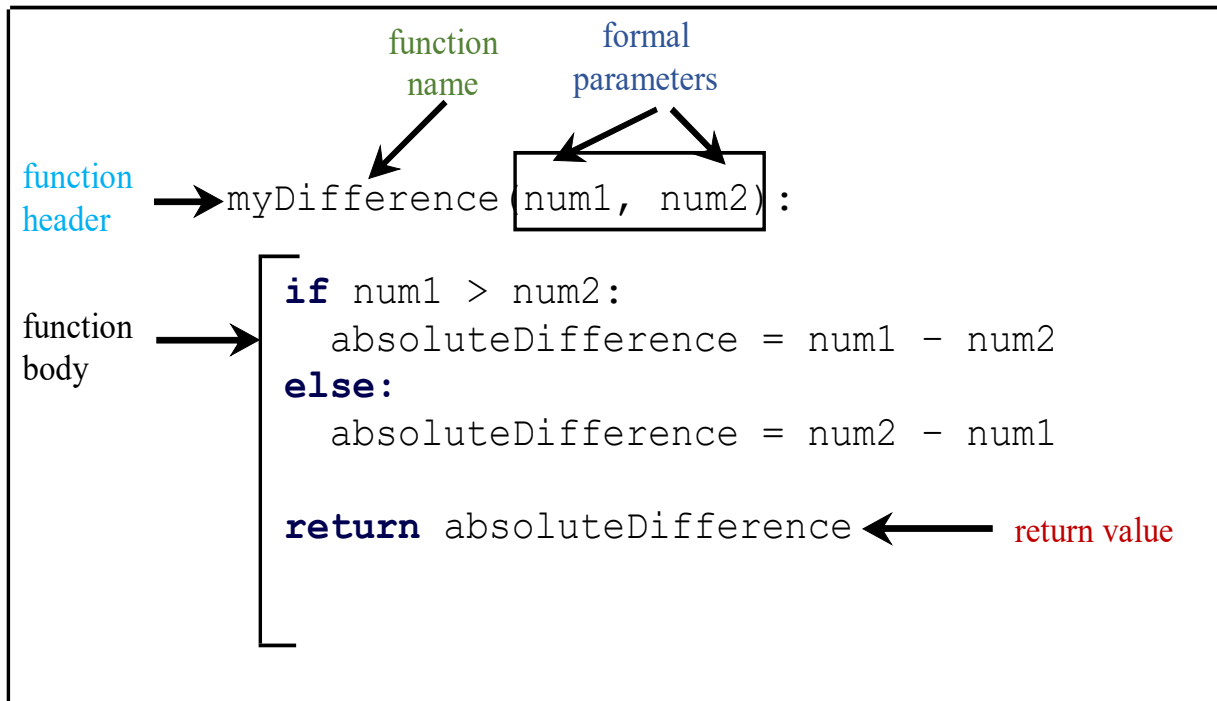
```
functionName (list of parameters):  
    # function body
```



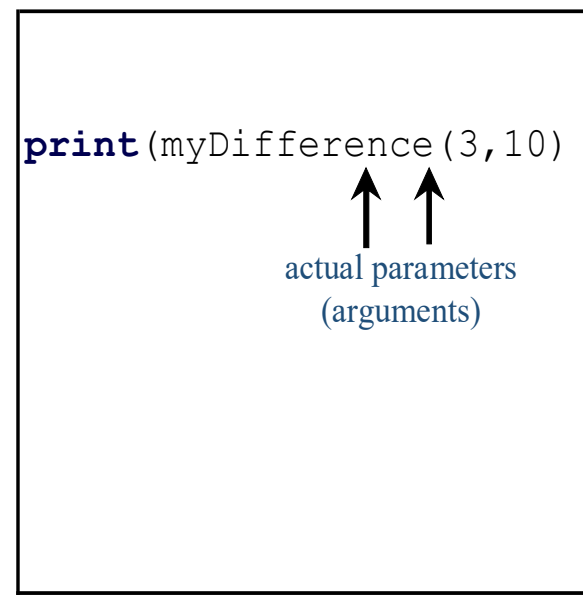
Defining a Function

- A function is a collection of statements that are grouped together to perform an operation

