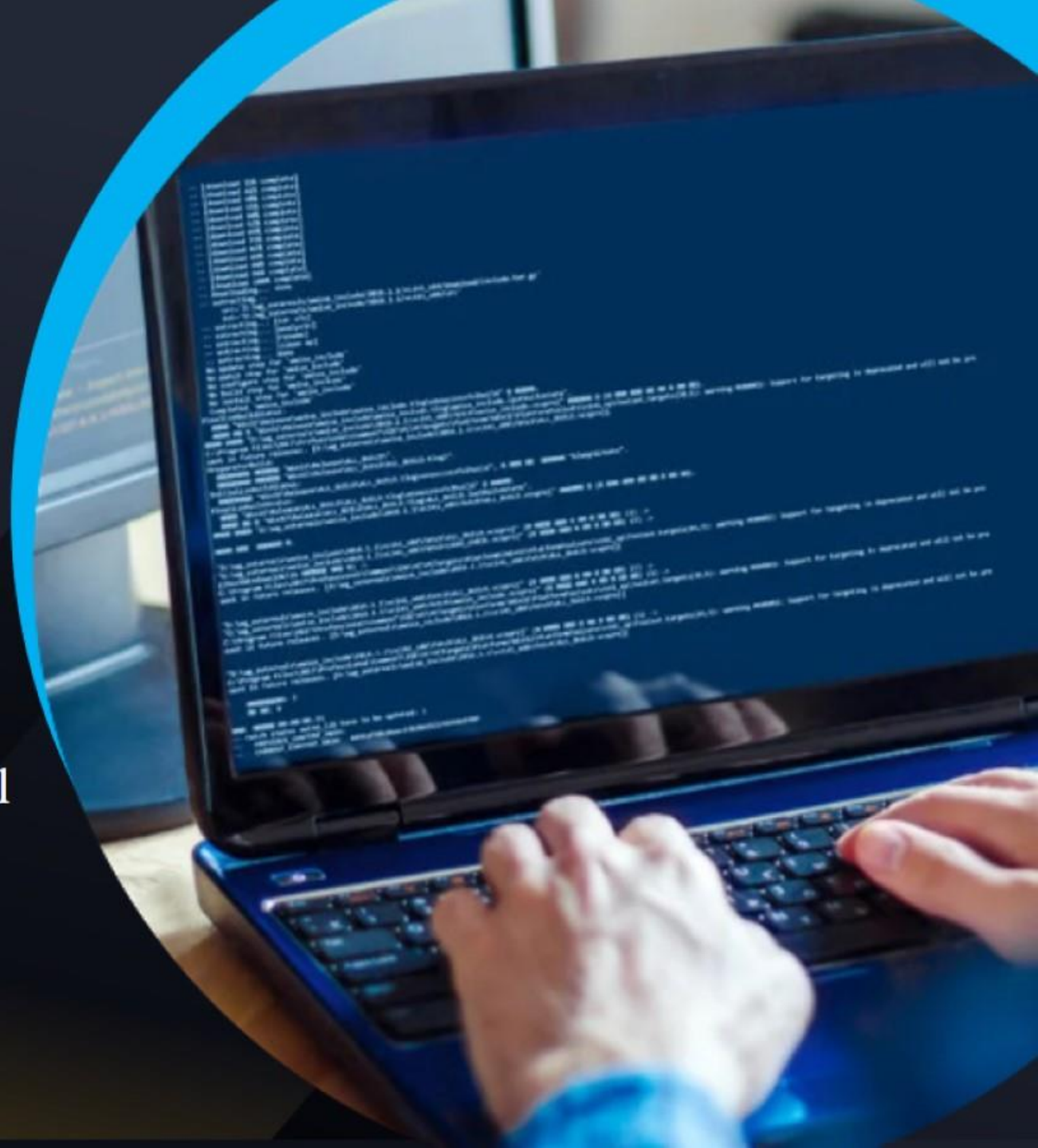


Programming Foundations in Python

Prepared by Dr. Hamada I. AbdulWakel

Lecture 6



The background features a series of vertical panels with rounded ends. The first panel shows hands typing on a laptop. The second panel has a graduation cap icon. The third panel has a play button icon. The fourth panel has a target icon. The fifth panel has an open book icon. The sixth panel has a magnifying glass icon. The seventh panel has a group of people icon. The eighth panel has a bar chart icon. The title 'Functions in Python' is written in a large, blue, sans-serif font across the middle of the panels.

