Coding Bootcamp 4

Alliance For Digital Employability

Lesson 1 & 2

Introduction to Programming

(Common)

Student Name: .....

Coding Bootcamp4

Formal Methods
Propositional Calculus, Predicate Calculus
Binary and Other Systems
Regular Expressions

Fundamental Theme

- What are the capabilities and limitations of computers and computer programs?
  - What can we do with computers/programs?
  - Are there things we cannot do with computers/programs?

Studying the Theme

- How do we prove something **CAN** be done by **SOME** program?
- How do we prove something **CANNOT** be done by **ANY** program?

Languages

The terms *language* and *word* are used in a strict technical sense in this course:

A ***language*** is a set of words.

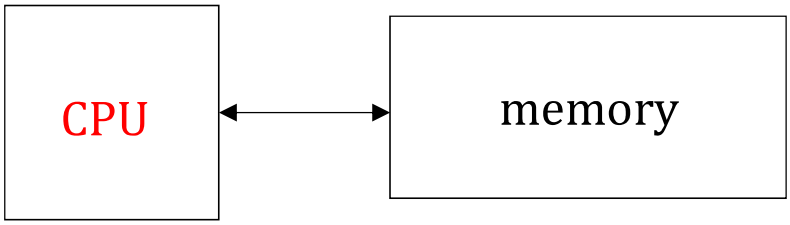
A ***word*** is a sequence (or string) of symbols.

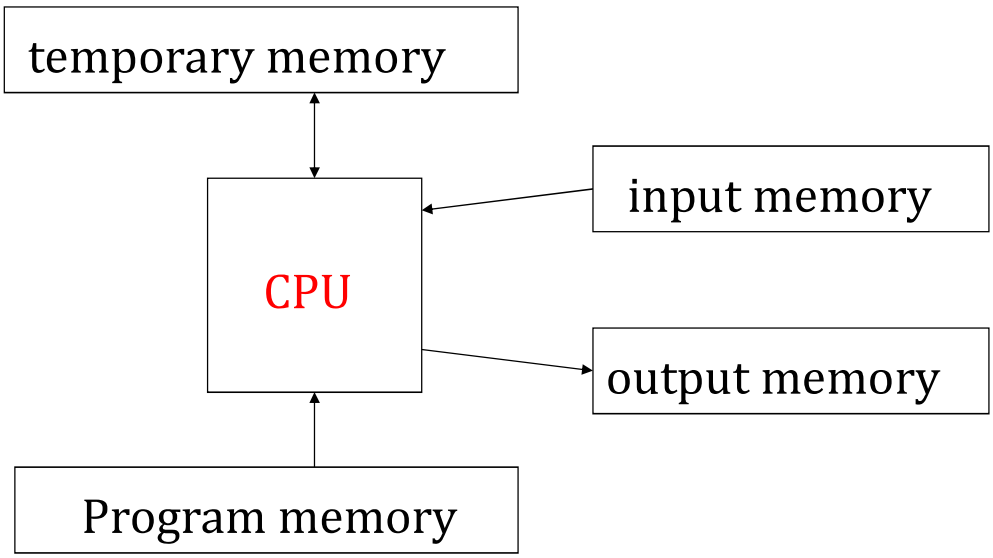
ε (or λ) denotes the ***empty word***, the sequence of zero symbols.

Symbols and Alphabets

- What is a symbol, then?
- Anything, but it has to come from an ***alphabet*** which is a ***finite*** set.
- A common (and important) instance is $\sum = \{0, 1\}$.
- ε , the empty word, is ***never*** a symbol of an alphabet.

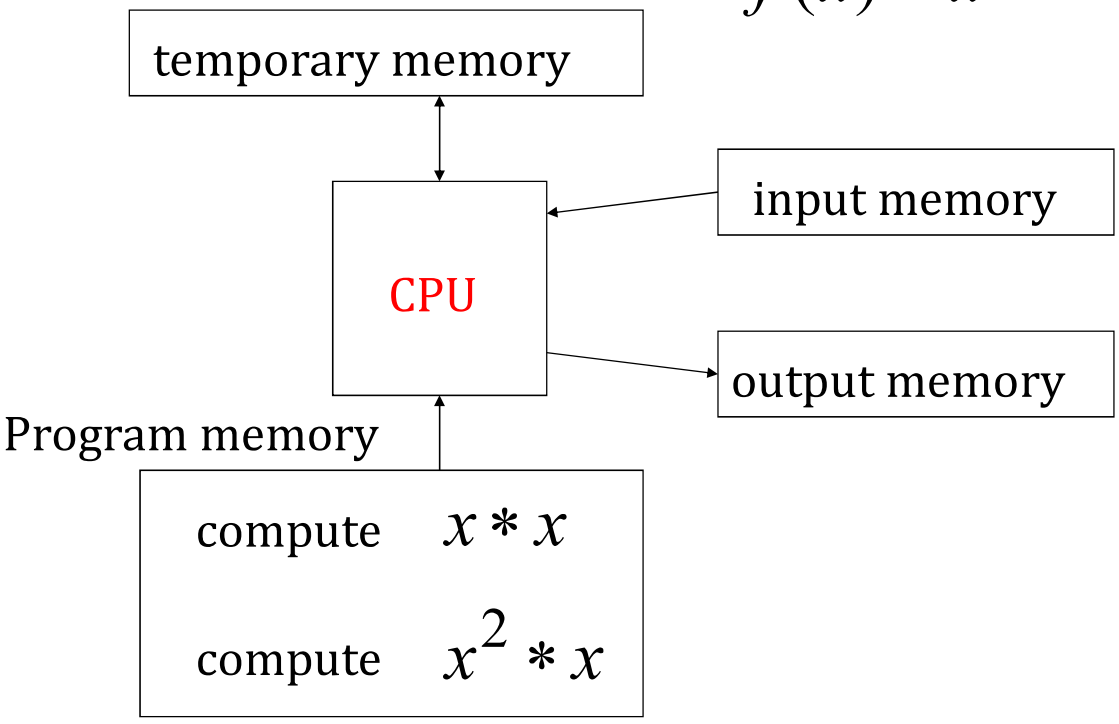
Computation





Example:

$$f(x) = x^3$$



$$f(x) = x^3$$

temporary memory

input memory

$x = 2$

CPU

output memory

Program memory

compute $x * x$

compute $x^2 * x$

temporary memory

$z = 2 * 2 = 4$

$f(x) = z * 2 = 8$

$$f(x) = x^3$$

input memory

$x = 2$

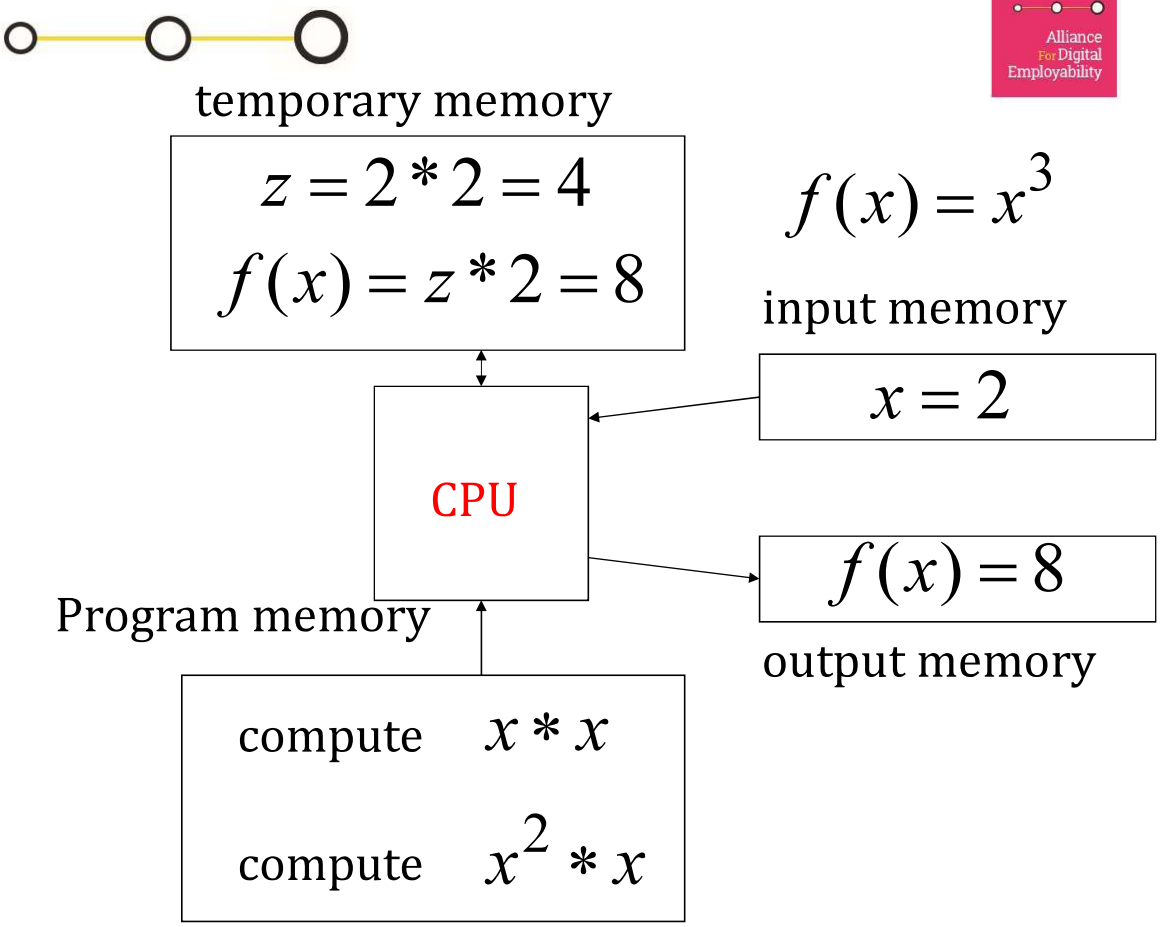
CPU

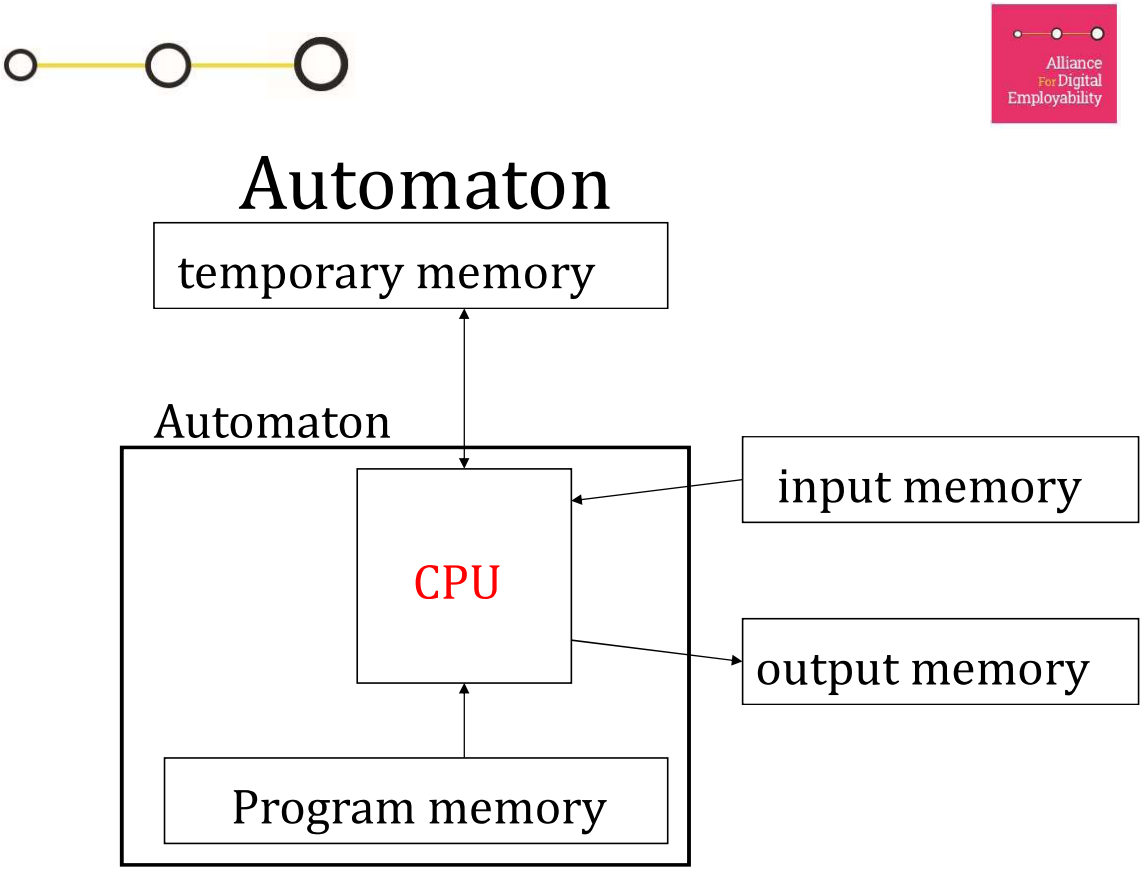
output memory

Program memory

compute $x * x$

compute $x^2 * x$


temporary memory
$z = 2 * 2 = 4$
$f(x) = z * 2 = 8$
$f(x) = x^3$
input memory
$x = 2$
CPU
$f(x) = 8$
Program memory
output memory
compute $x * x$
compute $x^2 * x$

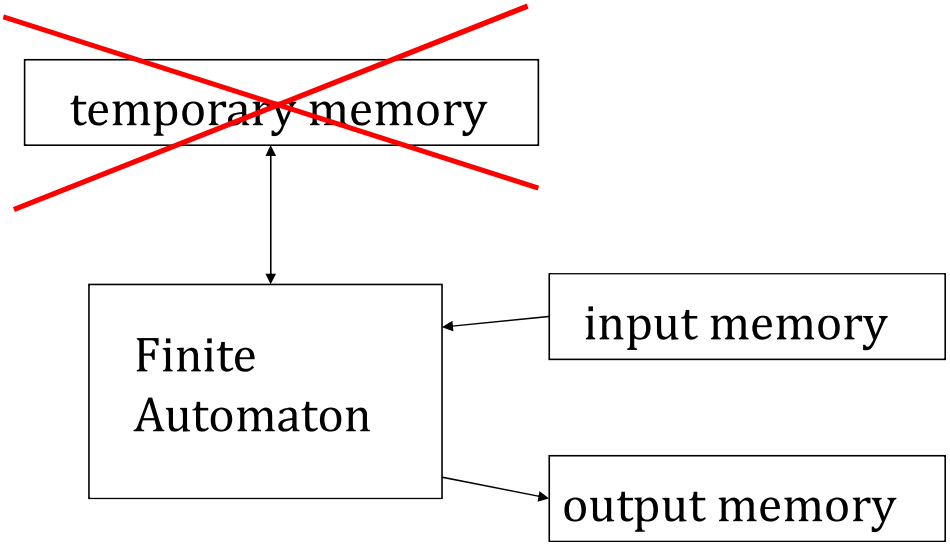

Automaton
temporary memory
Automaton
input memory
CPU
output memory
Program memory

Different Kinds of Automata

Automata are distinguished by the temporary memory

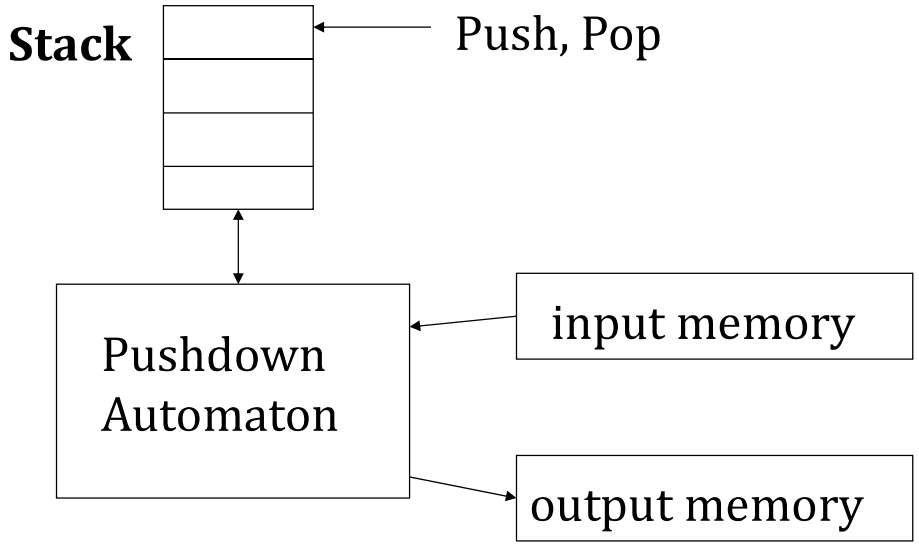
- **Finite Automata**: no temporary memory
- **Pushdown Automata**: stack
- **Turing Machines**: random access memory

