session1_summary.md

Programming Training

Program	Programming Concepts - Python
Session	Sunday, 24 June 2018, 14:00 - 18:30

Overview

In this program, we will focus on `problem solving` and `computational thinking`, instead of implementation details or language syntax. Programming languages are mostly used to translate our ideas in solving a problem into instructions that computer understands. In theory, you can use any language to achieve the same outcome. Furthermore, languages are changing rapidly nowadays, hence it's much better to focus on the computational thinking. We will use `Python` to facilitate the learning in this program, with a little bit of `C/C++` to understand concepts which are not so obvious in Python.

The Big Picture: How Program Works

```
Input --> Algorithm --> Output
```

The above is the simplification of how program works. Given an input and desired output, we have a "blackbox" called algorithm that transforms the input into the desired output. We will focus on how to implement this magic box called algorithm.

How to Represent Input and Output

Unlike human that understand languages like English and Bahasa, computer doesn't. That's why we need to understand the way input and output are represented in computer.

At the core of it, we have `binary system`. Human are using `decimal system (0 to 9)` to represent number, but in binary, we only have 2 symbols, `0 and 1`, this is to represent whether electricity current passes through the wire (1), or not (0).

Some example of conversion between base 10 (decimal) and base 2 (binary):

> $321_{10} = 3 * 10^2 + 2 * 10^1 + 1 * 10^0$

> $101_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 5_{10}$

Going one step further, we can represent characters in binary form too. One of the standards being used is `ASCII (American Standard Code for Information Interchange)`. Basically it is mapping each character into decimal number, which will be converted into binary so it can be understood by computers.

> A = 65_{10} = 01000001

> B = 66_{10} = 01000010

More about ASCII

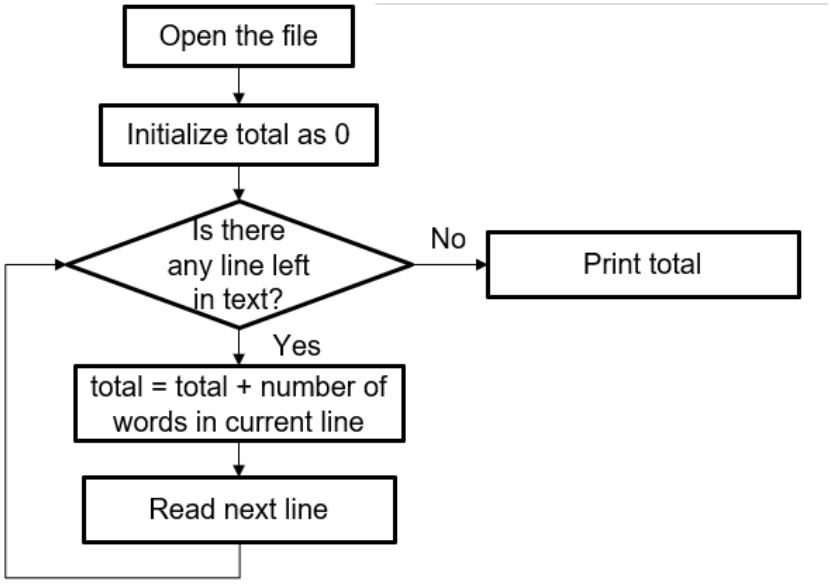
Similarly, for pictures, we can break it down into pixels. For each pixel, we can represent the colors with some color scheme. For RGB, each color is represented with a number ranging from 0 to 255, hence we can represent each of them with 3 numbers.

Algorithm

After understanding how data are represented in computers, it's time to focus on the algorithm itself. In short, we can define algorithm as `step-by-step instructions to solve a problem`. The daily life example would be recipe books. It consists of `detailed`, `non-ambiguous`, and `step-by-step` instructions to achieve something, in this case: cooking a dish. Similarly, computational thinking is about formulating step-by-step instructions to solve a problem.

Flowchart

In order to capture the logic or procedure, one of the ways is using flowchart. It is a graphical representation of the step-by-step instructions.



Some funny video on flowchart

Pseudocode

As you might observe, having graphical representation is nice, but it could be very tedious to draw as the size of the program grows. This is where pseudocode comes in handy. Basically we are using human language to depict the logic of the program. It looks like code, but not exactly an actual code, hence the term `pseudo`.

```
Open the file
lines <- read lines in file
total <- 0
For each line in lines:
        counter <- number of words in line
        total <- total + counter
Print total
```

Introduction to Python

Some of the reasons why we choose Python for this introductory course are:

- Python is more intuitive to understand, similar with human language (to some extend)
- Less syntax to worry about, relatively :)

Operators

Assignment Operator

Assignment operator, `=`, is used to assign a value into a variable

```
a = 20
b = "Hi, there!"
```

Arithmetic Operators

More or less the same with arithmetic operators that we know in maths.

```
+ for addition
- for subtraction
* for multiplication
/ for floating-point number division (in Python 3)
```

```
// for integer division (in Python 3)
** for power
% for modulo operation
```

Relational Operators

Again, very similar with what we have in math. :)

```
< <= == > >= !=
== is for equality
!= is for non-equality
```

Logical Operators

There are only 2 values for boolean data type, namely `True` and `False`. In terms of logical operators, we will focus on these 3 operators for now.

```
not --> for negation
and --> for conjunction
or  --> for disjunction
```

String

In python, string can be delimited with both single quotes or double quotes.

```
print('Hi, there!')
print("Hello, there!")
```

Indexing

Indexing is accessing a single character of string. Note: In programming, we start counting from 0, even though some (very small subset) of programming languages count from 1. :)

```
a = "Hi, there!"
a[0] will be 'H'
a[4] will be 't'
a[-1] will be '!' --> Python support reversed indexing
```

Slicing

Slicing is accesing a block of character of string.

```
a = "Hi, there!"
a[4:7] will be 'the' --> starting from 4, up to before 7
a[:5]  will be 'Hi, t'
a[4:]  will be 'there!'
```

Len Operator

We can get the length of a string by using len operator.

```
a = "Hi, there!"
print(len(a)) --> will produce 10
```

String is immutable object in Python

After a string value is assigned to a variable, we can't change the elements of that string. This is because in Python, string is immutable object (can't be modified), we need to reassign the string for modification.

```
a = "abc"
a[1] = 'd' --> error
a = "adc"  --> this is okay
```

Concatenation Operator

We can concatenate strings with `+` operator.

```
a = 'abc'
b = 'def'
a = a + b --> a will be "abcdef"
```

List

List is `array` in Python. In Python, there is no restriction for list to contain elements with the same data type, which is the case for other languages such as C++ and Java.

```
a = [1, 2, 3]
b = ["abc", 2, 10, True, False]
```

`Indexing`, `Slicing`, `len` and `concatenation` work in similar way as in the case for string.

List is Mutable

Unlike strings, list is mutable object, hence we can modify the elements of list.

```
a = [1, "abc", [2, 3]]
a[0]    = 2      --> valid
a[1]    = "def"  --> valid
a[1][0] = 'e'    --> invalid
a[2][1] = 1      --> valid
```

Nested list

We can have list inside a list, as many levels as we want.

```
a = [1, 2, [3, 4, [5], 6, [7, 8]]]
a[0]       --> 1
a[2][0]    --> 3
a[2][4][1] --> 8
```

More on introduction to Python