Introduction to Computer Science

Subodh Sharma
svs@cse.iitd.ac.in
https://subodhvsharma.github.io

IIT Delhi, Computer Science Department

History Trivia

- **Language, Recursion, Transformation**: The earliest use of "formalized" grammar was by Panini (5^{th} century BC) in Ashtadhyayi.

History Trivia

- **Language, Recursion, Transformation**: The earliest use of "formalized" grammar was by Panini (5^{th} century BC) in Ashtadhyayi.
- **Computers and Programs**:

History Trivia

- **Language, Recursion, Transformation**: The earliest use of "formalized" grammar was by Panini (5^{th} century BC) in Ashtadhyayi.
- **Computers and Programs**:
  1. **mechanical calculators** started with Pascal and refined by Leibniz. Refer to Pascal's calculator and Leibniz's digital arithmometer!

History Trivia

- **Language, Recursion, Transformation**: The earliest use of "formalized" grammar was by Panini (5^{th} century BC) in Ashtadhyayi.
- **Computers and Programs**:
  1. **mechanical calculators** started with Pascal and refined by Leibniz. Refer to Pascal's calculator and Leibniz's digital arithmometer!
  2. The first forms of a general purpose computer: the **analytical engine** in 1837 by Charles Babbage. It was only a design!

History Trivia

- **Language, Recursion, Transformation**: The earliest use of "formalized" grammar was by Panini (5^{th} century BC) in Ashtadhyayi.
- **Computers and Programs**:
  1. **mechanical calculators** started with Pascal and refined by Leibniz. Refer to Pascal's calculator and Leibniz's digital arithmometer!
  2. The first forms of a general purpose computer: the **analytical engine** in 1837 by Charles Babbage. It was only a design!
  3. First general purpose computer by Konrad Zuse in 1941, called Z3.

History Trivia

- **Language, Recursion, Transformation**: The earliest use of "formalized" grammar was by Panini (5^{th} century BC) in Ashtadhyayi.
- **Computers and Programs**:
  1. **mechanical calculators** started with Pascal and refined by Leibniz. Refer to Pascal's calculator and Leibniz's digital arithmometer!
  2. The first forms of a general purpose computer: the **analytical engine** in 1837 by Charles Babbage. It was only a design!
  3. First general purpose computer by Konrad Zuse in 1941, called Z3.
  4. Ada Lovelace wrote the first **computer program** to calculate Bernoulli numbers using the description of Babbage's machine.

Last Lecture's Summary

- What is computing?

Last Lecture's Summary

- What is computing?
- What are computing tools?

Last Lecture's Summary

- What is computing?
- What are computing tools?
- What are the essential aspects of a computational process?

Last Lecture's Summary

- What is computing?
- What are computing tools?
- What are the essential aspects of a computational process?
- What are algorithms, programming languages and programs?

Computing Model:Programming Models

Types of Programming Models:

- **Functional**: A program is specified as amathematical expression.

Computing Model:Programming Models

Types of Programming Models:

- **Functional**: A program is specified as amathematical expression.
- **Imperative**: A program is specified by a sequence of commands.

Computing Model:Programming Models

Types of Programming Models:

- **Functional**: A program is specified as amathematical expression.
- **Imperative**: A program is specified by a sequence of commands.

Computing Model:Programming Models

Types of Programming Models:

- **Functional**: A program is specified as amathematical expression.
- **Imperative**: A program is specified by a sequence of commands.

Various programming languages support the above models. **Python** is an imperative PL. However, we will use it to understand both the programming models.

Programming Models: Primitive Expressions

The simplest objects and operations in the computing model. These include

- **Basic data elements**: numbers, characters, boolean, etc.

Programming Models: Primitive Expressions

The simplest objects and operations in the computing model. These include

- **Basic data elements**: numbers, characters, boolean, etc.
- **Basic operations**: addition, subtraction, multiplication, string operations, etc.

Programming Models: Primitive Expressions

The simplest objects and operations in the computing model. These include

- **Basic data elements**: numbers, characters, boolean, etc.
- **Basic operations**: addition, subtraction, multiplication, string operations, etc.
- **Naming mechanism**: Named expressions to be used without repetition

Programming Models: Combination and Abstraction

- **Combination**: Composition of functions, Inductive definitions, etc.