Finite Automaton



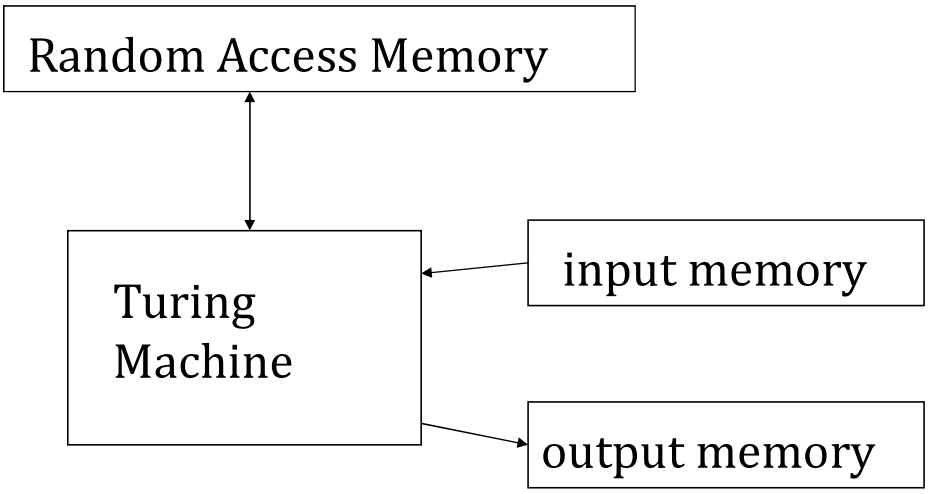
Example: Vending Machines
(small computing power)

Pushdown Automaton



Example: Compilers for Programming Languages
(medium computing power)

Turing Machine



Examples: Any Algorithm
(highest computing power)

