Lecture 1, Part 1: Introduction to Computing -- Problem Solving and Data Manipulation

Bijendra Nath Jain <bnjain@iiitd.ac.in> (Section A)

Md. Shad Akhtar <shad.akhtar@iiitd.ac.in> (Section B)

Course outline

- Part 1 Introduction to Computing and Programming (first 2 weeks):
    - Problem solving: Problem statement, algorithm design, programming, testing, debugging
    - Scalar data types: integers, floating point, Boolean, others (letters, colours)
    - Arithmetic, relational, and logical operators, and expressions
    - Data representation of integers, floating point, Boolean
    - Composite data structures: string, tuple, list, dictionary, array
    - Sample operations on string, tuple, list, dictionary, array
    - Algorithms (written in pseudo code) vs. programs
    - Variables and constants (literals): association of names with data objects
    - A language to write pseudo code
    - Programming languages: compiled vs. interpreted programming languages
    - Python as a programming language
    - Computer organization: processor, volatile and non-volatile memory, I/O

Course outline (may change a bit)

- Part 2 Algorithm design and Programming in Python (balance 11 weeks):
  - Arithmetic/Logical/Boolean expressions and their evaluations in Python
  - Input/output statements (pseudo code, and in Python)
  - Assignment statement (pseudo code, and in Python)
  - Conditional statements, with sample applications
  - Iterative statements, with sample applications
  - Function sub-programs, arguments and scope of variables
  - Recursion
  - Modules
  - Specific data structures in Python (string, tuple, list, dictionary, array), with sample applications
  - Searching and sorting through arrays or lists
  - Handling exceptions
  - Classes, and object-oriented programming
  - (Time permitting) numerical methods: Newton Raphson, integration, vectors/matrices operations, continuous-time and discrete-event simulation

Computing == Problem solving

Using ChatGPT, but suitably edited by us …

Computing encompasses the study, design and use of computer systems to acquire, process, store, communicate or manage information.
-- This leads us to conclude that computers may be used to undertake calculations, ‘store & recall’ information, exercise control other physical systems
– And one wishes to do all of this with a view to evaluate different options, take informed decisions, and implement them.

Computing == Problem solving

- Computing is about solving problems using computers
  - With a view to help evaluate different options, take informed decisions, and implement them
  - Using computer's ability to do calculations accurately or 'store & recall' information
- Example: Consider a bank branch that has (say) 3 tellers encashing checks or depositing cash for customers that are queued up in <u>ONE</u> queue
  - Current observation/understanding:
    - Today, queues are in fact somewhat long & waiting times are large
    - Cost of operating 3 tellers is high
    - Waiting times will be smaller if the number of tellers is increased
    - Waiting times will be larger if the number of tellers is decreased
  - Question: Do we increase the number of tellers to 4 or reduce the number of tellers to 2?
    - That is, what is the relationship between cost of deploying K number of tellers and resulting average waiting time
  - Undertake "discrete-event simulation" to evaluate waiting times vs. number of tellers deployed. This requires one to:
    - Compute & compare event times, t1, t2, t3, t4 to determine when will the "next event" occur
    - Compute average of N numbers (waiting times experienced by N customers)

Computing == Problem solving

- Computing is about solving problems using computers
  - With a view to help evaluate different options, take informed decisions, and implement them
  - Using computer's ability to do calculations accurately or 'store & recall' information
- Example: Consider a bank branch that has (say) 3 tellers encashing checks or depositing cash for customers that are queued up in <u>ONE</u> queue
  - Current observation/understanding:
    - Today, queues are in fact somewhat long & waiting times are large
    - Cost of operating 3 tellers is high
    - Waiting times will be smaller if the number of tellers is increased
    - Waiting times will be larger if the number of tellers is decreased
  - Question: Do we increase the number of tellers to 4 or reduce the number of tellers to 2?
    - That is, what is the relationship between cost of deploying K number of tellers and resulting average waiting time
  - Undertake "discrete-event simulation" to evaluate waiting times vs. number of tellers deployed. This requires one to:
    - Compute & compare event times, t1, t2, t3, t4 to determine when will the "next event" occur
    - Compute average of N numbers (waiting times experienced by N customers)