Functions in Python

Lecture 6

Prepared by
Dr. Hamada I. AbdulWakel

www.midocean.edu.km

Lecture

06

RECAP: LECTURE 5

- List data type and its functions.
- Dictionary data type and its functions.

Lecture

06

LECTURE MAIN POINTS

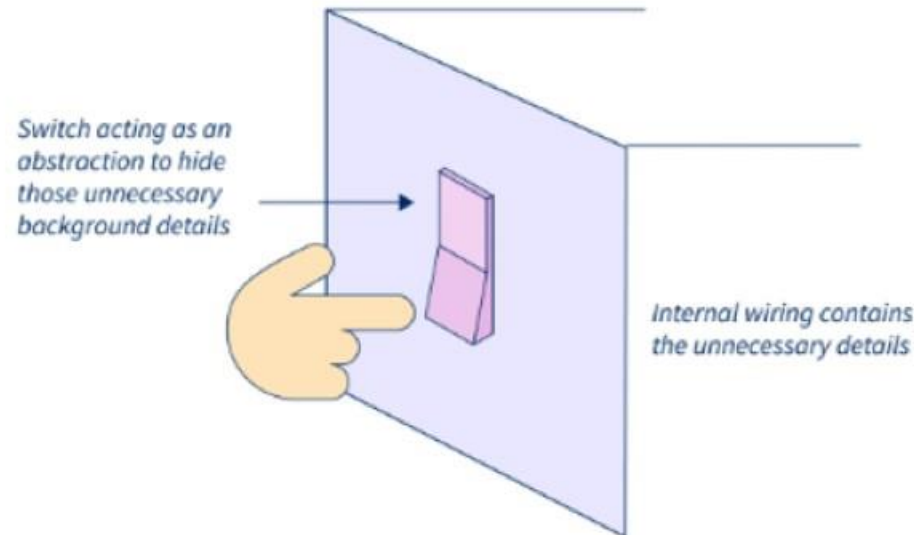
- Abstraction mechanism.
- The creation of the user-defined functions.
- The difference between local and global variables.
- The top-down program design.
- The recursive functions.

Lecture

06

ABSTRACTION MECHANISM

- Abstraction mechanism is used to hide details.
- A programmer hides all but the relevant data in order to reduce complexity and increase efficiency.



Lecture

06

FUNCTIONS AS AN ABSTRACTION MECHANISM

- Functions serve as abstraction mechanisms.
- Functions are the re-usable pieces/block of code which help us to organize structure of the code and do a task.
- Functions allow us to run a specific statements multiple times without repeating these statements.
 - This achieve the concept of "DRY" *don't repeat yourself*.

BUILT-IN VS. USER-DEFINED FUNCTIONS

- The built-in functions: the functions provided by python language.