Power of Automata

Finite Automata < Pushdown Automata < Turing Machine

Less power ⟶ More power

Solve more computational problems

Mathematical Preliminaries

Mathematical Preliminaries

- Sets
- Functions
- Relations
- Graphs
- Proof Techniques

SETS

A set is a collection of elements

$$A = \{1, 2, 3\}$$

$$B = \{train, bus, bicycle, airplane\}$$

We write

$$1 \in A$$

$$ship \notin B$$

Set Representations

C = { a, b, c, d, e, f, g, h, i, j, k }

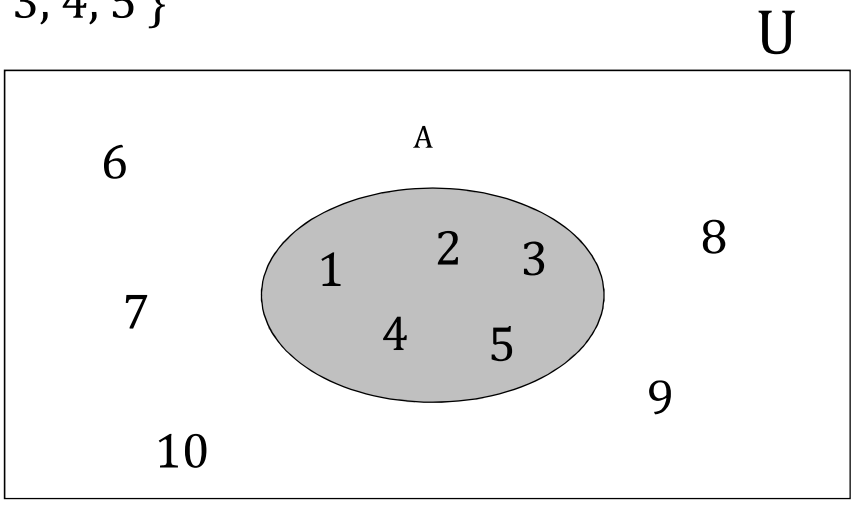
C = { a, b, …, k } ⟶ *finite set*

S = { 2, 4, 6, … } ⟶ *infinite set*

S = { j : j > 0, and j = 2k for some k>0 }

S = { j : j is nonnegative and even }

A = { 1, 2, 3, 4, 5 }



Universal Set: all possible elements

U = { 1 , … , 10 }

Set Operations

$A = \{ 1, 2, 3 \}$ $B = \{ 2, 3, 4, 5\}$

- Union

  $A \cup B = \{ 1, 2, 3, 4, 5 \}$

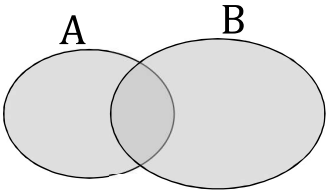
- Intersection

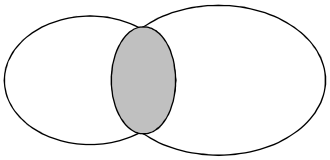
  $A \cap B = \{ 2, 3 \}$

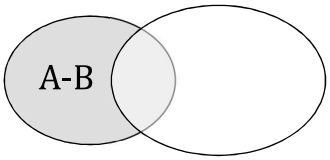
- Difference

  $A - B = \{ 1 \}$

  $B - A = \{ 4, 5 \}$



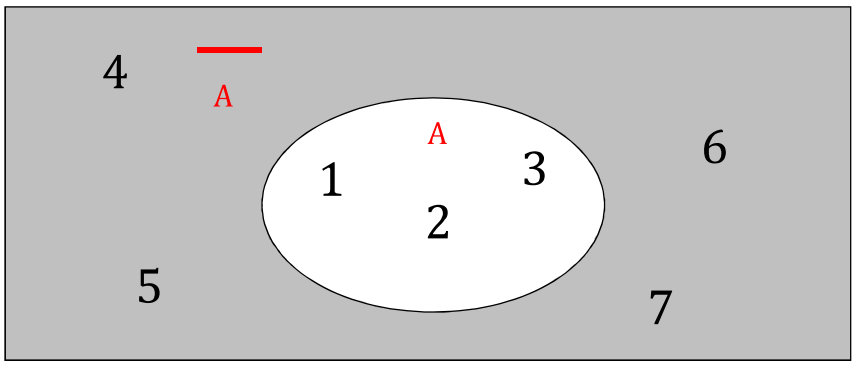




- Complement

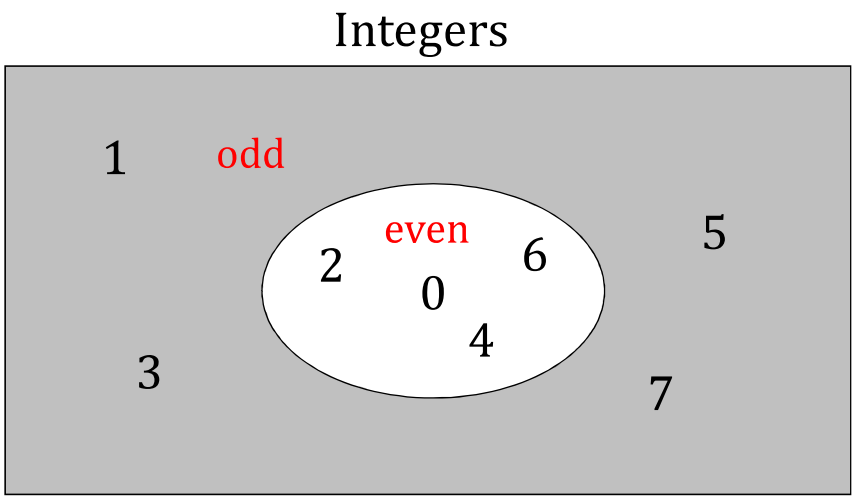
  Universal set = {1, …, 7}

  $A = \{ 1, 2, 3 \} \Rightarrow \overline{A} = \{ 4, 5, 6, 7\}$



$\overline{\overline{A}} = A$

$$\overline{\{\text{even integers}\}} = \{\text{odd integers}\}$$



DeMorgan's Laws

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

Empty, Null Set: $\emptyset$

$\emptyset = \{\}$

$S \cup \emptyset = S$

$S \cap \emptyset = \emptyset$

$\overline{\emptyset}$ = Universal Set

$S - \emptyset = S$

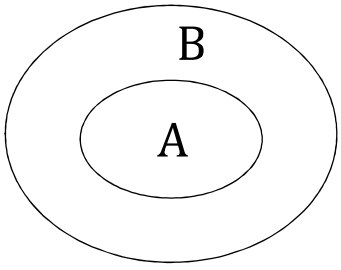
$\emptyset - S = \emptyset$

Subset

$A = \{ 1, 2, 3\}$ $B = \{ 1, 2, 3, 4, 5 \}$

$A \subseteq B$

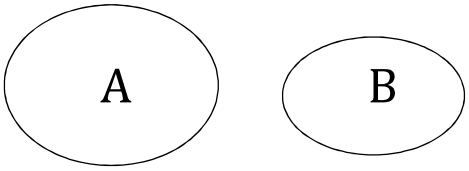
Proper Subset: $A \subset B$



Disjoint Sets

$A = \{ 1, 2, 3 \}$ $B = \{ 5, 6\}$

$A \cap B = \emptyset$



Set Cardinality

- For finite sets

  $A = \{ 2, 5, 7 \}$

  $|A| = 3$

Powersets

A powerset is a set of sets

S = { a, b, c }

Powerset of S = the set of all the subsets of S

2^S = {∅, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c} }

Observation: $| 2^S | = 2^{|S|}$ ($8 = 2^3$)

Cartesian Product

A = { 2, 4 } B = { 2, 3, 5 }

A X B = { (2, 2), (2, 3), (2, 5), (4, 2), (4, 3), (4, 5) }

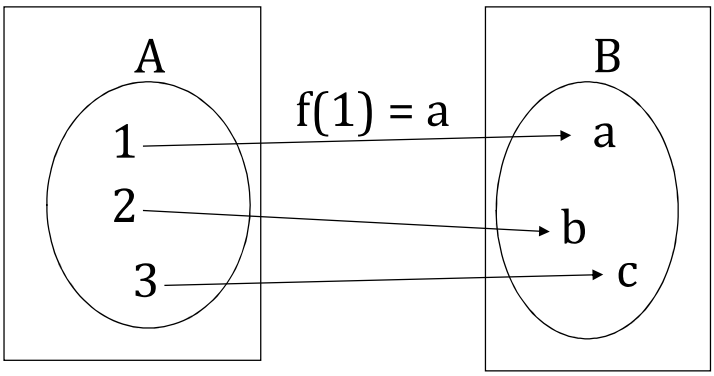
|A X B| = |A| |B|

Generalizes to more than two sets

A X B X … X Z

FUNCTIONS

domain range



f : A -> B

If A = domain
then f is a total function
otherwise f is a partial function

RELATIONS

$R = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots\}$

$x_i \; R \; y_i$

e. g. if R = '>': 2 > 1, 3 > 2, 3 > 1

In relations x_i can be repeated

Equivalence Relations

- Reflexive: x R x
- Symmetric: x R y ⟹ y R x
- Transitive: x R y and y R z ⟹ x R z

Example: R = '='

- x = x
- x = y ⟹ y = x
- x = y and y = z ⟹ x = z

Equivalence Classes

For equivalence relation R

equivalence class of x = {y : x R y}

Example:

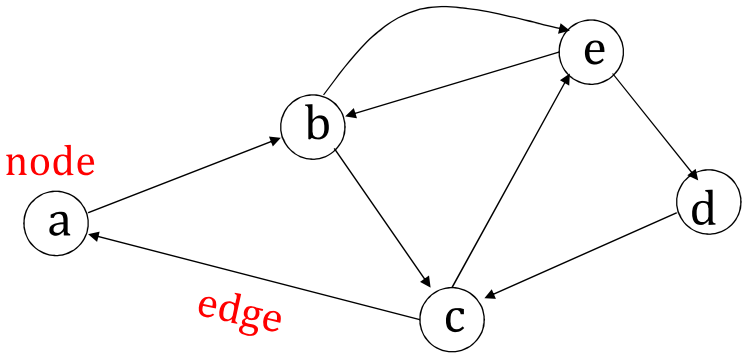
R = { (1, 1), (2, 2), (1, 2), (2, 1),
(3, 3), (4, 4), (3, 4), (4, 3) }

Equivalence class of 1 = {1, 2}

Equivalence class of 3 = {3, 4}

GRAPHS

A directed graph



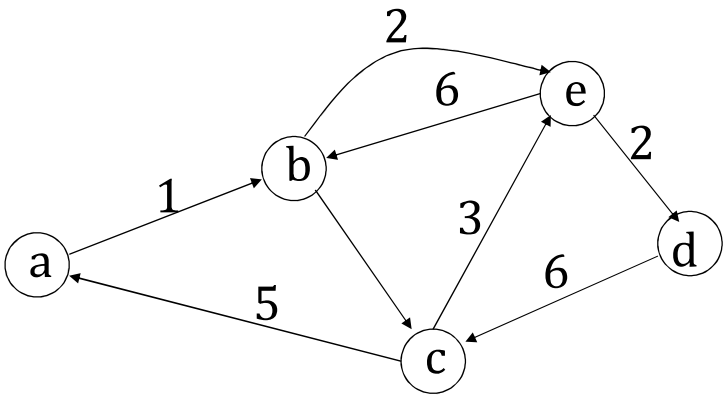
- Nodes (Vertices)

  V = { a, b, c, d, e }

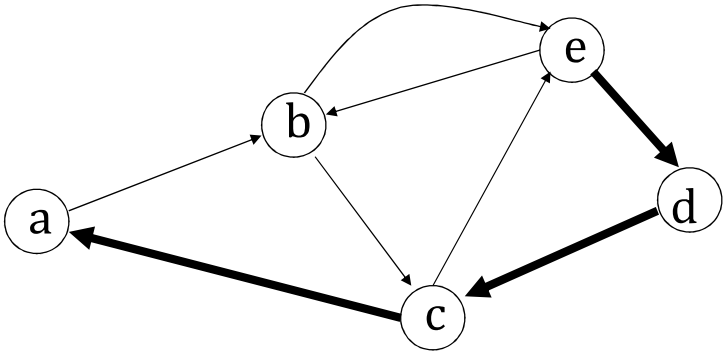
- Edges

  E = { (a,b), (b,c), (b,e),(c,a), (c,e), (d,c), (e,b), (e,d) }

Labeled Graph

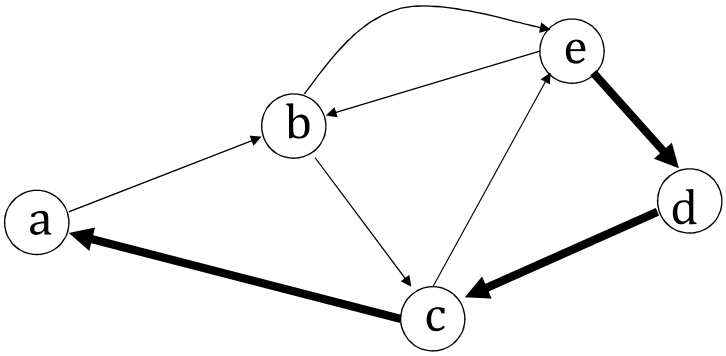


Walk



Walk is a sequence of adjacent edges
(e, d), (d, c), (c, a)