Computing == Problem solving

- Computing is about solving problems using computers
- Example: Consider a bank branch that has (say) 3 tellers encashing checks or depositing cash for customers that are queued up in <u>ONE</u> queue
  - Current observation/understanding:
    - Today, queues are in fact somewhat long & waiting times are large
    - Cost of operating 3 tellers is high
    - Waiting times will be smaller if the number of tellers is increased
    - Waiting times will be larger if the number of tellers is decreased
  - Question: Do we increase the number of tellers to 4 or reduce the number of tellers to 2?
    - That is, what is the relationship between cost of deploying K number of tellers and resulting average waiting time
  - Undertake "discrete-event simulation" to evaluate waiting times vs. number of tellers deployed. This requires one to:
    - Compute & compare event times, t1, t2, t3, t4 to determine when will the "next event" occur
    - Compute average of N numbers (waiting times experienced by N customers)
    - Generate sequence of random numbers
    - Maintain a list of customers waiting in queue, etc.

Computing == Problem solving

- Computing is about solving problems using computers
- Example: Consider a bank branch that has (say) 3 tellers encashing checks or depositing cash for customers that are queued up in <u>ONE</u> queue
  - Current observation/understanding:
    - Today, queues are in fact somewhat long & waiting times are large
    - Cost of operating 3 tellers is high
    - Waiting times will be smaller if the number of tellers is increased
    - Waiting times will be larger if the number of tellers is decreased
  - Question: Do we increase the number of tellers to 4 or reduce the number of tellers to 2?
    - That is, what is the relationship between cost of deploying K number of tellers and the resulting average waiting time
  - Undertake "discrete-event simulation" to evaluate waiting times vs. number of tellers deployed. This requires one to:
    - Compute & compare event times, t1, t2, t3, t4 to determine when will the "next event" occur
    - Compute average of N numbers (waiting times experienced by N customers)
    - Generate sequence of random numbers
    - Maintain a list of customers waiting in queue, etc.

Computing == Problem solving

- Computing is about solving problems using computers
- Example: Consider a bank branch that has (say) 3 tellers encashing checks or depositing cash for customers that are queued up in <u>ONE</u> queue
  - Current observation/understanding:
    - Today, queues are in fact somewhat long & waiting times are large
    - Cost of operating 3 tellers is high
    - Waiting times will be smaller if the number of tellers is increased
    - Waiting times will be larger if the number of tellers is decreased
  - Question: Do we increase the number of tellers to 4 or reduce the number of tellers to 2?
    - That is, what is the relationship between cost of deploying K number of tellers and resulting average waiting time
  - Undertake "discrete-event simulation" to evaluate waiting times vs. number of tellers deployed. This requires one to:
    - Compute & compare event times, t1, t2, t3, t4 to determine when will the "next event" occur
    - Compute average of N numbers (waiting times experienced by N customers)
    - Generate sequence of random numbers
    - Maintain a list of customers waiting in queue, etc.

Computing == …, detailed problem statement, …

- Computing requires detailed and unambiguous problem statement, including
  - Nature of input data
  - Expected output
  - And how the expected output is related to input data
- Example problem statement: Consider a bank branch that has (say) 3 tellers each encashing checks or depositing cash for customers that are queued up
  - Given:
    - No. of tellers is K
    - No. of queues is 1 (yes, one)
    - Time to encash a check or receive deposit is given by “uniform” probability distribution function, f(t)
    - Time interval between successive arrivals of 2 customers is given by “uniform” probability distribution function, g(t)
    - First-in-first-out discipline
  - Determine (using simulation):
    - The average time a customer has to wait in queue, including time to encash a check or deposit money when K = 2, 3, or 4