`print()`
`input()`
`popitem()`

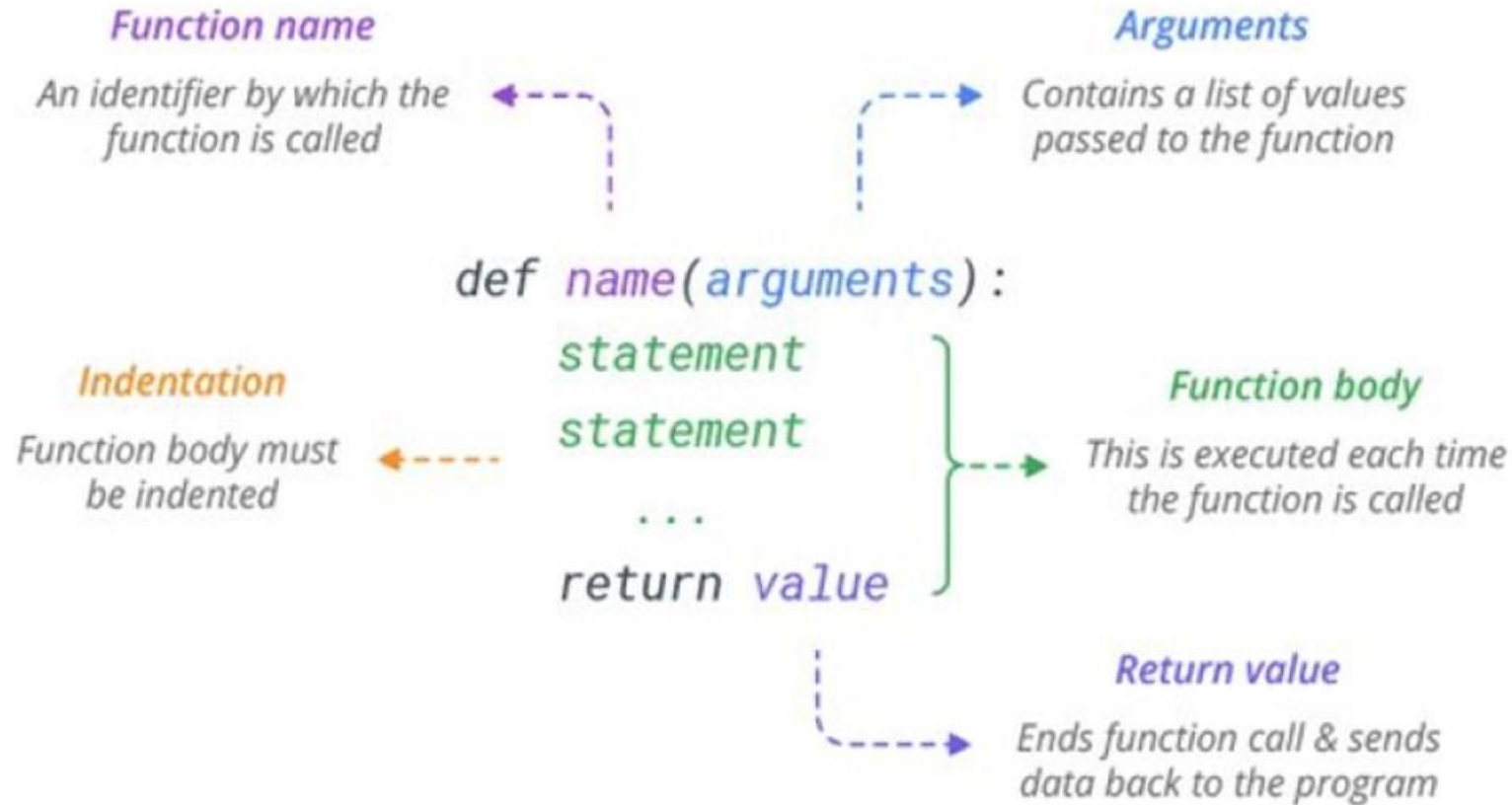
- The user-defined functions: the functions that the user creates it to do a specific task.

