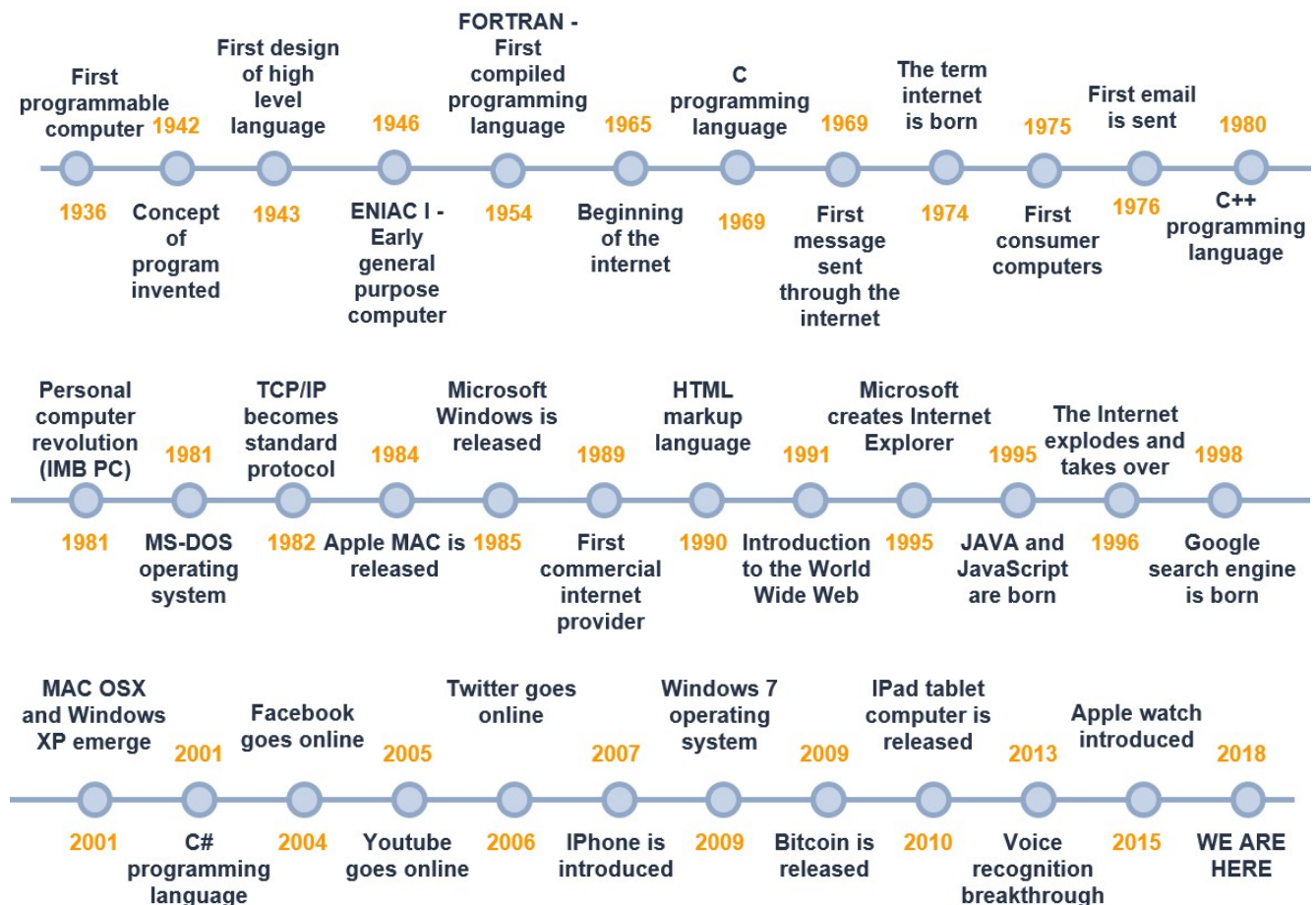


## 1 Introduction to Web Development

No. of Classes: 16 hours

Introduction to the academy and what it offers, short history about programming and how the web works and Q&A session.

