

# CSEN 102

## Introduction to Computer Science

### Lecture 4: Algorithmic Problem Solving Iterative Operations

Prof. Dr. Slim Abdennadher  
Dr. Mohammed Salem, [slim.abdennadher@guc.edu.eg](mailto:slim.abdennadher@guc.edu.eg),  
[mohammed.salem@guc.edu.eg](mailto:mohammed.salem@guc.edu.eg)  
German University Cairo, Department of Media Engineering and Technology

20.10.2018 - 25.10.2018

## 1 Synopsis

### 1.1 Conditional operations

#### Synopsis – conditional operations

- Rationale
  - Determines whether or not a condition is true; and based on whether or not it is true; *selects the next step* to do
- Notation
  - Use the same primitives as before plus the following:

```
1  if condition:
2      # <operations for the then-part>
3  else
4      # <operations for the else-part>
```
- Execution
  - Evaluate `condition` expression to see whether it is true or false.
  - If true, then execute operations in `if`-part
  - Otherwise, execute operations in `else`-part

## Algorithms: operations

Algorithms can be constructed by the following operations:

- Sequential Operation
- Conditional Operation
- *Iterative Operation*

## 2 Iterative operations

### 2.1 Introduction

What is life?

*“Life is just one damn thing after another.”*

—Mark Twain

*“Life isn’t just one damn thing after another...  
it is the same damn thing over and over again.”*

—Edna St. Vincent Millay

### Iterative Operation – Loops

Repeat a set of steps over and over – also called a *looping operation*



## 2.2 Iterative operation – basics

### Iterative Operation – syntax

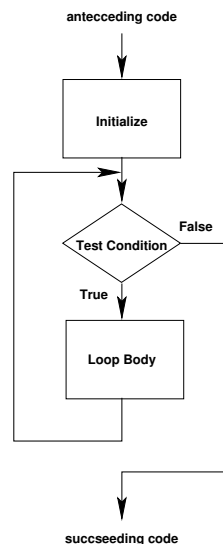
#### General Format:

```
1 while <condition>:  
2     step 1: <operation>  
3     ...  
4     step i: <operation>
```

#### Execution

1. Evaluate the condition
2. If condition is true, execute steps 1 to i, then go back to 1.
3. Otherwise, if condition is false continue the execution after the while loop.

### Iterative operation – diagram



## 2.3 Constructing iterative algorithms

### How to write a while-loop?

1. Formulate the test which tells you whether the loop needs to be run again

```
count <= 3
```

2. Formulate the actions for the loop body which take you one step closer to termination

```
print("count_is:", count)  
count = count + 1 # add one to count
```

3. In general, initialization is required before the loop and some postprocessing after the loop

```
count = 1
```

## 2.4 Iterative operations: Examples

### Iterative operations: Example I

*Example 1.* Given is a natural number  $n$ . Compute the sum of numbers from 1 to  $n$ .

```
1 n = eval(input())
2 result = 0
3 i = 1
4 while i <= n:
5     result = (result+i)
6     i = (i+1)
7 print(result)
```

### Iterative operations: Example II

*Example 2.* Write an algorithm to perform the average of  $n$  numbers entered by the user.

```
1 n = eval(input())
2 result = 0
3 i = 1
4 while (i <= n):
5     num = eval(input())
6     result = result + num
7     i = i + 1
8
9 average = result/n
10 print(average)
```

### Iterative Operation: Example III

*Example 3.* Multiplication of two integers  $N$  and  $M$  via addition

- Example:  $N = 3$  and  $M = 4$

→

N	M	result
3	4	0
3	3	3
3	2	6
3	1	9
3	0	12

```
1 N, M = eval(input()), eval(input())
2 result = 0
3 while M > 0:
4     result = result + N
5     M = M - 1
6 print(result)
```