`getUserAccount()`
`updateStudentAddress()`

Lecture

06

FUNCTION CREATION



FUNCTION CREATION

Cont'd

- Code:

```
def sayHello( ):
    print("Hello to Python from ESU")
```

```
def Mysum( ):
    X = 4
    Y = 5
    print("Sum = ", X + Y)
```

Lecture

06

FUNCTION CALLING

- The **def** statement only creates/declares a function but does not call it.
- After creating the function, you can call (run) the function by adding parentheses after the function's name.

```
sayHello( )
```

```
Mysum( )
```

```
mysum( ) #NameError
```

Lecture

06

FUNCTION PARAMETERS
AND ARGUMENTS

- A parameter is a **variable** in a function definition. It is a placeholder and hence does not have a concrete value.

```
def Mysum(x, y):  
    print("Sum = ", x + y)
```

- An argument is a **value** passed during function calling.

```
Mysum(2, 3)
```

Lecture

06

RETURNING VALUES FROM FUNCTIONS

- Any function may return a computed value or the default value **None**.
- A function can return a value using **the return statement**.
- Unlike C, C++ or Java, multiple values can be returned by a python function.

Lecture

06

FUNCTION EXAMPLES

- Example1: Create function called greetings() that prints a greeting message to the user.

```
def greetings():  
    print("Hello to my program :) ")
```

```
print(greetings())
```

FUNCTION EXAMPLES

Cont'd

- Example2: Create function called sum() with two parameters start and end to calculate the sum of all the numbers starting from start to end.

```
def sum(start, end):  
    result = 0  
    for x in range(start, end+1):  
        result += x  
    print(result)
```

```
print(sum(1, 10))
```

FUNCTION EXAMPLES

Cont'd

- Example2': Create function called sum() with two parameters start and end to calculate the sum of all the numbers starting from start to end.

```
def sum(start, end):  
    result = 0  
    if start > end:  
        print("End must be greater than start!!!!")  
        return # will return None  
    for x in range(start, end+1):  
        result += x  
    return result  
print(sum(1, 10))
```

FUNCTION EXAMPLES

Cont'd

- Example3:

```
def add(a, b):
```

```
    return a + b
```

```
Res = add(5, 7) # Res will equal to 12
```

- Example4:

```
def hello():
```

```
    print("Hello")
```

```
Res = hello() #will print Hello and return None
```

FUNCTION EXAMPLES

Cont'd

- Example5: returning multiple values

```
def bigger(a, b):  
    if a > b:  
        return a , b  
    else:  
        return b , a
```

```
Res = bigger(5, 7)  
print(Res , type(Res))
```

FUNCTION EXAMPLES

Cont'd

- Example6: write a function to get a class students marks and display the level of student (i.e., Above, Equal, or Below) the average.

```
students = [20, 25, 30, 65, 48, 35]
def stdAverage():
    sum = 0
    for std in students:
        sum += std
    avg = sum / len(students)
    .....
```

FUNCTION EXAMPLES**Cont'd**

- Example6: write a function to get a class students marks and display the level of student (i.e., Above, Equal, or Below) the average.

```
.....  
for std in students:  
    if std > avg:  
        print("Above the average")  
    elif std < avg:  
        print("Below the average")  
    else:  
        print("Equal the average")  
stdAverage()
```

FUNCTION EXAMPLES

Cont'd

- Example7: write a function to calculate and return the factorial $n!$ of a given number.

```
def fact(num):  
    prod = 1  
    for x in range(num, 1, -1):  
        prod *= x  
    return prod
```

Lecture

06

IDENTIFIER SCOPES

- Identifier scope refers to the **accessibility** of identifiers in different parts of code; it is an important concept related to functions.
- The **local scope** refers to the variable defined and used inside a block of code, (i.e., function).

```
def testLocal():  
    a = "Hello im a local variable :)"  
    print(a)
```

```
print(a)
```

- The **global scope** refers to the variable not belongs to a specific block and used inside the whole program.

```
a = "Hello im a global variable :)"  
def testGlobal():  
    print(a)  
  
testGlobal()
```

IDENTIFIER SCOPES

Cont'd

- If you modify a global scope variable inside a function, this will throw an error called **UnboundLocalError**.

```
a = "Hello im a global variable :)"  
def testGlobal():  
    print(a)  # will throw an error  
    a = "Hello"  #this means a new variable  
    print(a)
```

```
testGlobal()
```