Define a function



Invoke a function



Defining a function - Function Signature

- **Function signature** is the combination of the **function name** and the **parameter list**

Function signature

```
def myDifference (num1, num2) :  
    if num1 > num2:  
        absoluteDifference = num1 - num2  
    else:  
        absoluteDifference = num2 - num1  
    return absoluteDifference
```



Defining a Function - Formal Parameters

- The variables defined in the function header are known as **formal parameters**

formal parameters



```
def myDifference (num1, num2) :  
    if num1 > num2:  
        absoluteDifference = num1 - num2  
    else:  
        absoluteDifference = num2 - num1  
    return absoluteDifference
```



Defining a function

Invoking a function



Invoking a Function - Actual Parameters

- When a function is **invoked**, you pass a value to the parameter. This value is referred to as **actual parameter** or **argument**

```
def myDifference (num1, num2) :  
    if num1 > num2:  
        absoluteDifference = num1 - num2  
    else:  
        absoluteDifference = num2 - num1  
    return absoluteDifference
```

diffVariable = myDifference(3, 10)

Actual parameters
(arguments)



Parameters

- The variables defined in the **function header** are known as **formal parameters** or simply parameters
- A parameter is like a **placeholder**: when a function is invoked, you **pass actual value** of the parameter
- This value is referred to as the **actual parameter** or argument
- Parameters are **optional**; that is a function may contain no parameters



Invoking a Function – Return Value

- A function **may or may not return** a value

```
def myDifference (num1, num2) :  
    if num1 > num2:  
        absoluteDifference = num1 - num2  
    else:  
        absoluteDifference = num1 - num2  
    return absoluteDifference
```

**Return
value** →



Review Exercise 1

What is the value of **integer** ?

```
def myInteger(num1, num2):  
    return num2 - num1  
integer = myInteger (5, 14)
```

Ans: integer = 9

Why: myInteger is invoked with actual parameters of 5 and 14
myInteger returns value of $14 - 5$
integer = 9



Review Exercise 2

What is the value of **integer** ?

```
def myInteger(num2, num1):  
    return num2 % num1  
integer = myInteger (5, 14)
```

Ans: integer = 5

Why: myInteger is invoked with actual parameters of 5 and 14

myInteger returns a value of 5 % 14

% means **remainder** - 5 divided by 14 - remainder is 5

integer = 5



Review Exercise 3

What is the value of **integer** ?

```
def myInteger(num1, num2):  
    return num2 / num1  
integer = myInteger (5, 14)
```

Ans: integer = 2.8

Why: myInteger is invoked with actual parameters of 5 and 14
myInteger returns a value of 14 / 5
14 divided by 5 is 2.8
integer = 2.8



Review Exercise 4

What is the value of **integer** ?

```
def myInteger(num1, num2):  
    return num2 // num1  
integer = myInteger (5, 14)
```

Ans: integer = 2

Why: myInteger is invoked with actual parameters of 5 and 14
// means integer division
14 divided by 5 is 2.8, integer part is 2
integer = 2



Value-returning function

Void function



Return Value Type

- A function that returns a value is called as a **value-returning function**
- A function terminates when a return statement is executed
- A function that does not return a value is called as a **void function**



Value-returning functions

```
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"  
  
print("The grade is", getGrade (78.5))
```



Void functions

```
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")
```

```
print("The grade is")
```

```
printGrade (78.5)
```



Review Exercise 5

What is the value of **integer** ?

```
def myInteger (num1, num2):  
    print(num2 / num1)  
integer = myInteger (5, 14)
```

Ans: None

Why: myInteger function does not return a value
integer is defined, but it's value is not set
integer = None