### 1.1 TIMELINE OF COMPUTERS AND THE WEB



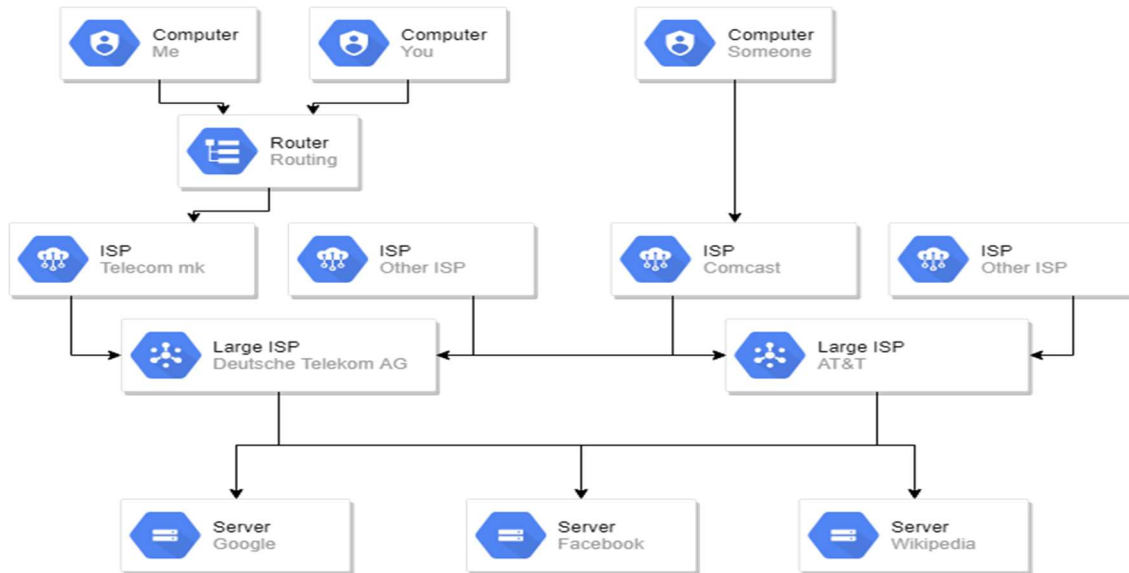
### 1.2 WHAT IS INTERNET AND HOW DOES IT WORK?

- It is not ONE SINGLE thing that everyone access
- It is not a huge company
- We don't open the internet by opening google

# INTRODUCTION TO WEB DEVELOPMENT – SESSION I

DAY 1 – WEDNESDAY 17<sup>TH</sup> OCTOBER 2018

WEB DEVELOPER PROGRAM – SEDC



---

❖ **Programmer:** is a person who fixes a problem that you don't know you have, in a way you don't understand.

---

**Algorithm is:**

- The sequence of instructions used to solve a specified problem in a finite amount of time with finite amount of data.
  - Written (or verbal) description of logical sequence of actions, applied to specific objects.
- 

## 1.3 WEB DEVELOPER CATEGORIES

- 
- ❖ **Front End:** Developers that create or maintain the client side of the application. This side of the application is everything that the user of the application sees and can interact with.
- ❖ **Back End:** Developers that create or maintain the server side of the application. This side of the application is all the things that the user can't see or interact like databases and server mechanisms like request handling and security.
- 

## 1.4 BLOGS OR NEWSLETTERS I CAN FOLLOW

- ✓ [Hacker news](#)
- ✓ [Joel on software](#)
- ✓ [Coding horror](#)
- ✓ [Fabulous adventures in coding](#)
- ✓ [Programmer's digest](#)
- ✓ [dev.to](#)

## 1.5 BOOKS THAT I SHOULD READ

- ✓ The pragmatic programmer
- ✓ Code: The Hidden Language of Computer Hardware and Software
- ✓ Code complete
- ✓ Clean Code
- ✓ Head-first Design patterns
- ✓ Secrets of the JavaScript Ninja
- ✓ C# In Depth

## 1.6 TOOLS THAT I NEED

1. You need to have Notepad.exe – You definitely have notepad in your machine, except if it's from World War 1.
2. You need to have a way to INTERPRET or COMPILE your code in order to produce result. Browser-based client-side code (HTML, CSS, JavaScript) is interpreted, therefore the browser engine will take care of it when presenting.

# 1.7 PROBLEM SOLVING AND DECISION MAKING

Problem solving and how humans and machines handle it and introduction to programming, languages and programmers.

## 1.7.1 What is Programming?

- The process of writing a sequence of instructions (algorithms) with a chosen language to be executed by a computer
- Part of the process for creating software and applications
- The process of implementing solutions by writing code

## 1.7.2 What Does a Programmer Do?

### They solve problems

- The programmer begins the programming process by analysing the problem, breaking it into manageable parts, and developing a general solution for each piece called algorithm.
- Usually, at minimal, computer problems could be only one set of logical instructions (one algorithm), but sometimes programmers need to write and combine multiple sets of logical instructions (algorithms) to solve certain problem.

### They create software programs

- Complex software programs could be consisted of thousands or millions of algorithms that all together solve a lot of problems through automation and therefore creating intelligence.
- At the end of this academy program, you'll be able to write complex software applications with abnormal complexity.

## 1.7.3 Virtues Of a Programmer

- **Hubris / Vanity:** Excessive pride that drives programmers to create programs, the quality that makes you write programs that other people won't want to say bad things about, that makes you go back to a working program saying "I can do better". And that makes you strive to write the next great piece of software.
- **Impatience:** "Do it right away" attitude and restlessness. The anger you feel when the computer is being lazy. What you can and do right now is what matters, not what will happen "someday". Perfect is the enemy of good (especially good enough). This makes you write programs that don't just react to your needs, but actually anticipate them. This makes you write programs that other people will enjoy using
- **Laziness:** The power to make the computer do work instead of you and the ability to work less but efficient rather than more and inefficient. The quality that makes you go to great effort to reduce overall energy expenditure. It makes you make the computer work instead of you. It makes you write labour-saving programs that other people will find useful. It makes you document what you wrote so you don't have to answer so many questions about it.
- **Humility:** The quality that makes you recognize greatness when you see it. Makes you ask stupid questions and get labour-saving results. Makes you go back to a working program

and replace your code with someone else's superior implementation. That makes you ultimately become a better programmer than you were six months ago.