Lecture

06

IDENTIFIER SCOPES

Cont'd

- If you need to modify a global scope variable, you need to use the **global** reserved word.

```
a = "Hello im a global variable :)"
```

```
def testGlobal():
```

```
    global a
```

```
    print(a)
```

```
    a = "Hello"
```

```
    print(a)
```

```
testGlobal()
```

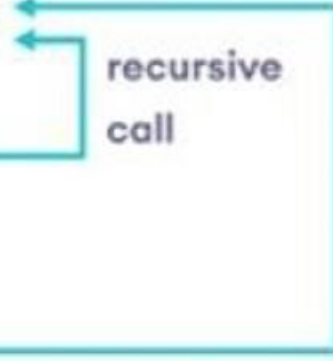
Lecture

06

RECURSIVE FUNCTIONS

- Recursion means iteration. Recursion can be used in place of looping by allowing a function to call itself.
- A function is called recursive, if the body of function calls the function itself until the condition for recursion is true. A python recursive function has a termination condition.

```
def recurse():  
    ...  
    recurse()  
    ...  
  
recurse()
```



RECURSIVE FUNCTIONS

Cont'd

- A recursive function has:
 - Base case: the smallest version of the problem.
 - Recursive case.
- The factorial calculation:
 - Base case: 1 if $number = 0$
 - Recursive case: $number * (number - 1)!$ if $number > 0$

RECURSIVE FUNCTIONS

Cont'd

- Code:

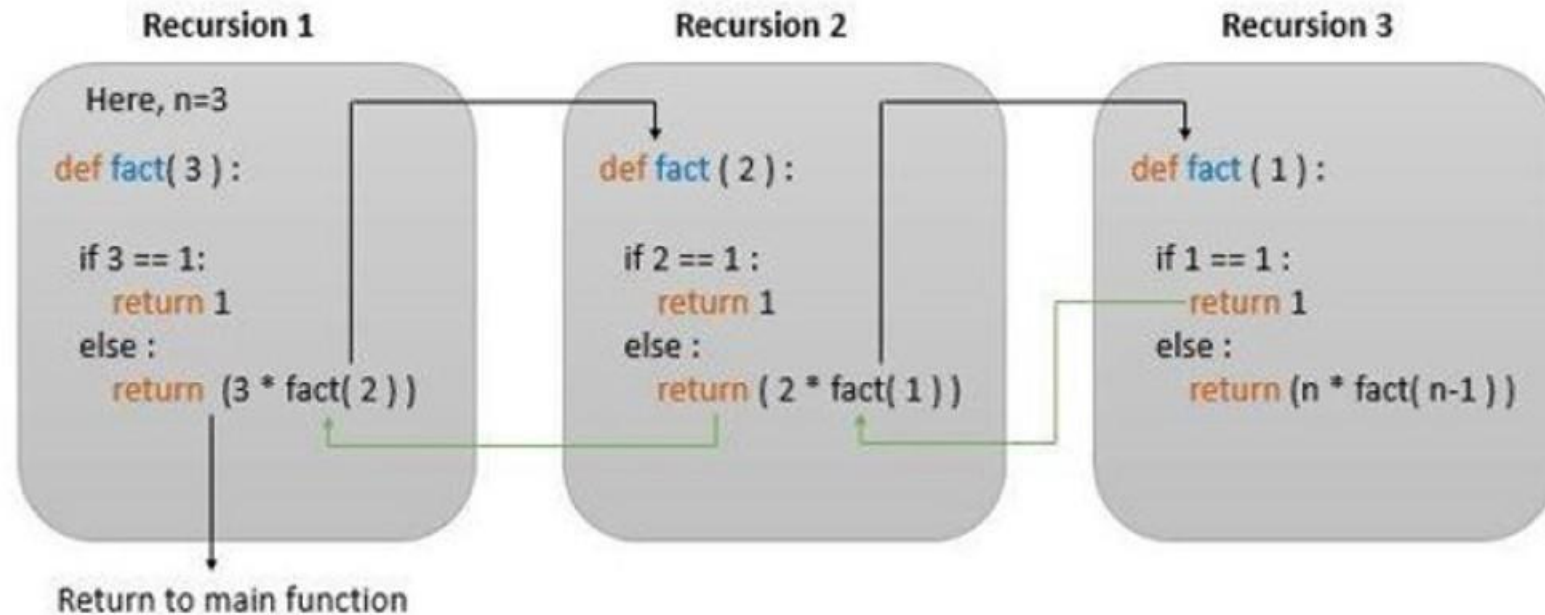
```
def fact(num):  
    if num == 1:  
        return 1  
    else:  
        return num * fact(num-1)
```

```
fact(5)
```

