CSEN 102– Introduction to Computer Science

Lecture 2:
Python
and Sequential Algorithms

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

8.10.2016 - 13.10 2016

Synopsis

- What is computer science?
- What is an algorithm?
  - An algorithm is a well-ordered collection of unambiguous and effectively computable operations that, when executed, produces a result and halts in a finite amount of time.
- What is a computing agent?

Representing algorithms

- What language to use?
  - Expressive
  - Clear, precise and unambiguous
- For example, we could use:
  - Natural Languages (*e. g.*, English)
  - Formal Programming Languages (e.g. Java, C++)
  - Something close?

Representing algorithms: Natural languages

Example

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

Representation with Natural Language

Initially, set the value of the variable `result` to `0` and the value of the variable `i` to `1`. When these initializations have been completed, begin looping until the value of the variable `i` becomes greater than `n`. First, add `i` to `result`. Then add `1` to `i` and begin the loop all over again.

Disadvantages:

- too verbose
- unstructured
- too rich in interpretation (ambiguous)
- imprecise

Representing algorithms: Formal programming language

Example

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

Representation with Formal Programming Language (Java)

```
public class Sum {
  public static void main(String[] args)
  {  int result = 0;
     int n = Integer.parseInt(args[0]);
     int i = 1;
     while (i <= n) {
       result = result + i;
       i =   i + 1;}
  System.outp.println(result);}}
```

Representing algorithms: Formal programming language

Disadvantages:

- Too many implementation details to worry about
- Too rigid syntax

Representing algorithms: Script Programing Language

A Less strict Formal Programing Language:
$\Rightarrow$ JavaScript, Python

We will use Python as it has:

- English like constructs (or other natural language) but
- is still a programming language.

Python-code: Input/output and computation

- Input operations: allow the computing agent to receive from the outside world data values to use in subsequent computations.

General Format:

```
<variable>= eval(input())
```

- Output operations: allow the computing agent to communicate results of the computations to the outside world.

General Format:

```
print(<variable>)
print("text")
```

- Computation: performs a computation and stores the result.

General Format:

```
<variable>= <expression>
```

Python: What kind of operations do we need?

Example

Given the radius of circle, determine the area.

- Decide on names for the objects in the problem and use them consistently (*e. g.*, `radius`, `area`). We call them variables
- Use the following primitive operations:
  - Get a value (input) *e. g.*, `radius = eval(input())`
  - print a value or message (output) *e. g.*,
    `print(area)`, or
    `print("Hello")`
  - Set the value of an object (*e. g.*, `Pi = 3.14`)
    $\Rightarrow$ Performs a computation and stores the result.
  - Arithmetic operations: e.g. $+$, $-$, $*$, ...

Python: Example

Example

Given the radius of circle, determine the area and circumference.

- Names for the objects:
  - Input: `radius`
  - Outputs: `area`, `circumference`
- Algorithm in Python:

```
1 radius = int(input())
2 area = (radius * radius * 3.14)
3 print(area)
4 circumference = (2 * radius * 3.14)
5 print(circumference)
```

Python: Variables

A variable is a named storage

- A value can be stored into it, overwriting the previous value
- Its value can be copied

Example

- `A = 3`
  The variable `A` holds the value three after its execution
- `A = A + 1`
  Same as: add 1 to the value of `A` (`A` is now 4)

Algorithms: operations

Algorithms can be constructed by the following operations:

- Sequential Operation
- Conditional Operation
- Iterative Operation

Sequential operations

Each step is performed in the order in which it is written

Example (1)

Write an algorithm to find the result of a division operation for the given two numbers x and y

```
1  x = int(input())
2  y = int(input())
3  quotient = x/y
4  print(quotient)
```

- Let `x=5` and `y=2`
- `quotient = 2.5`

Sequential operations

Example (2)

Algorithm for finding the average of three numbers.

```
1  A = int(input())
2  B = int(input())
3  C = int(input())
4  Sum = A + B + C
5  Average = Sum/3
6  print(Average)
```

- Let `A=12`, `B=10`, and `C=8`
- `Sum = 30`
- `Average = 10`

Sequential operations

Example (3)

The distance between two points (x_1, y_1) and (x_2, y_2) can be calculated using the following equation:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Write an algorithm that calculates the value of d.

```
1  import math
2  x1, y1 = eval(input()), eval(input())
3  x2, y2 = eval(input()), eval(input())
4  d = math.sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1))
5  print(d)
```

Python: Functions

A function is a convenient way to divide your code into useful blocks, allowing us to:

- order our code
- make it more readable
- reuse it and save some time
- way to define interfaces so programmers can share their code

You already know some Python built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

Python: Defining a Function

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function.

Sequential Operations using Functions

Example (1)

Write a function that given an input from the user, the function passes it as a parameter and prints it.

```
1  x = eval(input())
2
3  def printme(m):
4    print(m)
5
6  printme(x)
```

Sequential Operations using Functions

Example (2)

Write a function that given number of eggs, find out how many dozen eggs we have and how many extra eggs are left over.

```
1  eggs = eval(input())
2
3  def dozens(a):
4      d = int(a / 12)
5      extra = a - (d * 12)
6      print("Your number of eggs is ",d," dozen(s) and ")
7      print(extra," extra(s)")
8
9  dozens(eggs)
```

Sequential operations using Functions

Example (3)

Write a function that given two numbers, swaps the values of two numbers.

```
1  x = eval(input())
2  y = eval(input())
3
4  def dozens(a,b):
5      temp = a
6      a = b
7      b = temp
8      print(a," ", b)
9
10 dozens(x,y)
```

- Let `x = 2` and `y = 8`
- `x = 8` and `y =2`

Sequential operations using Functions

Example (4)

Write a function that given a two-digit number, returns and prints the sum of its digits.

```
1  def digitSum():
2      num = eval(input())
3      tens = int(num / 10)
4      ones = num - (tens * 10)
5      s = (tens + ones)
6      return s
7
8  print(digitSum())
```

- Let `num=45`
- `s = 9`

Where the function **`int`** rounds down the result to an integer. For

Modulus Operator

Example (5)

Write a function that given a two-digit number, returns and prints the sum of its digits.

```
1  def digitSum_Mod():
2      num = eval(input())
3      a= num % 10
4      b= num // 10
5      s = a + b
6      return s
7
8  print(digitSum_Mod())
```

- Let `num=45`
- `s = 9`