- **Endurance:** The ability to keep going through a seemingly bizarre impossible situation, to test and stress an application over and over again until it fails. This makes you write programs that won't crash as soon as people start actually using them, and create robust and reliable programs.
- **Perseverance:** A virtue that drives programmers to invest energy and focus on a difficult single task problem for a prolonged amount of time, makes you able to focus on an issue to the exclusion of the rest of the world, and sometimes do the computers work to show that it's wrong.

## 1.8 PROGRAMMING LANGUAGES

---

❖ **Programming language:** A formal language designed to communicate instructions to a machines, particularly a computer. They are used to create programs and control the behavior of the machines. It's the way how we 'talk' to Computers, and sometimes to each other.

---

### 1.8.1 What They Are Made of

---

❖ **Syntax:** The text of the language, which has its own reserved keywords and basic rules of how to combine them. It's a textual representation of the instruction that we give to the machine.

---

Ex.: What is **x** ... Where do we put **;** ... What code do we enclose in **{ }**

---

❖ **Semantics:** The meaning of the language and its features. It is what creates limits and logic behind the text and the structure.

---

Ex.: Is **x** a type that holds numbers? If so how many numbers can it store?

---

### 1.8.2 Low-Level Programming Languages

They are machine-dependent, which means every machine has its own unique machine language. Machine languages are the natural languages of the computers. Instructions are set of 1s and 0s.

Assembly languages are one level above machine languages and usually consist simple and crude syntaxes. Assembly language allows programmers to write machine code using symbolic programming.

### 1.8.3 High-Level Programming Languages

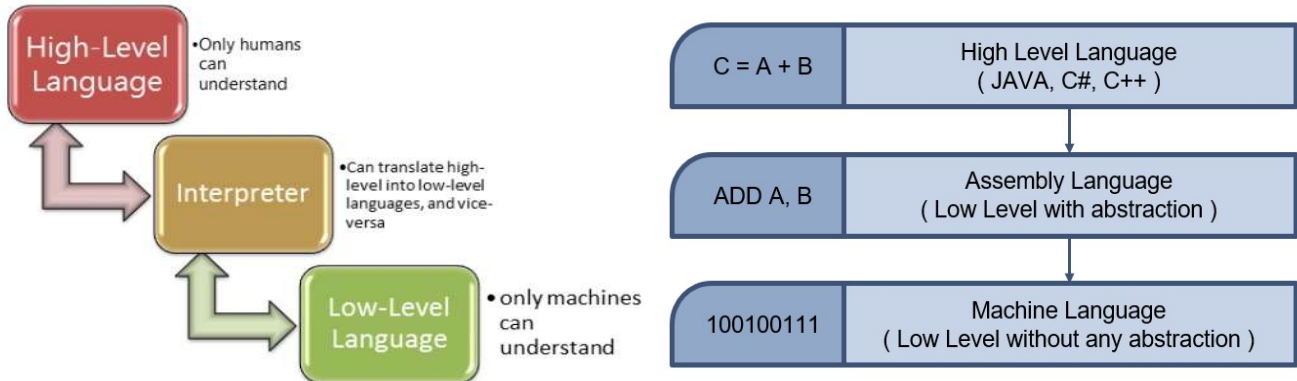
They are machine independent, and can be executed on various computers. High-level languages allow programmers to write programs which are very similar to the natural English language.

---

❖ **Source Code:** is the code written in high-level languages.

❖ **Object Code:** is the code written in low-level languages.

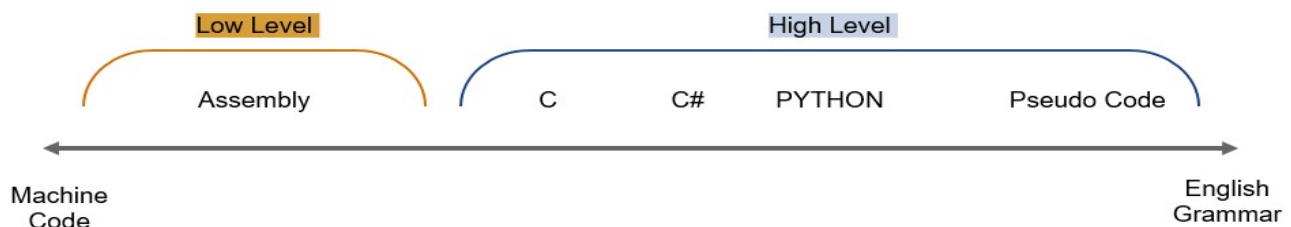
---



## 1.8.4 Types of Translators

In order source code to be translated successfully into machine code, we need a process of compilation. The process of translation helps creating from source code to end-result object, which is the executable output used to represent the program developed by the programmer. There are two types of translators.

