CSEN 102

Introduction to Computer Science

Lecture 2:

Python
and Sequential Algorithms

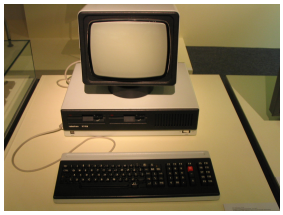
Prof. Dr. Slim Abdennadher
Dr. Nada Sharaf
Dr. Mohammed salem, slim.abdennadher@guc.edu.eg,
nada.hamed@guc.edu.eg,
mohammed.salem@guc.edu.eg

German University Cairo, Department of Media Engineering and Technology

24.10.2020 - 29.10.2020

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?



1 Sequential Algorithms

1.1 Algorithms – notation

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 1. 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 2. 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.

1.2 Python

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

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

```
radius = eval(input())
response = input("Type A for area or C for circumference")
if (response == "A"):
 area = (radius * radius * 3.14)
 print(area)
else:
 circumference = (2 * radius * 3.14)
 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 5.
- `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)

1.3 Algorithms – operations

Algorithms: operations

Algorithms can be constructed by the following operations:

- Sequential Operation
- Conditional Operation
- Iterative Operation

1.4 Algorithms – examples for sequential operations

Sequential operations

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

Example 6 (1). Write an algorithm to find the result of a division operation for the given two numbers x and y

```
x = int(input())
y = int(input())
quotient = x/y
print(quotient)
```

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

Sequential operations

Example 7 (2). Algorithm for finding the average of three numbers.

```
A = int(input())
B = int(input())
C = int(input())
Sum = A + B + C
Average = Sum/3
print(Average)
```

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

Sequential operations

Example 8 (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.

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

Sequential Operations

Example 9 (4). For a given number of eggs, find out how many dozen eggs we have and how many extra eggs are left over.

```
eggs = eval(input())
dozens = int(eggs / 12)
extras = eggs - (dozens * 12)
print("Your number of eggs is ")
print(dozens)
print(" dozen(s) and ")
print(extras)
print(" extra(s)")
```

Where the function **`int`** rounds down the result to an integer. For example **`int`**`(10/3) = 3`.

Modulus Operator

Example 10 (4). For a given number of eggs, find out how many dozen eggs we have and how many extra eggs are left over.

```
eggs = eval(input())

d = eggs//12
extra = eggs%12
print("Your number of eggs is ",d," dozen(s) and ")
print(extra," extra(s)")
```

Sequential operations

Example 11 (5). Given a two-digit number, find the sum of its digits.

```
num = eval(input())
tens = int(num / 10)
ones = num - (tens * 10)
s = (tens + ones)
print(s)
```

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

Where the function **`int`** rounds down the result to an integer. For example **`int`**`(10/3) = 3`.

Modulus Operator

Example 12 (5). Given a two-digit number, find the sum of its digits.

```
num = eval(input())
a= num % 10
b= num // 10
s = a + b

print(s)
```

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

Sequential operations

Example 13 (6). Write an algorithm that given two Strings returns their sum (concatenation)

```
x = input() # "Hello"
y = input() # " CSIS104!"
z  = x + y
print(z)    # Prints "Hello CSIS104!"
```