Programming Models: Combination and Abstraction

- **Combination**: Composition of functions, Inductive definitions, etc.
- **Abstraction**: Named functions, data structures, classes, modules, etc.

Functional Programming:Factorial

Mathematical Definition:

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ 1 \times 2 \times \ldots \times n & \textit{otherwise} \end{cases}$$

Functional Programming:Factorial

Mathematical Definition:

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ 1 \times 2 \times \ldots \times n & \textit{otherwise} \end{cases}$$

Using induction in the definition, we get:

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ n \times (n-1)! & \textit{otherwise} \end{cases}$$

Functional Programming:Factorial

Python Program:

```
def factorial(x):
        if x ==1:
                return 1
        else:
                return x * factorial(x-1)
```

Functional Programming:Factorial

Python Program:

```
def factorial(x):
        if x ==1:
                return 1
        else:
                return x * factorial(x-1)
```

- **Variable**: A named entity which represents (or stores) a value. Eg: x is the input variable.

Functional Programming:Factorial

Python Program:

```
def factorial(x):
        if x ==1:
                return 1
        else:
                return x * factorial(x-1)
```

- **Variable**: A named entity which represents (or stores) a value. Eg: x is the input variable.
- **Function Declaration**: With a keyword **def**

Functional Programming:Factorial

Python Program:

```
def factorial(x):
      if x ==1:
            return 1
      else:
            return x * factorial(x-1)
```

- **Variable**: A named entity which represents (or stores) a value. Eg: x is the input variable.
- **Function Declaration**: With a keyword **def**
- **return keywors**: Returns an output of an expression

Functional Programming:Factorial

Python Program:

```
def factorial(x):
        if x ==1:
                return 1
        else:
                return x * factorial(x-1)
```

- **Variable**: A named entity which represents (or stores) a value. Eg: `x` is the input variable.
- **Function Declaration**: With a keyword **def**
- **return keywors**: Returns an output of an expression
- **Condition**: A relational expression that evaluates to either `true` or `false`

Functional Programming:Factorial

Python Program:

```
def factorial(x):
    if x ==1:
        return 1
    else:
        return x * factorial(x-1)
```

- **Variable**: A named entity which represents (or stores) a value. Eg: `x` is the input variable.
- **Function Declaration**: With a keyword **def**
- **return keywors**: Returns an output of an expression
- **Condition**: A relational expression that evaluates to either `true` or `false`

Functional Programming:Factorial

Python Program:

```
def factorial(x):
    if x ==1:
        return 1
    else:
        return x * factorial(x-1)
```

- **Variable**: A named entity which represents (or stores) a value. Eg: `x` is the input variable.
- **Function Declaration**: With a keyword **def**
- **return keywors**: Returns an output of an expression
- **Condition**: A relational expression that evaluates to either `true` or `false`

With Python, one can work in two modes:

- **Interactive**: Executes one statement at a time; the results of previously executed statements are in active memory.
- **Compiled**: The entire program is *interpreted* into an executable object

Functional Programming:Factorial

Are all mathematical definitions computable? What about the following?

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ (n+1)!/(n+1) & \textit{otherwise} \end{cases}$$

To answer this question, we must first understand how the program was **evaluated**.

$$3! = 3 * (3-1)! = 3 * (2 * (2-1)!) = 3 * (2 * (1))$$

The recursive function evaluation indicates a **"defered" computation**!

What is not an algorithm

$$sqrt(n) = \begin{cases} m & \text{if } m * m = n \\ 0 & \text{if } \nexists m : m * m = n \end{cases}$$

The above is mathematically valid specification, yet it is not an algorithm! Why?

What is not an algorithm

$$sqrt(n) = \begin{cases} m & \text{if } m * m = n \\ 0 & \text{if } \nexists m : m * m = n \end{cases}$$

The above is mathematically valid specification, yet it is not an algorithm! Why? The description does not tell us how to evaluate the function.

Interpreter Demo

- Primitive Operator: $+, *, /, //, \%, \ldots$
- Primitive Relations: $=, \geq, \leq, <, >, != , and, or, not$