RECURSIVE FUNCTIONS

Cont'd

- The factorial iterations look like.



RECURSIVE FUNCTIONS

Cont'd

- Example10: a function to display the numbers from a lower bound to an upper one

```
def displayRange(lower, upper):  
    if lower <= upper:  
        print(lower)  
        displayRange(lower+1, upper)
```

RECURSIVE FUNCTIONS

Cont'd

- Example11: write a recursive function to return True if word is a palindrome, False if not.

```
def is_palindrome(word):  
    if len(word) <= 1:  
        return True  
    else:  
        return word[0] == word[-1] and is_palindrome(word[1:-1])
```

Lecture

06

INFINITE RECURSION

- Infinite recursion arises when programmer fails to specify base case or to reduce size of problem in a way that terminates the recursive process.
 - Python virtual machine (PVM) eventually runs out of memory resources to manage the process.

```
def runForever():
```

```
    number = 5
```

```
    runForever()
```

```
runForever() #RecursionError: maximum recursion depth exceeded
```

Lecture

06

PROBLEM-SOLVING USING TOP-DOWN DESIGN

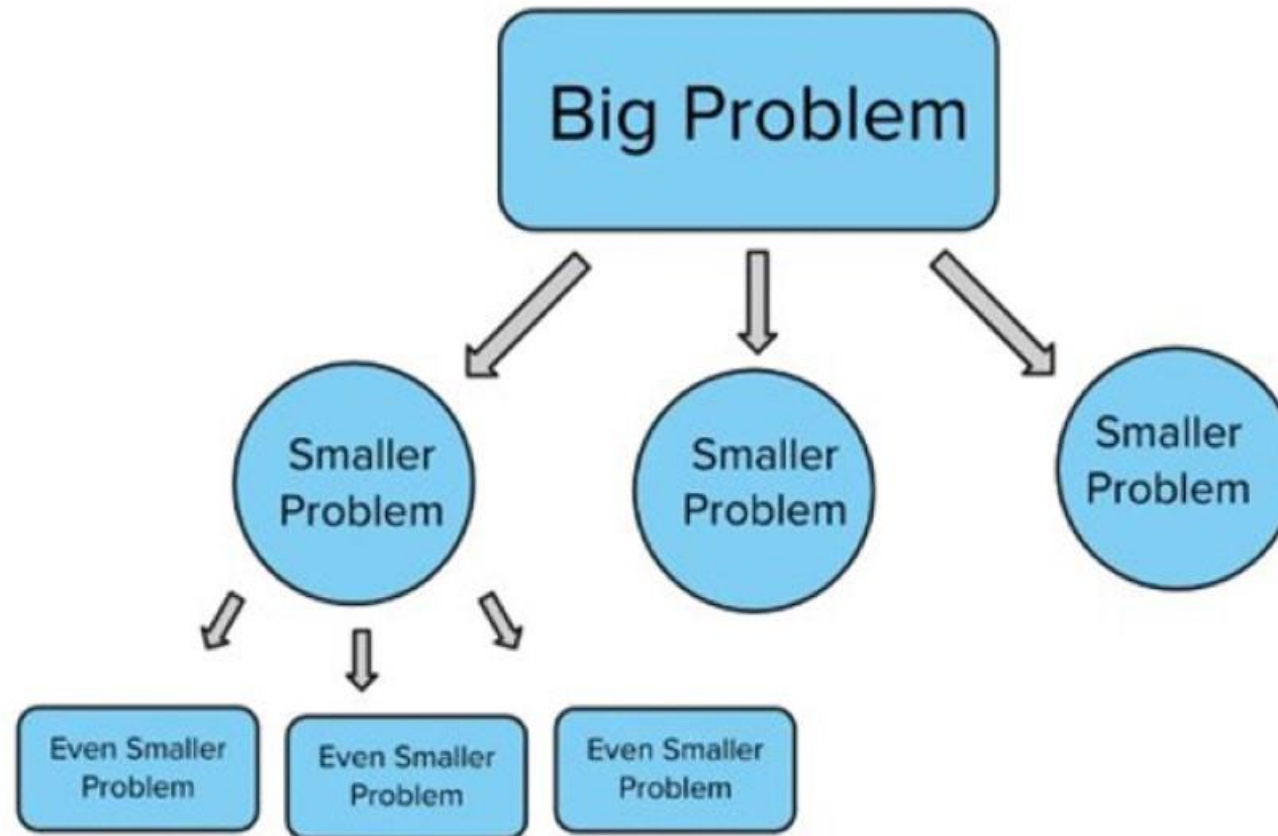
- We may quickly write a large amount of code, but later discover that large portions of the code are unnecessary. These issues emphasize the importance of program design.
- In a well-organized system, each part does its own job in collaborating to achieve a common goal.
- Each function should perform a single coherent task.
 - Example: Computing a summation.