- ❖ **Compiler:** a translation program that convert the programmer's entire high-level program (source code), into a machine language code (object code). This translation process is called **COMPILATION**.
- ❖ **Interpreter:** a translation program that converts each program statement (line by line) into machine code on the fly, just before the program statement is to be executed. Translation and execution occur immediately, one after another, one statement at a time.



## 1.9 NAMING THINGS


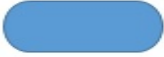










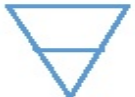


- ❖ Must be **descriptive**. Must be **unique**. Must **not** be **too short**. Must **not** be **too long**. Must **follow standards**.
- ❖ Should **reveal intention**. Should **avoid disinformation**. Should be **meaningful**. Should be **pronounceable**.
- ❖ Should **use nouns and verbs correctly** (*GetName, SetName, ChangeName*). Should **have one word per concept** (*get, fetch, retrieve, controller, manager, driver*). Should **have correct opposites** (*Start with Stop, Open with Close*).
- ❖ Should **be consistent**. Should **use solution / problem domain names correctly**.
- ❖ Abstractions are useful tools. Abstractions are powerful tools. Abstractions can be dangerous. Abstractions can be leaky.

- ❖ **Camel Case:** writing compound words so each word in the middle begins with a capital letter.



## 1.10 FLOWCHART

It is a type of diagram that represents an algorithm, workflow or process. The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem.

Symbol	Symbol Name	Description
	Flow lines	Flow lines are used to connect symbols used in flowchart and indicate direction of flow.
	Terminal (START / STOP)	This is used to represent start and end of the flowchart.
	Input / Output	It represents information which the system reads as input or sends as output.
	Processing	Any process is represented by this symbol. For example, arithmetic operation, data movement.
	Decision	This symbol is used to check any condition or take decision for which there are two answers. Yes (True) or No (False).
	Connector	It is used to connect or join flow lines.
	Off-page Connector	This symbol indicates the continuation of flowchart on the next page.
	Document	It represents a paper document produced during the flowchart process.
	Annotation	It is used to provide additional information about another flowchart symbol which may be in the form of descriptive comments, remarks or explanatory notes.
	Manual Input	It represents input to be given by a developer or programmer.
	Manual Operation	This symbol indicates that the process has to be done by a developer or programmer.
	Online Storage	It represents online data storage such as hard disks, magnetic drums or other storage devices.
	Offline Storage	It represents offline data storage such as sales on OCR, data on punched cards.
	Communication Link	It represents the data received or to be transmitted from an external system.
	Magnetic Disk	It represents data input or output from and to a magnetic disk.

# INTRODUCTION TO WEB DEVELOPMENT – SESSION 2

DAY 2 – MONDAY 22<sup>ND</sup> OCTOBER 2018

WEB DEVELOPER PROGRAM – SEDC

Flowchart Symbol	Explanation
Flowlines 	Flow of the direction
Start / Stop (terminator) 	A start and an end of a module. For some end for a main module and exit for other module
Processing 	Processing block, for the calculation, and other step of instructions.
Input / Output (I/O) 	Input and output data to from the computer memory.
Decision 	Usually has one entrance and two exit.
Process Module 	Task that process in different place, module
Automatic Counter loop 	The number of loop execute will start with A, counter incremented by S, and until the end value B
Connector on page  off page	Flowchart can be sectioned, the connector will show where the section continue

Name	Symbol	Use in flowchart
Oval		Denotes the beginning or end of a program.
Flow line		Denotes the direction of logic flow in a program.
Parallelogram		Denotes either an input operation (e.g., INPUT) or an output operation (e.g., PRINT).
Rectangle		Denotes a process to be carried out (e.g., an addition).
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes (e.g., IF/THEN/ELSE).

Picture	Shape	Name	Action Represented
	Oval	Terminal Symbol	Represents start and end of the Program
	Parallelogram	Input/Output	Indicates input and output
	Rectangle	Process	This represents processing of action. Example, mathematical operator
	Diamond	Decision	Since computer only answer the question yes/no, this is used to represent logical test for the program
	Hexagon	Initialization/Preparation	This is used to prepare memory for repetition of an action
	Arrow Lines & Arrow Heads	Direction	This shows the flow of the program
		Annotation	This is used to describe action or variables
	Circle	On page connector	This is used to show connector or part of program to another part.
	Pentagon	Off-page connector	This is used to connect part of a program to another part on other page or paper

Process	Preparation	Alternate Process
Decision	Manual Input	Subprocess
Document	Online Storage	Predefined Process
Start/Stop	Display	Connector
Input/Output	Tape	Connector
Manual Operation	Disk	
		Directional Arrows Flow Lines
		Connecting Line

Flowcharts are:

- Shows the logic of an algorithm
- Emphasizes individual steps and their interconnections
- Visualizes control flow from one action to the next



## 1.11 HOW CAN WE THINK LIKE PROGRAMMERS?

Introduction to basic programming concepts and methodologies and developing a programming mindset through pseudocode and flowcharts. The programmer should have a mindset of:

- ❖ A willingness to experiment.
- ❖ An acceptance that you will make mistakes.
- ❖ An understanding that your improvement will not be linear.