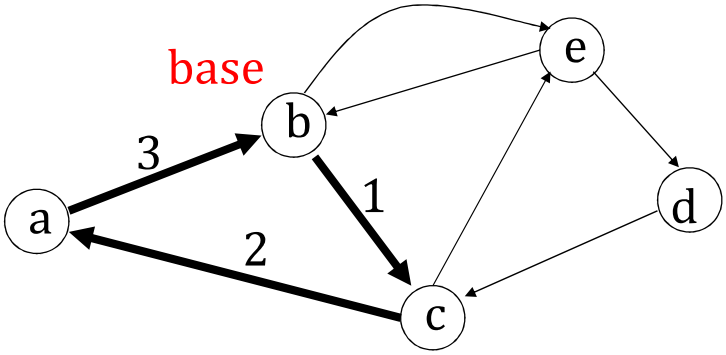
Path



Path is a walk where no edge is repeated

Simple path: no node is repeated

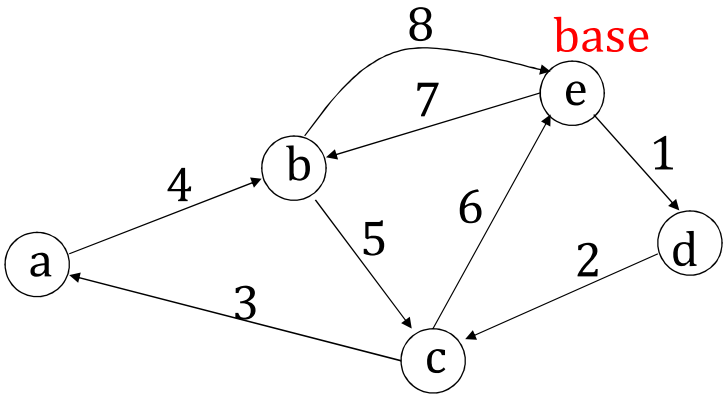
Cycle



Cycle: a walk from a node (base) to itself

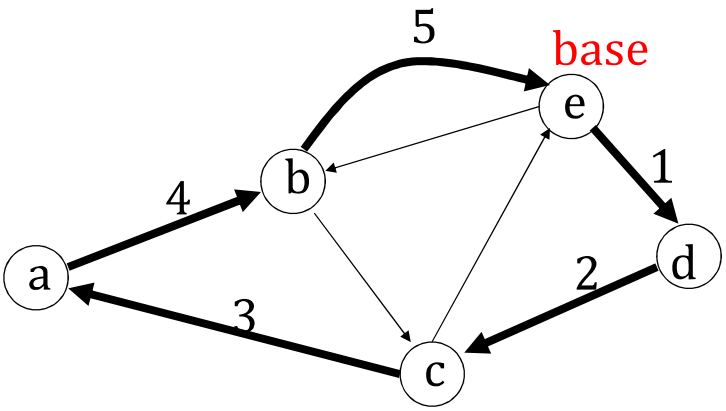
Simple cycle: only the base node is repeated