Lecture

06

PROBLEM-SOLVING USING TOP-DOWN DESIGN

Cont'd



PROBLEM-SOLVING USING TOP-DOWN DESIGN

Cont'd

- Top-down design (a.k.a stepwise refinement and problem decomposition) starts with a global view of the entire problem and breaks the problem into smaller, more manageable subproblems.
- The program that implements this design reflects the quality code's characteristics: readable, extendable, and debuggable.
- As functions are developed to solve subproblems, solution to overall problem is gradually filled out.

PROBLEM-SOLVING USING TOP-DOWN DESIGN

Cont'd

- Example12: Write a program to demonstrate the top-down code structure design.

```
def main():
```

```
    stdcount = int(input("Enter total number of students in your class: "))
```

```
    studentsMarks = getMarks(stdcount)
```

```
    markAverage = calAverage(studentsMarks)
```

```
    displayStdLevel(studentsMarks, markAverage)
```

Lecture

06

PROBLEM-SOLVING USING TOP-DOWN DESIGN

Cont'd

```
def getMarks(stds):  
    students = list()  
    for x in range(stds):  
        mark = int(input("Enter the student mark for student"+str(x+1)+": "))  
        students.append(mark)  
    return students
```

Lecture

06

PROBLEM-SOLVING USING TOP-DOWN DESIGN

Cont'd

```
def calAverage(marks):  
    sum = 0  
    for x in marks:  
        sum += x  
    avg = sum / len(marks)  
    return avg
```

Lecture

06

PROBLEM-SOLVING USING TOP-DOWN DESIGN

Cont'd

```
def displayStdLevel(Marks, average):  
    for x in Marks:  
        if x > average:  
            print("Above the average")  
        elif x < average:  
            print("Below the average")  
        else:  
            print("Equal the average")
```

```
main()
```

Lecture

06

QUESTIONS FOR ONLINE DISCUSSION

- Write a program using function to display the multiplication table of 1 to 5.
- Write a function that takes a number argument x and returns 'A' if $x > 10$, 'B' if $0 \leq x \leq 10$, and 'C' if $x < 0$.
- Write a function to return the smallest number in a list. Don't use build-in functions.
- Write a program applying top-down approach to read the number of students and for each student the ID number and mark. Then displays the ID number and mark of the student with the highest mark.



Thank You