## 1.12 PSEUDOCODE

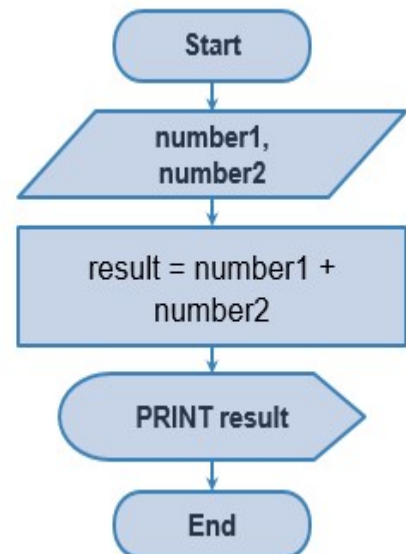
- ❖ Term used to describe writing your computer instructions in plain English to the point it is readable by anyone.
- ❖ It focuses on what should happen rather than how it happens
- ❖ Two main priorities:
  - It should be easy to read
  - It should be easy to understand what it is doing
- ❖ Pseudocode, means Fake in Latinic.

### Example:

(1): Person inputs two numbers in the program. The program sums the first and the second number and adds them in a result. The program returns the result to the user.

(2): Input => number1, number2  
 result = number1 + number2  
 Output => result

(3): Begin {  
     a = input();  
     b = input();  
     result = a + b;  
     return result;  
   } End