Computing == …, detailed problem statement, …

- Computing requires detailed and unambiguous problem statement, including
  - Nature of input data
  - Expected output
  - And how the expected output is related to input data
- Example problem statement: Consider a bank branch that has (say) 3 tellers each encashing checks or depositing cash for customers that are queued up
  - Given:
    - No. of tellers is K
    - No. of queues is 1 (yes, one)
    - Time to encash a check or receive deposit is random, but is given by “uniform” probability distribution function, f(t)
    - Time interval between successive arrivals of 2 customers is random, but is given by “uniform” probability distribution function, g(t)
    - First-in-first-out discipline
  - Determine (using simulation):
    - The average time a customer has to wait in queue, including time to encash a check or deposit money when K = 2, 3, or 4

Computing == …, design of algorithm, …

- Computing is about doing calculations accurately or storing & recalling information
  - First step is to design an algorithm that solves individual problems

    An algorithm is a sequence of "instructions" which when executed produce the expected result
  - The next step(s) would be to combine solutions to these problems
- Example of one of many requirements to simulate a bank teller:

  having computed T1, T2, T3, T4, calculate T = min(T1, T2, T3, T4), where T1, T2, T3, T4 are event time instants

```
input(T1, T2, T3, T4);
minT = T1;
if (T2 < minT) then minT = T2;
if (T3 < minT) then minT = T3;
if (T4 < minT) then minT = T4;
output(minT)
```

Computing == …, design of algorithm, …

- Computing is about its ability to do calculations accurately or store & recall information
  - First step is to design an algorithm that solves individual problems
  - The next step(s) would be combine solution to these
- Example: given T1, T2, T3, T4, compute min(T1, T2, T3, T4)

  Above algorithm may be re-written as a sub-program of function "**MinTime**"

```
define function MinTime(T1, T2, T3, T4);
    [minT= T1;
    if (T2 < minT) then minT = T2;
    if (T3 < minT) then minT = T3;
    if (T4 < minT) then minT = T3;
    return(minT)]

 nextEventTime = MinTime(65.0, 87.1, 26, 75.0)
 output(nextEventTime)
```

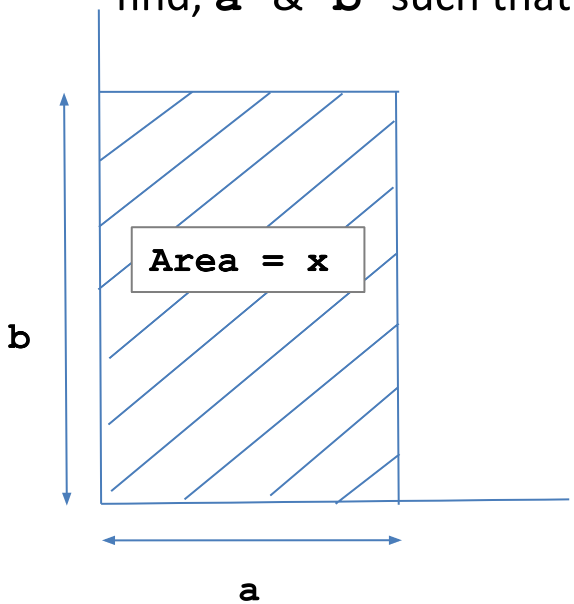
Output:

```
26.0
```

Computing == …, design of algorithm, …

- Computing is about its ability to do calculations accurately or store & recall information
  - First step is to design an algorithm that solves individual problems
  - The next step(s) would be combine solution to these
- Another example: computing the square root **`y = √x,`** where **`x > 0`**
- Its algorithm is based on solving the problem:

  find, **`a`** & **`b`** such that **`a*b = x,`** and **`a = b`**

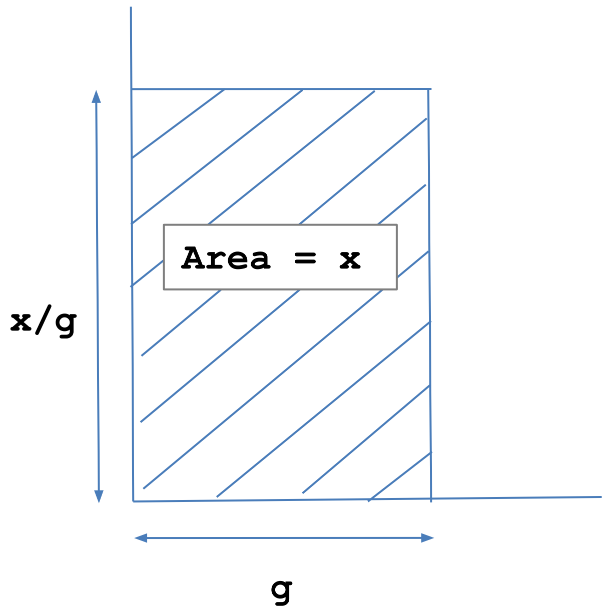


Computing == …, design of algorithm, …

- Computing is about its ability to do calculations accurately or store & recall information
    - First step is to design an algorithm that solves individual problems
    - The next step(s) would be combine solution to these
- Another example: computing the square root **`y = √x`**, where **`x > 0`**

  Somewhat <u>informal</u> version of an algorithm

```
1.  Start with a guess, g = x/2        # for instance
2.  if |g*g – x| is small              # such as < 10^-6 * x
    then [conclude g = √x; output(g); exit]
    else [compute new guess g = (g + x/g)/2; repeat step 2]
```



Computing == …, design of algorithm, …

- Computing is about its ability to do calculations accurately or store & recall information
  - First step is to design an algorithm that solves individual problems
  - The next step(s) would be combine solution to these
- Another example: computing the square root $y = \sqrt{x}$, where $x > 0$

  Somewhat <u>informal</u> version of an algorithm

```
1.  Start with a guess, g = x/2        # for instance
2.  if |g*g – x| is small              # such as < 10^-6 * x
    then [conclude g = √x; output(g); exit]
    else [compute new guess g = (g + x/g)/2; repeat step 2]
```

Let x = 3, example outcome after 3 rounds

Round	g	|g*g-x|
1	1.5	0.75
2	1.75	0.0625
3	1.732143	0.000319

Computing == …, design of algorithm, …

- Computing is about its ability to do calculations accurately or store & recall information
  - First step is to design an algorithm that solves individual problems
  - The next step(s) would be combine solution to these
- Another example: computing the square root $y = \sqrt{x}$, where $x > 0$

  Somewhat <u>informal</u> version of an algorithm

```
1.  Start with a guess, g = x/2        # for instance
2.  if |g*g – x| is small              # such as < 10^-6 * x
    then [conclude g = √x; output(g); exit]
    else [compute new guess g = (g + x/g)/2; repeat step 2]
```

Let x = 16, example outcome after 4 rounds

Round	g	|g*g-x|
1	8	48
2	5	9
3	4.1	0.81
4	4.00122	0.009758

Computing == …, design of algorithm, …

- Computing is about its ability to do calculations accurately or store & recall information
  - First step is to design an algorithm that solves individual problems
  - The next step(s) would be combine solution to these
- Another example: computing the square root **y = √x**, where **x > 0**

  Refined & formal version of the earlier algorithm

```
define function sqrt(x, epsilon);
     [g = x/2;
     while |g*g – x| > epsilon do
           [g = (g + x/g)/2)]
     return(g)]
root = sqrt(3, 0.001)
output(root)
```

Output

```
1.732
```

In-class Exercise 1.1 Section A

- Follow the link
- For Section A: https://tinyurl.com/y6j8b4de
- For Section B: https://tinyurl.com/mr45pp8u

Computing == …, converting algorithm into program, …

- Computing is about its ability to do calculations accurately or store & recall information
  - First step is to design an algorithm that solves individual problems
  - Second step is convert the algorithm into a Python program