Euler Tour



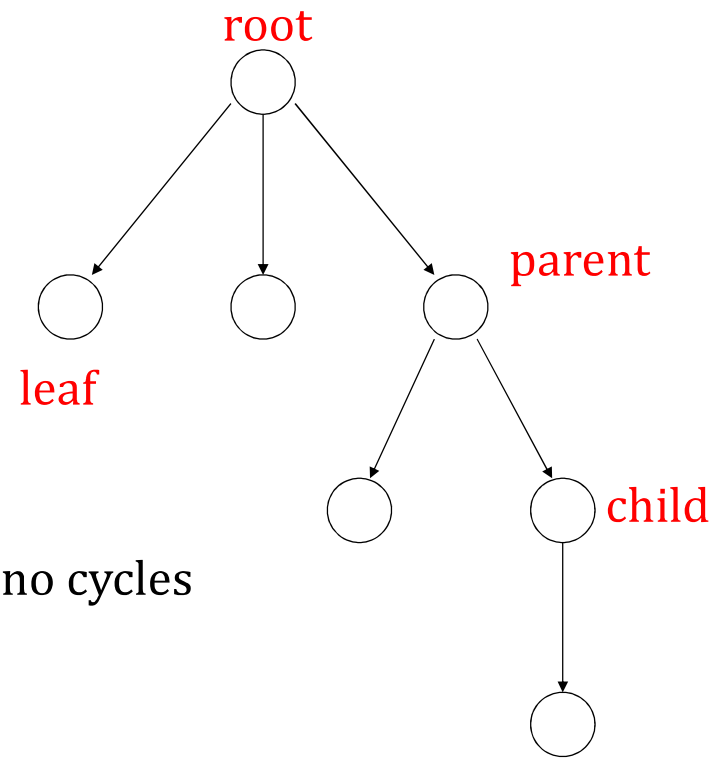
A cycle that contains each edge once

Hamiltonian Cycle

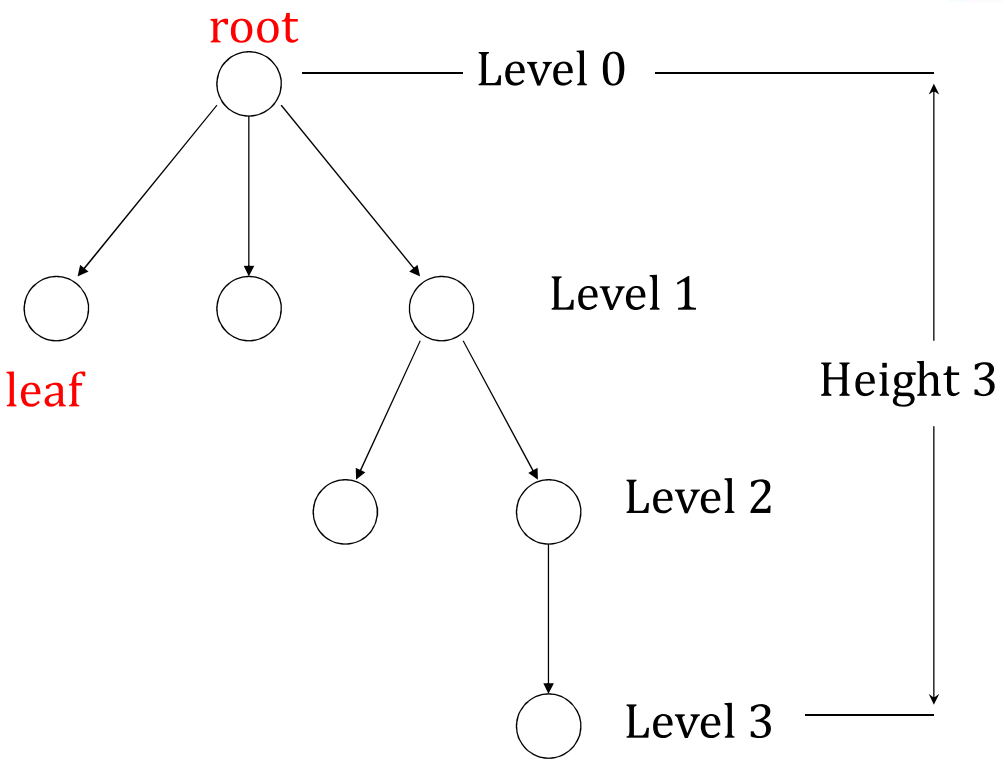


A simple cycle that contains all nodes

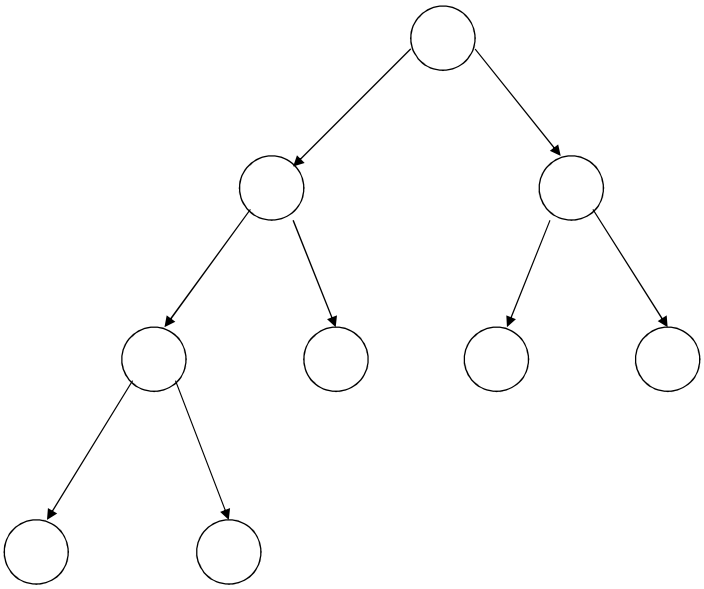
Trees



Trees have no cycles



Binary Trees



PROOF TECHNIQUES

- Proof by induction
- Proof by contradiction

Induction

We have statements P_1, P_2, P_3, …

If we know

- for some b that P_1, P_2, …, P_b are true
- for any k >= b that

  P_1, P_2, …, P_k imply P_{k+1}

Then

Every P_i is true

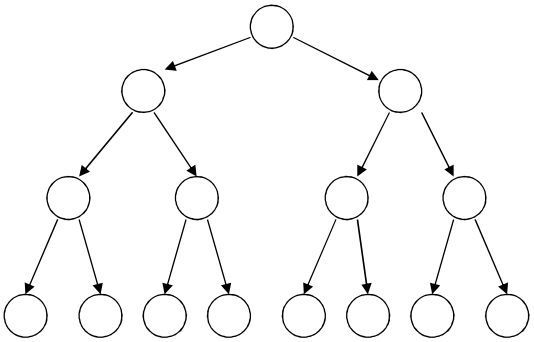
Proof by Induction

- Inductive basis
  Find P_1, P_2, …, P_b which are true
- Inductive hypothesis
  Let's assume P_1, P_2, …, P_k are true,
  for any k >= b
- Inductive step
  Show that P_{k+1} is true

Example

Theorem: A binary tree of height n
has at most 2^n leaves.

Proof by induction:
let L(i) be the number of leaves at level i



L(0) = 1

L(1) = 2

L(2) = 4

L(3) = 8

We want to show: $L(i) <= 2^i$

- Inductive basis

  $L(0) = 1$ (the root node)

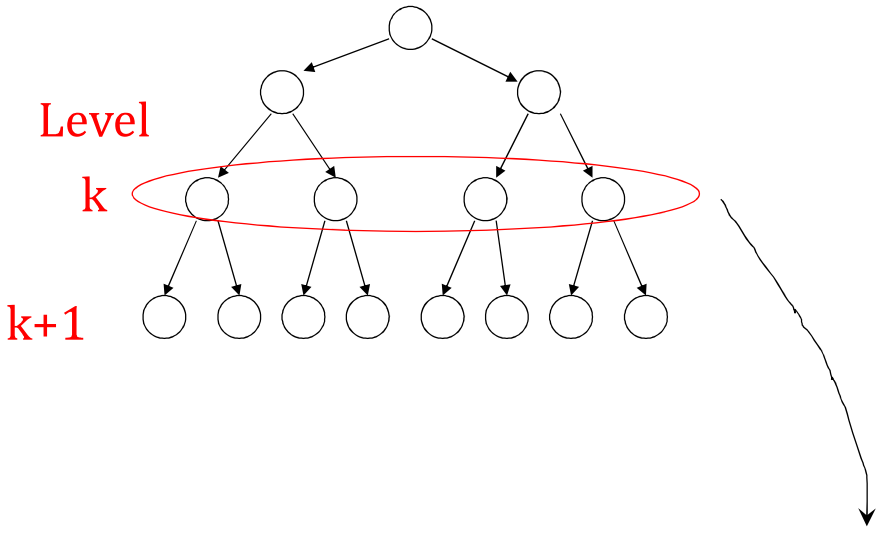
- Inductive hypothesis

  Let's assume $L(i) <= 2^i$ for all $i = 0, 1, \ldots, k$

- Induction step

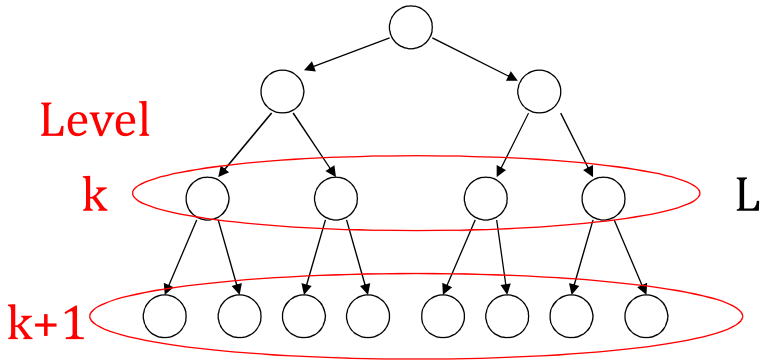
  we need to show that $L(k + 1) <= 2^{k+1}$

Induction Step



From Inductive hypothesis: $L(k) <= 2^k$

Induction Step



$L(k) <= 2^k$

$$L(k+1) <= 2 * L(k) <= 2 * 2^k = 2^{k+1}$$

Remark

Recursion is another thing

Example of recursive function:

$$f(n) = f(n-1) + f(n-2)$$

$$f(0) = 1, \quad f(1) = 1$$

Proof by Contradiction

We want to prove that a statement P is true

- we assume that P is false
- then we arrive at an incorrect conclusion
- therefore, statement P must be true

Example

Theorem: $\sqrt{2}$ is not rational

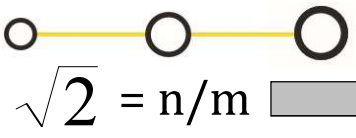
Proof:

Assume by contradiction that it is rational

$$\sqrt{2} = n/m$$

n and m have no common factors

We will show that this is impossible



$\sqrt{2} = n/m \Rightarrow 2\,m^2 = n^2$

Therefore, n^2 is even $\Rightarrow$ n is even, $n = 2k$

$2\,m^2 = 4k^2 \Rightarrow m^2 = 2k^2 \Rightarrow$ m is even, $m = 2p$

Thus, m and n have common factor 2

Contradiction!

Program Documentation

- Programs very hard to understand!
- Typical documentation in English

- Typical documentation

```
// Sets i to 5
i = 5
```

Formal Documentation

- Ways to make documentation precise.
- More like programs.
- Why? To make you think more formally about what program is doing.

Program State

- All of the information that would be necessary to restart at same place
  - Program counter
  - Values of variables
  - Also something about I/O…
    - (we will typically ignore the state of I/O)

Assertions

- An *assertion* is a statement that something is true.
  - Precondition – an assertion about state before a program (or chunk of code).
  - Postcondition – assertion after code

Silly Example


```
// Precondition: x == 6
x++;
// Postcondition: x == 7
```

- The behavior of the program is described.
- If x == 6 at the beginning,
- then x == 7 at the end.

Pre and Post conditions

- Says nothing about implementation!
  - Could be

```
// Precondition: x == 6
x = 7;
// Postcondition: x == 7
```

- This description is called the *specification*
- Also says nothing about termination.

Propositional Calculus

- A language of Boolean expressions
- Values are **true** or **false**
- We've seen this before

```
if( x > 10 && y < 5 )
```

Propositions

- Can be constructed using Boolean operators
  - **`&&, ||, ==, !, !=, xor`** (**`^`** in Java and C#)
  - In Python…
    - and, or, ==, not, !=
  - Example
  - **`(p && q) || !q`**

Boolean Operators

p	*q*	*! p*	*p && q*	*p || q*	*p xor q*	*p == q*
false	***false***	*true*	*false*	*false*	*false*	*true*
false	***true***	*true*	*false*	*true*	*true*	*false*
true	***false***	*false*	*false*	*true*	*true*	*false*
true	***true***	*false*	*true*	*true*	*false*	*true*

One and Zero as True and False

p	*q*	*! p*	*p && q*	*p || q*	*p xor q*	*p == q*
0	***0***	*1*	*0*	*0*	*0*	*1*
0	***1***	*1*	*0*	*1*	*1*	*0*
1	***0***	*0*	*0*	*1*	*1*	*0*
1	***1***	*0*	*1*	*1*	*0*	*1*

Implication

- Less common Boolean operator **=>**

p	*q*	*p => q*
0	0	1
0	1	1
1	0	0
1	1	1

- Read as *p implies q*
- q is true whenever p is true

Combination of Arithmetic and Boolean Expressions

Just like Java

```
((x + y) < 7) && (y > 9)
```

Arithmetic

Comparison

Boolean

Weak vs. Strong Assertions

- **`A => B`**
  - A is said to be stronger than B (or B weaker than A)
- Example

```
((x > 3) && (x < 7)) =>
        ((x > 0) && (x < 10))
```

Other Examples

`(p && r ) => p`

If p == true,
true => true
false => true
If p == false,
false => false

`p => (p || r)`

If p == true,
true => true
If p == false,
false => false
false => true

p	*q*	*p* => *q*
0	0	1
0	1	1
1	0	0
1	1	1

Other Examples

- Is this true?