## 1.13 PHASES OF MAKING A PROGRAM

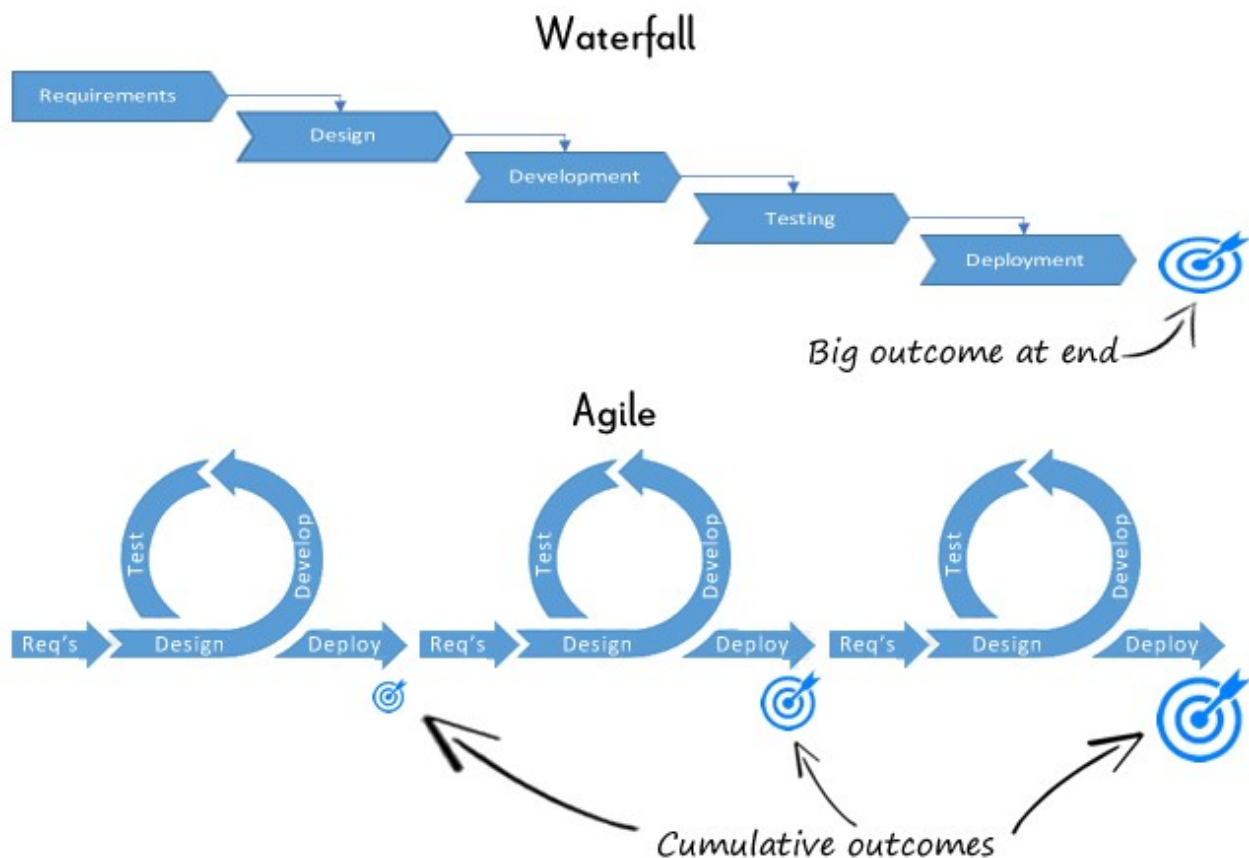
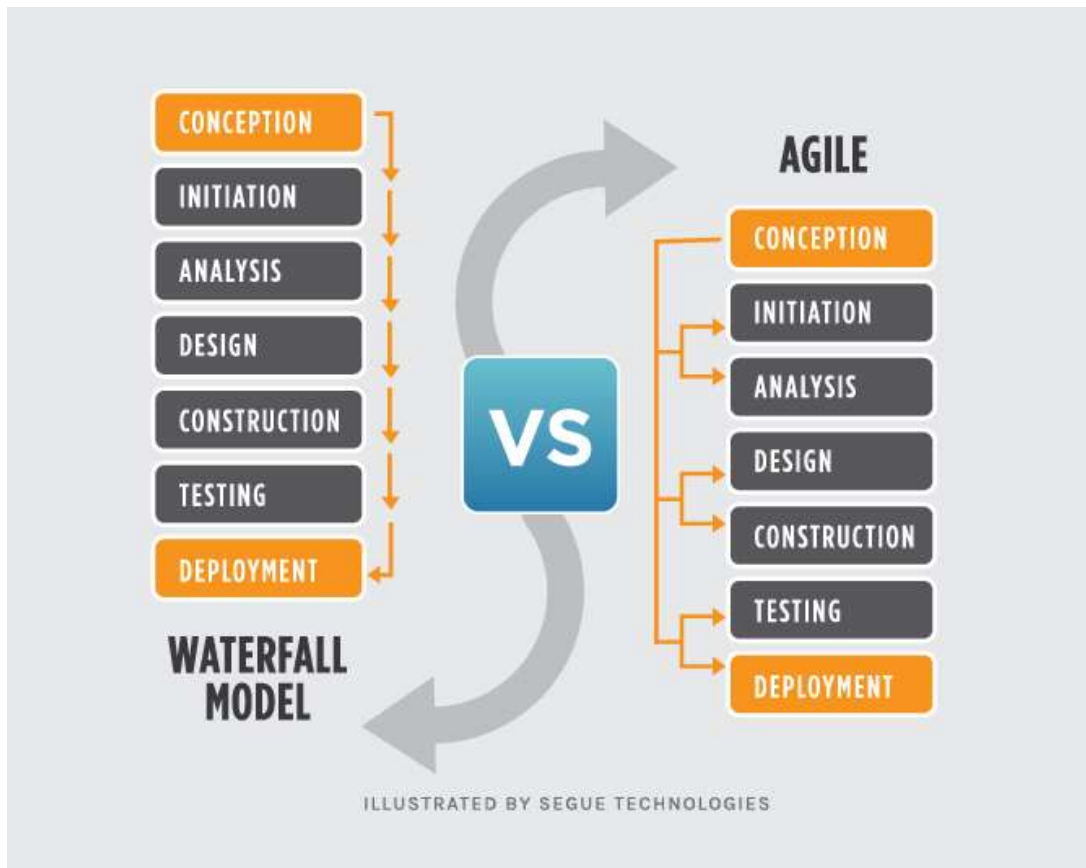
A typical programming task can be divided into several phases:

- ❖ **Clarification/Analysis phase:** make sure we have everything we need to know.
- ❖ **Problem solving phase:** produce an ordered sequence of steps that describe the solution.
- ❖ **Implementation phase:** implement the program in some programming language.
- ❖ **Testing phase:** verify that our code makes sense and actually works.
- ❖ **Maintenance phase:** verify that our code will need a minimal amount of change.

