CSEN 102

Introduction to Computer Science

Lecture 4:

Algorithmic Problem Solving
Iterative Operations

Prof. Dr. Slim Abdennadher
Dr. Aysha Alsafty, slim.abdennadher@guc.edu.eg,
aysha.alsafty@guc.edu.eg

German University Cairo, Department of Media Engineering and Technology

21.10.2017 - 26.10.2017

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:

```
if condition:
        # <operations for the then-part>
else
        # <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

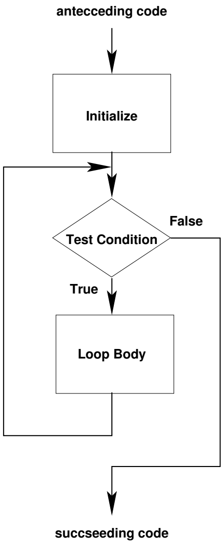
General Format:

```
while <condition>:
    step 1: <operation>
    ...
    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.

```
n = eval(input())
result = 0
i = 1
while i <= n:
    result = (result+i)
    i = (i+1)
print(result)
```

Iterative operations: Example II

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

```
n = eval(input())
result = 0
i = 1
while (i <= n):
  num = eval(input())
  result = result + num
  i = i + 1

average = result/n
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

```
N, M = eval(input()), eval(input())
result = 0
while M > 0:
    result = result + N
    M = M - 1
print(result)
```

Iterative operations: Example IV

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

```
n = eval(input())
result = 1
while n > 1:
    result = (result * n)
    n = (n - 1)
print(result)
```

Iterative operations: Example V

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

```
max = -1
i = 1
while (i <= 4):
  num = eval(input())
  if (num > max):
    max = num

  i = i + 1

print(max)
```

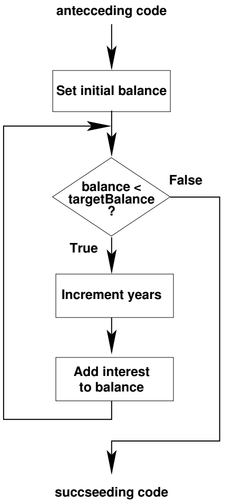
Iterative operations: Example VI

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

Compund interest python:

```
balance = eval(input())
rate = eval(input())
targetBalance = 20000
year = 0
while (balance < targetBalance):
        year = year + 1
        interest = balance * rate / 100
        balance = balance + interest
print("The investment doubled after")
print(year)
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?

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

```
A = 1
while (A % 2 is 1):  # check if A is odd
    A = A + 2
    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

```
number = eval(input())
count = 2
result = 0
while count < number:
    result = result + count
    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"

Compund interest python:

Find the error in this version!

```
balance = eval(input())
rate = eval(input())
targetBalance = 20000
year = 0
while not balance == targetBalance:
        year = year + 1
        interest = balance * rate / 100
        balance = balance + interest
print("The␣investment␣doubled␣after")
print(year)
print("years")
```

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.

```
A, B = eval(input()), eval(input())
while not A == B:
    if A > B:
        A = A - B
    else:
        B = B - A
print("The␣GCD␣is␣")
print(A)
```