ITD62-123 COMPUTER PROGRAMMING

IMI62-122 FUNDAMENTAL OF COMPUTER PROGRAMMING

Theerat Saichoo & Yunyong Punsawad
School of Informatics, Walailak University

Chapter 4
Functions

python™

Python Functions

- A function is a block of code which only runs when it is called.
- We can pass data, known as parameters, into a function.
- A function can return data as a result.

Creating a Function

- In Python a function is defined using the **def** keyword:
- Syntax

```
def function_name():
        command statements
```

- Example

```
def my_function():
        print("Hello from a function")
```

Calling a Function

- To call a function, use the function name followed by parenthesis:
- Syntax

my_function()

- Example

```
def print_hello():
        print("Hello from a function")

print_hello()
```

Arguments

- Information can be passed into functions as **arguments**.
- Arguments are specified *after* the function name*, inside the parentheses*. We can add as many arguments as you want, just separate them with a comma(,).
- The following example has a function with one argument (f_name). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Arguments

- Example

```
def my_function(fname):
        print(fname + " Refsnes")


my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Parameters or Arguments?

- The terms *parameter* and *argument* can be used for the same thing: information that are passed into a function.
- From a function's perspective:
  - A parameter is the variable listed inside the parentheses in the function definition.
  - An argument is the value that is sent to the function when it is called.

Number of Arguments

- By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.
- Example

```
def my_function(fname, lname):
    print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

Arbitrary Arguments, *args

- If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
- This way the function will receive a tuple of arguments, and can access the items accordingly:

Arbitrary Arguments, *args

- Example

```
def my_function(*kids):
    print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

Keyword Arguments

- We can also send arguments with the *key* = *value* syntax.
- This way the order of the arguments does not matter.
- Example

```
def my_function(child3, child2, child1):
        print("The youngest child is " + child3)


my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Arbitrary Keyword Arguments, **kwargs

- If you do not know how many keyword arguments that will be passed into your function, add two asterisk: **** before the parameter name in the function definition.
- This way the function will receive a **dictionary of arguments**, and can access the items accordingly:

Arbitrary Keyword Arguments, **kwargs

- Example

```
def my_function(**kid):
        print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

Default Parameter Value

- The following example shows how to use a default parameter value.
- If we call the function without argument, it uses the default value:
- Example

```
def my_function(country = "Norway"):
        print("I am from " + country)
my_function("Sweden")
my_function()
my_function("Brazil")
```

Passing a List as an Argument

- We can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.
- E.g. if you send a List as an argument, it will still be a List when it reaches the function:

Passing a List as an Argument

- Example

```
def my_function(food):
    for x in food:
        print(x)


fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

Return Values

- To let a function return a value, use the return statement:
- Example

```
def my_function(x):
        return 5 * x

print(my_function(3))


z = my_function(5)
print(z)
```

Types of functions

- For better understanding of arguments and return value from the function, user-defined functions can be categorized as:
  - Function with no arguments and no return value
  - Function with no arguments and a return value
  - Function with arguments and no return value
  - Function with arguments and a return value

Function with no arguments and no return value

- Example

```
def calculate_BMI():
    weight = 70
    height = 1.70
    bmi = weight/(height*height)
    print('Your BMI is {:.2f}'.format(bmi))

calculate_BMI()
```

```
Your BMI is 24.22
>
```

Function with no arguments and a return value

- Example

```
def calculate_BMI():
    weight = 70
    height = 1.70
    bmi = weight/(height*height)
    return bmi

result_bmi = calculate_BMI()
print('Your BMI is {:.2f}'.format(result_bmi))
```

```
Your BMI is 24.22
>
```

Function with arguments and no return value

- Example

```
def calculate_BMI(weight,height):
    bmi = weight/(height*height)
    print('Your BMI is {:.2f}'.format(bmi))

calculate_BMI(70,1.7)
```

```
Your BMI is 24.22
>
```

Function with arguments and a return value

- Example

```
def calculate_BMI(weight,height):
    bmi = weight/(height*height)
    return bmi

result_bmi = calculate_BMI(70,1.7)
print('Your BMI is {:.2f}'.format(result_bmi))
```

```
Your BMI is 24.22
>
```

Scope Rules

- A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed.
  - Inside a function or a block which is called **local** variables,
  - Outside of all functions which is called **global** variables.
  - In the definition of function parameters which are called **formal** parameters.

Local Variables

- Variables that are declared inside a function or block are called local variables.
- They can be used only by statements that are inside that function or block of code.

Global Variables

- Global variables are defined outside a function, usually on top of the program.
- Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.
- A global variable can be accessed by any function.
- A program can have same name for local and global variables but the value of **local variable** inside a function will take preference.

Special thanks to W3Schools

Class activity 6

เขียนคำสั่งภาษา Python เพื่อให้มีการทำงานของโปรแกรมดังนี้ ...

1. ฟังก์ชันสำหรับคำนวณหาค่าเฉลี่ยของชุดตัวเลขจำนวนหนึ่ง โดยรับค่า parameter เป็นชนิด list ซึ่งใน list ประกอบไปด้วยตัวเลขจำนวนเต็มหลายค่า จากนั้นคำนวณค่าเฉลี่ยแล้ว return ค่าออกเป็นตัวเลขทศนิยม

2. ฟังก์ชันสำหรับประเมินค่าอุณหภูมิร่างกายว่ามีไข้หรือไม่ โดยให้รับค่าอุณหภูมิจากผู้ใช้ (User input) ภายในฟังก์ชัน หากมีอุณหภูมิมากกว่า 37.5 องศา ให้ return ค่า true หากต่ำกว่าให้ return false

3. ฟังก์ชันสำหรับพิมพ์ข้อมูลผู้ป่วย โดยมี parameter 3 ตัว คือ Name, Age, Symptoms จากนั้นพิมพ์ข้อมูลจาก parameter ออกทางจอภาพ