Review Exercise 6

What does the following print on the console?

```
def myPrint (string, number):  
    print(string + number)  
myPrint (5, 14)
```

Ans: 19

Why: string is an integer variable and stores a value of 5
number = 14
print(5 + 14) , which prints 19



Review Exercise 7

What does the following print on the console?

```
def myPrint (string, number):  
    print(string, number)  
myPrint (5, 14)
```

Ans: 5 14

Why: string is an integer variable and stores a value of 5
number = 14
`print(5, 14)` , prints 5 14



Review Exercise 8

What does the following print on the console?

```
def myPrint (string, number):  
    print(string + number)  
myPrint ("5", 14)
```

Ans: Error

Why: string is "5", number is 14

"5" + 14 is error

Since "5" is text (i.e. string), whereas 14 is a number



In-class Exercise

Write a program that

- 1) asks the user to **enter two integers**: num1 and num2, and
- 2) **invokes a function** that takes two integers
- 3) The function returns **true** if the first number is greater than the second, else returns **false**
- 4) The program prints whether num1 is greater than num2



Answer

```
def myComparison (num1, num2):
```

```
    result = False
```

```
    if num1 > num2:
```

```
        result = True
```

```
    return result
```

```
num1 = int ( input( "Please input an integer and press Enter" ) )
```

```
num2 = int ( input( "Please input an integer and press Enter" ) )
```

```
if myComparison(num1, num2):
```

```
    print("The first number is greater!")
```

```
else:
```

```
    print("The second number is greater or equal!")
```



Invoking / Calling a function



Trace Function Invocation

i is 5

```
i = 5
j = 2
k = max(i, j)

print("The maximum between",
      i, "and", j, "is", k)
```

```
def max(num1, num2):
    if num1 > num2:
        result = num1
    else:
        result = num2
    return result
```



Trace Function Invocation

j is 2

```
i = 5
j = 2
k = max(i, j)

print("The maximum between",
      i, "and", j, "is", k)
```

```
def max(num1, num2):
    if num1 > num2:
        result = num1
    else:
        result = num2
    return result
```



Trace Function Invocation

invoke max(i, j)

```
i = 5
j = 2
k = max(i, j)

print("The maximum between",
i, "and", j, "is", k)
```

```
def max(num1, num2):
    if num1 > num2:
        result = num1
    else:
        result = num2
    return result
```



Trace Function Invocation

invoke max(i, j)
Pass the value of i to num1
Pass the value of j to num2

```
i = 5
j = 2
k = max(i, j)

print("The maximum between",
      i, "and", j, "is", k)
```

```
def max(num1, num2):
    if num1 > num2:
        result = num1
    else:
        result = num2
    return result
```



Trace Function Invocation

```
i = 5
j = 2
k = max(i, j)

print("The maximum between",
      i, "and", j, "is", k)
```

(num1 > num2) is **true**
since num1 is 5 and num2 is 2

```
def max(num1, num2):
    if num1 > num2:
        result = num1
    else:
        result = num2
    return result
```



Trace Function Invocation

```
i = 5
j = 2
k = max(i, j)

print("The maximum between",
      i, "and", j, "is", k)
```

```
def max(num1, num2):
    if num1 > num2:
        result = num1
    else:
        result = num2
    return result
```

result is 5



Trace Function Invocation

```
i = 5
j = 2
k = max(i, j)

print("The maximum between",
      i, "and", j, "is", k)
```

```
def max(num1, num2)
    if num1 > num2:
        result = num1
    else:
        result = num2
    return result
```

return result, which is 5



Trace Function Invocation

return max(i, j) and
assign the return value to k

```
i = 5
j = 2
k = max(i, j)

print("The maximum between",
      i, "and", j, "is", k)
```

```
def max(num1, num2):
    if num1 > num2:
        result = num1
    else:
        result = num2
    return result
```



Trace Function Invocation

Execute the print statement

```
i = 5  
j = 2  
k = max(i, j)
```

```
print("The maximum between",  
i, "and", j, "is", k)
```

```
def max(num1, num2):  
    if num1 > num2:  
        result = num1  
    else:  
        result = num2  
    return result
```



Review Exercise 9 – Is this OK ?