## INTRODUCTION TO WEB DEVELOPMENT – SESSION 3

DAY 3 – WEDNESDAY 24<sup>TH</sup> OCTOBER 2018

WEB DEVELOPER PROGRAM – SEDC



## 1.14 MAKING DECISIONS

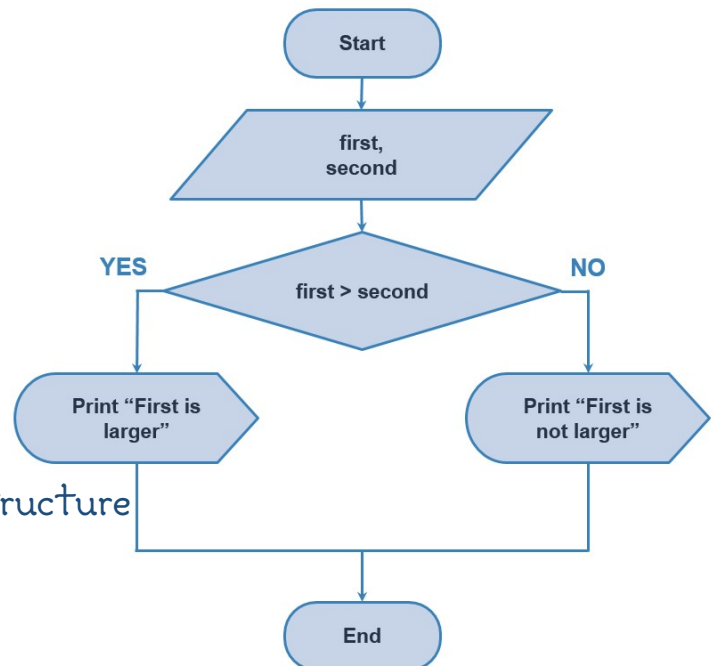
- ❖ We have things called **logical expressions**.
- ❖ They can only be **true** or **false**.
- ❖ Their value can always be determined.
- ❖ “If” can describe a **condition** we want to test.
- ❖ We can make decision based on its outcome.
- ❖ We can take different action based on the value of the condition.

### 1.14.1 IF - THEN - ELSE Structure

```
if condition then
true alternative action
else
false alternative action
end-if
```

#### Example:

```
If first > second then
    print "First is larger"
else
    print "First is not larger"
End-if
```



### 1.14.2 NESTED IF - THEN - ELSE Structure

```
if condition then
true alternative action
else
if condition then
true alternative action
else
false alternative action
end-if
end-if
```

#### Example:

```
If first > second then
    print "First is bigger"
else
    If first < second then
        print "Second is bigger"
    else
        print "They are equal"
    End-if
End-if
```

## 1.15 MAKING NON-BINARY DECISIONS

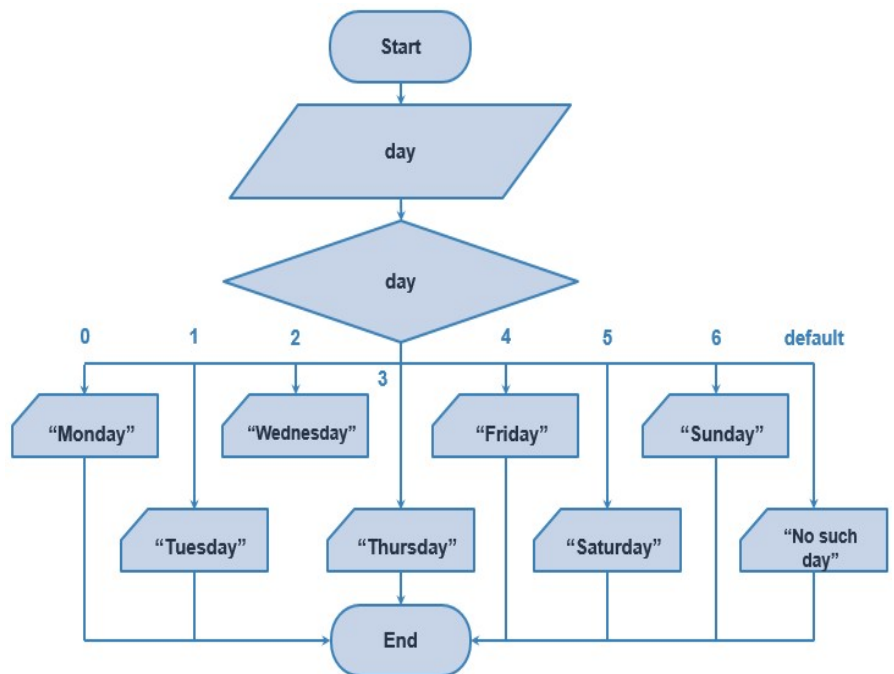
- Sometimes we have **more than two outcomes**.
- Implemented very differently in different languages.
- Fortunately, very similar in C# and JavaScript.
- We can take different actions based on the value of a variable.

### 1.15.1 Switch Structure

```
switch expression
case value1:
value1-statements
break
case value2:
value2-statements
break
...
default
default-statements
break
end-switch
```

#### Example:

```
switch day
case 0:
    print "Monday"
    break
case 1:
    print "Tuesday"
    break
...
default
    print "No such day"
    break
end-switch
```



## 1.16 RUNNING CODE MORE THAN ONCE

- Computers are great at repetitive work.
- If we can solve it logically from one step to the next, we can get the computer to do the repeating of the process.
- We can solve otherwise insolvably tedious problems easily.