  `p => true`

- How about this?

  `false => q`

p	*q*	*p* => *q*
0	0	1
0	1	1
1	0	0
1	1	1

No Implication Operator in Python, Java, C# and rest?

- No problem
- You can make one
- How?

p	*q*	*p* => *q*
0	**0**	1
0	**1**	1
1	**0**	0
1	**1**	1

!p || q

Quantifiers

- Still missing ability to say something about sets of variables
  - How could we test that list is sorted?
- Quantifiers allow you to say
  - *All of the entries of array B are > 0*
  - *There is an element of B that is zero*

Universal Quantifier

- For all integers *x*, *x > 3*
- We can write this as

  $\forall x \quad [x > 3]$
- Or, if you don't have cool symbols

  Ax [x > 3]
- Read as
  - For all x, x is greater than 3.
  - Generally false, of course.

Existential Quantifier

- There exists an integer x, such that x > 3

  $\exists x \quad [x > 3]$
- or

  Ex [x > 3]

Examples

- Informal:
  - The value 4 occurs in the array `B`
- Formal:
  - There exists a value of *i* between `0` and `n-1`, inclusive, such that `B[i] == 4`.
- Notation

```
(Ei: 0 <= i < n : B[i] == 4)
```

Examples

- Informal: The first element of the array `B` is the largest.
- Could mean:
  - The value of `B[0]` is at least as large as every entry of `B`.

```
(Ai: 0 < i < n: B[i] <= B[0])
```

- `or`
  - The value of `B[0]` is strictly larger than every other entry of `B`.

```
(Ai: 0 < i < n: B[i] < B[0])
```

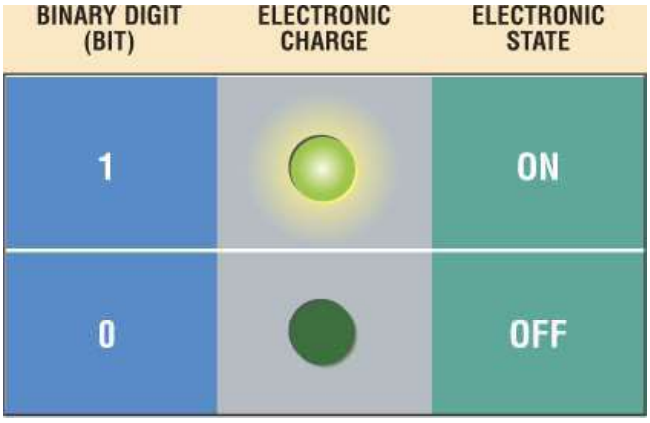
Examples

- Informal: The array `B` is sorted in non-decreasing order.
  - If i is less than j, then `B[i]` is less than or equal to `B[j]`

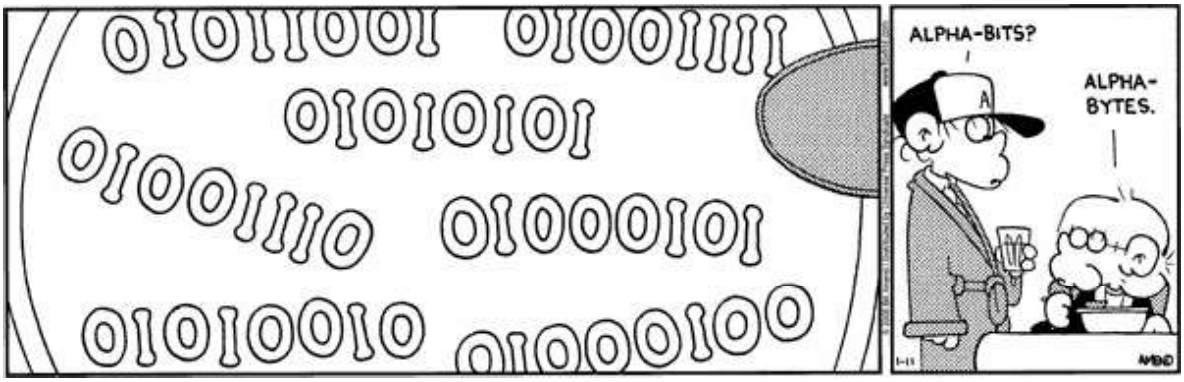
```
(Ai Aj : 0 <= i < j < n: B[i] <= B[j])
```

Data Representation

- How do computers represent data?
  - **Most computers are digital**



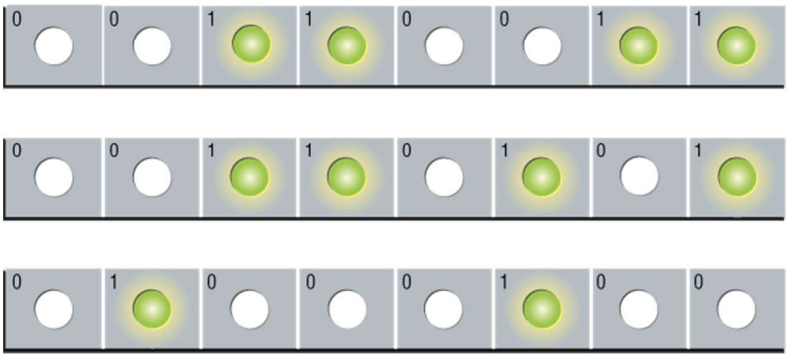
- Recognize only two discrete states: on or off
- Use a **binary system** to recognize two states
- Use number system with two unique digits: 0 and 1, called **bits** (short for binary digits)
  - Smallest unit of data computer can process



Data Representation

- What is a byte?

- **Eight bits grouped together as a unit**
- **Provides enough different combinations of 0s and 1s to represent 256 individual characters**
  - Numbers
  - Uppercase and lowercase letters
  - Punctuation marks



Converting Binary to Decimal

- Decimal number system is base 10
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Uses 10 numbers

Power of 10 representation	23,625 10^4	10^3	10^2	10^1	10^0
Decimal representation	10000	1000	100	10	1
Base 10 representation	20,000	3,000	600	20	5

Converting Binary to Decimal

Binary number system is base 2

- **0, 1**
- **Uses 2 numbers**

10010001 = 145

Base 2 representation	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal representation	128	64	32	16	8	4	2	1
Base 2 representation	1	0	0	1	0	0	0	1

Converting Decimal to Binary

- Convert decimal 35 to binary
  1. Using 8 bits, find largest power of 2 that will "fit" into 35
  2. Place a 1 into that slot
  3. If the # doesn't fit, place a 0 into that slot

Power of 2 representation	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal representation	128	64	32	16	8	4	2	1
Base 2 representation	0	0	1	0	0	0	1	1

35 = 00100011

Convert Binary to Decimal

1. Choose an 8 bit binary number = 10101110
2. Write the binary digits under the correct column
3. For each column with a 1, you will add that decimal value
4. You will not add the values of the columns you entered 0

Power of 2 representation	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal representation	128	64	32	16	8	4	2	1
Base 2 representation	1	0	1	0	1	1	1	0

128 + 32 + 8 + 4 + 2 = 174

10101110 = 174

Data Representation

- What are three popular coding systems to represent data?
  - **ASCII—American Standard Code for Information Interchange**
  - **EBCDIC—Extended Binary Coded Decimal Interchange Code**
  - **Unicode—coding scheme capable of representing all world's languages**

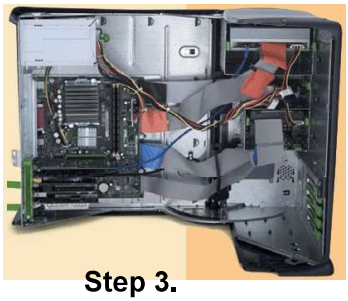
ASCII	Symbol	EBCDIC
00110000	0	11110000
00110001	1	11110001
00110010	2	11110010
00110011	3	11110011

Data Representation

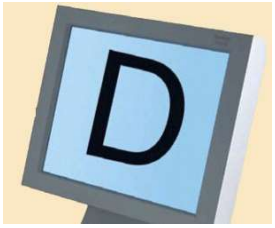
- How is a letter converted to binary form and back?