    But, before you convert, you need to know Python programming well
- Example: compute **`minT = min[T1, T2, T3, T4]`**

```
input(T1, T2, T3, T4);
minT = T1;
if (T2 < minT) then minT = T2;
if (T3 < minT) then minT = T3;
if (T4 < minT) then minT = T4;
output(minT)
```

```
T1 = float(input('Time 1? '))
T2 = float(input('Time 2? '))
T3 = float(input('Time 3? '))
T4 = float(input('Time 4? '))
minT = T1
if(T2 < minT):
    minT = T2
if(T3 < minT):
    minT = T3
if(T3 < minT):
    minT = T3
print('NextEvent ', minT)
```

https://tinyurl.com/3fhkje4u

Computing == …, testing the program, …

- Computing is about its ability to do calculations accurately or store & recall information
  - First step is to design an algorithm that solves individual problems
  - Second step is convert the algorithm into a Python program

    But, before you convert, you need to know Python programming well
- Example: compute **`minT = min[T1, T2, T3, T4]`**
- Save the 'script' as a file **`MinTime.py`**
- & run **`MinTime.py`** using different data such as:

  `(T1,T2,T3,T4) = (23,43,56.5,133)`

```
T1 = float(input('Time 1? '))
T2 = float(input('Time 2? '))
T3 = float(input('Time 3? '))
T4 = float(input('Time 4? '))
minT = T1
if(T2 < minT):
    minT = T2
if(T3 < minT):
    minT = T3
if(T3 < minT):
    minT = T3
print('NextEvent ', minT)
```

https://tinyurl.com/3fhkje4u

Computing == …, testing the program, …

- Computing is about its ability to do calculations accurately or store & recall information
  - First step is to design an algorithm that solves individual problems
  - Second step is convert the algorithm into a Python program

    But, before you convert, you need to know Python programming well
- Example: compute **`minT = min[T1, T2, T3, T4]`**
- Save the 'script' as a file **`MinTime.py`**
- & test run **`MinTime.py`** using different data e.g.

  `(T1,T2,T3,T4) = (23,43,56.5,133)`
  - Try other permutations as well
  - What would happen if

  `(T1,T2,T3,T4) = (23,43,-56.5,0.0)`

```
T1 = float(input('Time 1? '))
T2 = float(input('Time 2? '))
T3 = float(input('Time 3? '))
T4 = float(input('Time 4? '))
minT = T1
if(T2 < minT):
    minT = T2
if(T3 < minT):
    minT = T3
if(T3 < minT):
    minT = T3
print('NextEvent ', minT)
```

https://tinyurl.com/3fhkje4u

Computing == …, running a Python program, …

pythontutor.com/render.html#mode=display

GMAIL IIITD HDFC bank Bank PayTM SBI Direct ABSL MF MF MF MF Corp IDFC Kotak Mirae

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
known limitations

```
T1 = float(input('Time 1? '))
T2 = float(input('Time 2? '))
T3 = float(input('Time 3? '))
T4 = float(input('Time 4? '))
minT = T1
if(T2 < minT):
    minT = T2
if(T3 < minT):
    minT = T3
if(T3 < minT):
    minT = T3
print('NextEvent ', minT)
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 4 of 10

Print output (drag lower right corner to r

```
Time 1? 32
Time 2? 45
Time 3? 21
```

Frames Objects

Global frame

T1	32.0
T2	45.0
T3	21.0

Computing == …, documenting the program, …

- Computing is about its ability to do calculations accurately or store & recall information
  - First step is to design an algorithm that solves individual problems
  - Second step is convert the algorithm into a Python program
  - Third step is document the algorithm and/or the Python program
- Example: compute **`T = min (T1, T2, T3, T4)`**

```
# MinTime.py computes minT = min(T1,T2,T3,T4) and print minT
# Treats inputs T1,T2,T3,T4 as floating-point numbers
# Input T1,T2,T3,T4 may be integers or floating-point numbers
T1 = float(input('Time 1? '))
T2 = float(input('Time 2? '))
. . .
minT = T1          # an assumption, to be confirmed or rejected later
if(T2 < minT):
    minT = T2          # guess is updated if T2 is a likely min
if(T3 < minT):
    minT = T3          # guess is updated if T3 is a likely min
. . .
print('NextEvent  at time ', minT)
```

Q&A

- On algorithms
- On re-writing algorithms as Python programs
- On testing Python programs
- On documenting Python programs