Is the following code OK – runs without error?

```
def mySum (num1) :  
    num1 += 2  
    num2 += 5  
    return num1 + num2  
value = mySum (10, 2)
```

Ans: Error --- num2 is not defined

Why: num2 is not defined in function header



Review Exercise 10 – Is this OK ?

Is the following code OK – runs without error?

**Unreachable
code**

```
def mySum (num1, num2) :  
    num1 += 2  
    num2 += 5  
    return num1 + num2  
    num1 += 10  
value = mySum (10, 2)
```

Ans: It is OK – runs without error

Why: After **return**, the function terminates

Do not write any code after **return**



Review Exercise 11 – Is this OK ?

Is the following code OK – runs without error?

```
def mySum (num1, num2) :  
    num1 += 2  
    num2 += 5  
    return num1 + num2  
    num1 += 10  
    return num1 + num2  
value = mySum (10, 2)
```

Ans: Yes, it is OK – runs without error

Why: A second **return** is not executed, but Python does not show any error



Review Exercise 12 – Is this OK?

Is the following code OK – runs without error?

```
value = mySum (10, 2)
def mySum (num1, num2) :
    num1 += 2
    num2 += 5
    return num1 + num2
    num1 += 10
    return num1 + num2
```

Ans: No, there is an error

Why: Always define the function before invoking it



Call Stack

- Each time a function is invoked
 - The system creates an **activation record**
- **Activation record**
 - Stores parameters and variables for the function
 - Places the activation record in an area of memory
 - This memory area is known as a **call stack**
- **A local variable**
 - Variable defined inside a function



Trace Call Stack

```
def myInteger (num1, num2) :  
    num1 += 1  
    print ( num1 )  
    return num2 % num1  
num1 = 5  
num2 = 14  
print ( myInteger (num1, num2) )  
print ( num1 )
```

num1 is declared and initialized

The main program

num1: 5



Trace Call Stack

```
def myInteger (num1, num2) :  
    num1 += 1  
    print ( num1 )  
    return num2 % num1  
  
num1 = 5  
num2 = 14  
print ( myInteger (num1, num2) )  
print ( num1 )
```

num2 is declared and initialized

The main program

num2: 14
num1: 5



Trace Call Stack

```
def myInteger (num1, num2) :  
    num1 += 1  
    print ( num1 )  
    return num2 % num1  
  
num1 = 5  
num2 = 14  
print ( myInteger (num1, num2) )  
print ( num1 )
```

Invoke myInteger (5, 14)

The main program

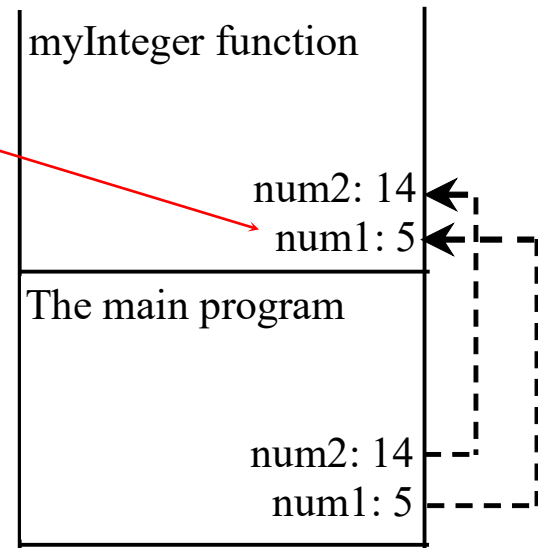
num2: 14
num1: 5



Trace Call Stack

pass the values of 5 and 14 to num1 and num2

```
def myInteger (num1, num2) :  
    num1 += 1  
    print ( num1 )  
    return num2 % num1  
  
num1 = 5  
num2 = 14  
print ( myInteger (num1, num2) )  
print ( num1 )
```