### Iterative operations: Example IV

*Example 4.* Write an algorithm that, given a positive number  $n$ , will calculate and print the value of  $n! = n \times (n - 1) \times (n - 2) \times \dots \times 1$

```

1 n = eval(input())
2 result = 1
3 while n > 1:
4     result = (result * n)
5     n = (n - 1)
6 print(result)

```

### Iterative operations: Example V

Example 5. Write an algorithm to find the largest of 4 numbers (range 0 to 10)

```

1 max = -1
2 i = 1
3 while (i <= 4):
4     num = eval(input())
5     if (num > max):
6         max = num
7
8     i = i + 1
9
10 print(max)

```

### Iterative operations: Example VI

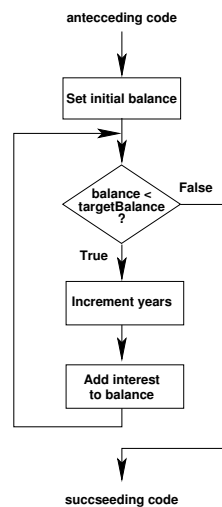
Investment with Compound Interest:

Invest 10000 Euro with 5% interest compounded annually.

Year	Balance
0	10,000.–
1	10,500.–
2	11,025.–
3	11,576.25
4	12,155.06
5	12,762.82

Question: When will the balance be *at least 20000 Euro*?

### Iterative operations: Example VI – Flowchart



## Iterative operations: Example VI – Python

### Compound interest python:

```
1 balance = eval(input())
2 rate = eval(input())
3 targetBalance = 20000
4 year = 0
5 while (balance < targetBalance):
6     year = year + 1
7     interest = balance * rate / 100
8     balance = balance + interest
9     print("The investment doubled after")
10    print(year)
11    print("years")
```

## 2.5 Iterative operations – Common Errors

### Common errors – infinite loops

*Infinite loops:*

*Example 1:*

```
while (3 > 2): <operations>
```

*Example 2:*

```
while (x <20): y = y + 1
```

If this loop is entered at all, it will run *forever*...

### Common errors – infinite loops

Why do these two algorithms *not* terminate?

```
1 i = 1
2 while i < 10:
3     print(i)

1 A = 1
2 while (A % 2 is 1): # check if A is odd
3     A = A + 2
4     print(A) # the value of A
```

### Common errors – “off by one”

Off-by-one errors

- Occur when loop executes one *too many* or *too few* times (often called “±1-errors”)
- Example: Add even integers from 2 to `number`, inclusive

```
1 number = eval(input())
2 count = 2
3 result = 0
4 while count < number:
5     result = result + count
6     count = count + 2
```

- Produces incorrect result if `number` is assigned an even `number`. Values from 2 to `number - 2` will be added (*i. e.*, `number` is excluded)
- Should be “`while (count <= number)`” in line 4!

### Common errors – “missing the target”

#### Compound interest python:

```

1 balance = eval(input())
2 rate = eval(input())
3 targetBalance = 20000
4 year = 0
5 while not balance == targetBalance:
6     year = year + 1
7     interest = balance * rate / 100
8     balance = balance + interest
9 print("The_investment_doubled_after")
10 print(year)
11 print("years")

```

Find the error in this version!

Provide for *reliable* termination of the loop!

## 2.6 Iterative operations – hints for construction

### Tracing

*ALWAYS HAND-SIMULATE* first, last and typical case through a loop

- to avoid off-by-one or infinite loop errors and
- to check the correctness of your algorithm.

## 3 Summary: Sequence, conditional, iteration

### Sequence, conditional, and iteration in one algorithm

- Remember the *Euclidean Algorithm* from lecture 1, slide 28 to determine the greatest common divisor (GCD) of two integers.
- *Method*: To find the GCD of two numbers, repeatedly replace the larger by subtracting the smaller from it until the two numbers are equal.

```

1 A, B = eval(input()), eval(input())
2 while not A == B:
3     if A > B:
4         A = A - B
5     else:
6         B = B - A
7 print("The_GCD_is_")
8 print(A)

```