## 1.16.1 For Structure

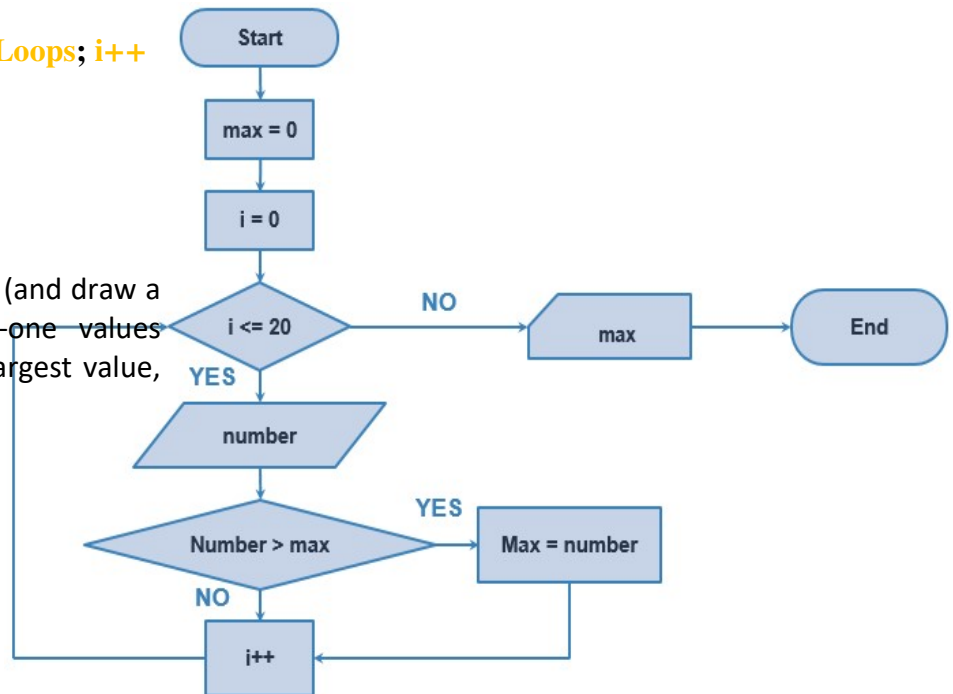
```
for initial-expression; test-expression; loop-expression
loop-statments
end-for
```

### Example:

```
for var i=0; i<numberOfLoops; i++
    loop-statements
end-for
```

**Task (1):** Write an algorithm (and draw a flowchart) to read twenty-one values from input, determine the largest value, and print it.

```
var max=0;
for var i=0; i<21; i++
    input number
    if number>max
        max=number
    end-if
end-for
print max
```



## 1.17 COLLECTIONS

- Computers are also **great for working with repeatable data**.
- Collections (**lists, arrays**) are a way of storing related data together, and easily accessing it as needed.
- Access is usually done with a **numerical index**.
- The index is (almost always) zero-based.

```
var collection = [ item1, item2, item3... ]
collection[0] = newItem
```



## Example

```
var students = [ ... ];
for var i=0; i<students.length; i++
    if students[i].length != 0
        print student[ i ]
    end-if
end-for
```

## 1.18 HAVING A REPEATABLE SUB-PROBLEM

- Sometimes we need the code to run more than once.
- Also we might not know how many times.
- We might need not a number of executions, but a condition to be met.

### 1.18.1 While Structure

```
while (expression)
while-statements
end-while
```

### 1.18.2 Do-While Structure

```
do
while-statements
while (expression)
```

**Task II:** Write an algorithm to determine a student's final grade and indicate whether it is passing or failing. The final grade is calculated as the average of four marks.

**Start**

**Input** mark1, mark2, mark3, mark4

**Calculate** marksAverage = (mark1 + mark2 + mark3 + mark4) / 4

**If** marksAverage >= 3

**Then** Print "Pass"

**Else** Print "Fail"

**End-if**

**End**

## 1.19 PROBLEM SOLVING IS A PROCESS

How to approach problems and create solutions, the process of making a program and exercises with pseudocode

### 1.19.1 Some Problems Programmers Face

- Naming things is really hard
- Finding mistakes and broken things ( bugs )
- Fixing mistakes and broken things ( bugs )

## 1.20 THE PROCESS OF PROBLEMS SOLVING

- Understanding the problem
- Devising a plan
- Execution of the plan
- Looking back

### 1.20.1 Understanding the Problem

- What is the unknown?
- What is the data?
- What is the condition? Is it possible to satisfy the condition?
- Is the condition sufficient to determine the unknown?
- Draw a diagram. Introduce suitable notation.

### 1.20.2 Division a Plan

- Have you seen the problem before? Do you know a related problem?
- Look at the unknown. Think of a problem having the same or similar unknown.
- Split the problem into smaller sub-problems.
- If you can't solve it, solve a more general version, or a special case, or part of it

### 1.20.3 Execution of the Plan

- Carry out your plan of the solution. Check each step.
- Can you see clearly that the step is correct?
- Can you prove that it is correct?

### 1.20.4 Looking Back

- Can you check the result?
- Can you derive the result differently?
- Can you use the result, or the method, for some other problem?