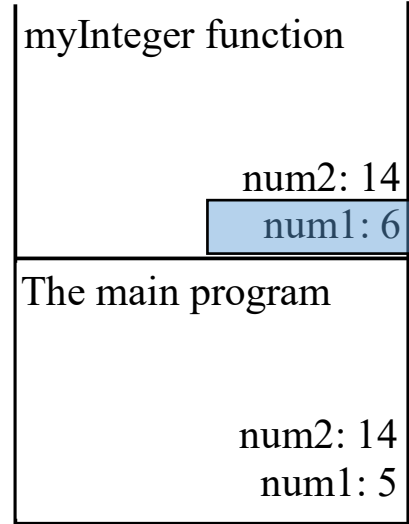
myInteger method
is invoked.



Trace Call Stack

```
def myInteger (num1, num2) :  
    num1 += 1  
    print ( num1 )  
    return num2 % num1  
  
num1 = 5  
num2 = 14  
print ( myInteger (num1, num2) )  
print ( num1 )
```

num1 in myInteger function is increased by 1



Trace Call Stack

```
def myInteger (num1, num2) :  
    num1 += 1  
    print ( num1 )  
    return num2 % num1  
  
num1 = 5  
num2 = 14  
print ( myInteger (num1, num2) )  
print ( num1 )
```

Print num1 in myInteger function,
which is 6

myInteger function

num2: 14
num1: 6

The main program

num2: 14
num1: 5



Trace Call Stack

```
def myInteger (num1, num2) :  
    num1 += 1  
    print ( num1 )  
    return num2 % num1  
  
num1 = 5  
num2 = 14  
print ( myInteger (num1, num2) )  
print ( num1 )
```

Return 14 % 6, the remainder is 2

myInteger function

num2: 14
num1: 6

The main program

num2: 14
num1: 5



Trace Call Stack

```
def myInteger (num1, num2) :  
    num1 += 1  
    print ( num1 )  
    return num2 % num1  
  
num1 = 5  
num2 = 14  
print ( myInteger (num1, num2) )  
print ( num1 )
```

2 is printed

The main program

num2: 14
num1: 5



Trace Call Stack

```
def myInteger (num1, num2) :  
    num1 += 1  
    print ( num1 )  
    return num2 % num1  
  
num1 = 5  
num2 = 14  
print ( myInteger (num1, num2) )  
print ( num1 )
```

5 is printed

The main program

num2: 14
num1: 5



Review Exercise 13

What does the following code print ?

```
def myInteger (num1, num2) :  
    num1 //= 5  
    print ( num1 )  
    return num2 % num1  
num1 = 15  
num2 = myInteger(33, num1)  
print ( myInteger (num1*num2, num2) )  
print ( num2 )
```

Ans: First prints **6**, then **9**, then **3**, and finally **3**



```
def myInteger (num1, num2) :  
    num1 //= 5  
    print ( num1 )  
    return num2 % num1  
  
num1 = 15  
num2 = myInteger(33, num1)  
print ( myInteger (num1*num2, num2) )  
print ( num2 )
```

Ans: First prints **6**, then **9**, then **3**, and finally **3**

Why: **myInteger** is invoked with actual parameters of 33 and 15
num1 in **myInteger** becomes **6** when $\text{num1} \text{//} = 5$, **print(6)**

myInteger returns a value of $15 \% 6 = 3$

num2 in the main program becomes **3**

myInteger is invoked with actual parameters of $3*15$ and 3

num1 in **myInteger** becomes **9** when $45 \text{//} = 5$, **print(9)**

myInteger returns a value of $3 \% 9 = 3$, **print(3)**

num2 in the main program is still **3**, **print(3)**



Benefits of Functions

- Write a function **once** and **reuse** it anywhere
- Information hiding. Hide the implementation from the user
- Reduce complexity