Step 1.
The user presses the capital letter **D** (shift+D key) on the keyboard.

Step 2.
An electronic signal for the capital letter **D** is sent to the system unit.



Step 3.
The signal for the capital letter **D** is converted to its ASCII binary code (01000100) and is stored in memory for processing.



Step 4.
After processing, the binary code for the capital letter **D** is converted to an image, and displayed on the output device.

Memory

- What is memory?
  - **Electronic components that store instructions, data, and results**
  - **Consists of one or more chips on motherboard or other circuit board**
  - **Each byte stored in unique location called an address, similar to addresses on a passenger train**

Memory

- Stores three basic categories of items:
  1. OS and system software
  2. Application programs
  3. Data and information
- Byte is basic storage unit in memory
- To access data or instructions in memory, computer references the address that contain the bytes of data
- Manufacturers state the size of memory and storage devices in terms of number of bytes available

Memory

- How is memory measured?
  - **By number of bytes available for storage**
  - **KB = 1024 bytes**

Term	Abbreviation	Approximate Size
Kilobyte	KB or K	1 thousand bytes
Megabyte	MB	1 million bytes
Gigabyte	GB	1 billion bytes
Terabyte	TB	1 trillion bytes

Name	Abbr.	Size
Kilo	K	2^10 = 1,024
Mega	M	2^20 = 1,048,576
Giga	G	2^30 = 1,073,741,824
Tera	T	2^40 = 1,099,511,627,776
Peta	P	2^50 = 1,125,899,906,842,624
Exa	E	2^60 = 1,152,921,504,606,846,976
Zetta	Z	2^70 = 1,180,591,620,717,411,303,424
Yotta	Y	2^80 = 1,208,925,819,614,629,174,706,176

MEMORY AND STORAGE SIZES

Term	Abbreviation	Approximate Size	Exact Amount	Approximate Number of Pages of Text
Kilobyte	KB or K	1 thousand bytes	1,024 bytes	1/2
Megabyte	MB	1 million bytes	1,048,576 bytes	500
Gigabyte	GB	1 billion bytes	1,073,741,824 bytes	500,000
Terabyte	TB	1 trillion bytes	1,099,511,627,776 bytes	500,000,000

Regular Expression

- A pattern of special characters used to match strings in a search
- Typically made up from special characters called metacharacters
- Regular expressions are used across platforms:
  - UNIX Commands: grep, ed
  - C#, Java
  - Powershell
  - XML Schemas

Metacharacters

RE Metacharacter	Matches…
.	**Any one character, except new line**
[a-z]	**Any one of the enclosed characters (e.g. a-z)**
*	**Zero or more of preceding character**
? or \?	**Zero or one of the preceding characters**
+ or \+	**One or more of the preceding characters**

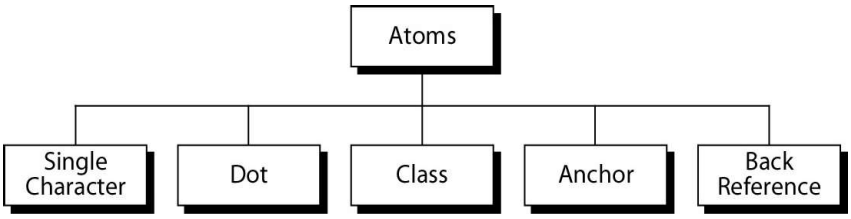
- any non-metacharacter matches itself

more Metacharacters

RE Metacharacter	Matches…
^	**beginning of line**
$	**end of line**
char	**Escape the meaning of *char* following it**
[^]	**One character not in the set**
\<	**Beginning of word anchor**
\>	**End of word anchor**
( ) or \(\)	**Tags matched characters to be used later (max = 9)**
| or |	**Or grouping**
x\{m\}	**Repetition of character x, m times (x,m = integer)**
x\{m,\}	**Repetition of character x, at least m times**
x\{m,n\}	**Repetition of character x between m and m times**

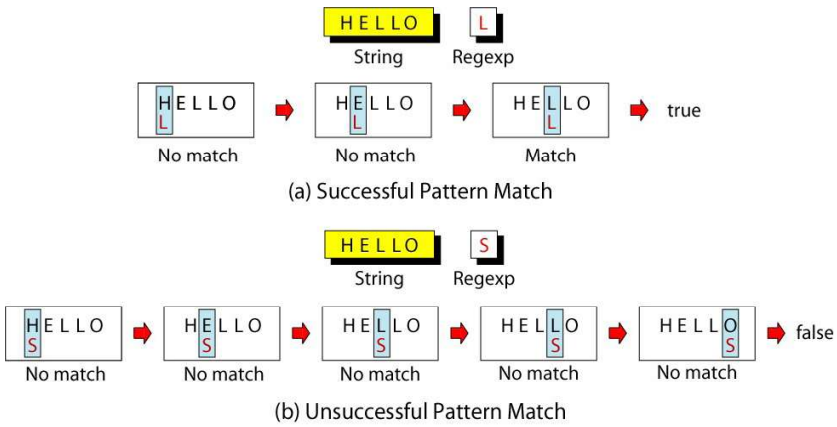
Atoms

An atom specifies what text is to be matched and where it is to be found.



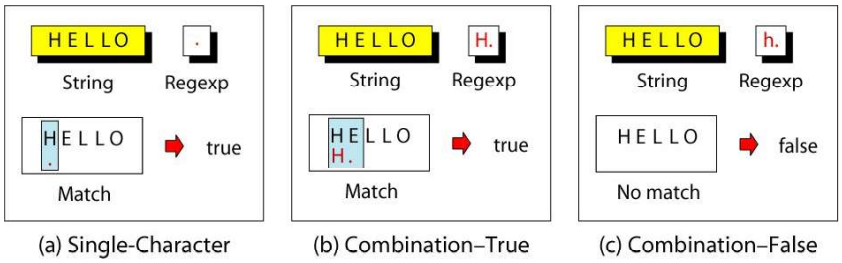
Single Character Atom

A single character matches itself



Dot Atom

matches any single character except for a new line character (\n)

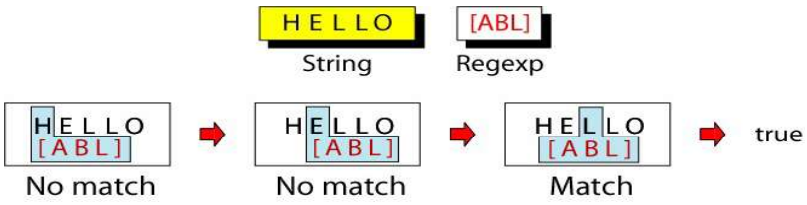


(a) Single-Character (b) Combination–True (c) Combination–False

Class Atom

matches only single character that can be any of the characters defined in a set:

Example: [ABC] matches either A, B, or C.



Notes:

1) A range of characters is indicated by a dash, e.g. [A-Q]
2) Can specify characters to be excluded from the set, e.g. [^0-9] matches any character other than a number.

Examples

RegExpr	Means	RegExpr	Means
[A-H]	[ABCDEFGH]	[^AB]	Any character except A or B
[A-Z]	Any uppercase alphabetic	[A-Za-z]	Any alphabetic
[0-9]	Any digit	[^0-9]	Any character except a digit
[[a]	[or a	[]a]	] or a
[0-9\-]	digit or hyphen	[^\^]	Anything except^

Anchors

Anchors tell where the next character in the pattern must be located in the text data.

Anchor	Means	Example
^	Beginning of line	One line of text.\n
$	End of line	One line of text.\n
\<	Beginning of word	One line of text.\n
\>	End of word	One line of text.\n

Navigate to…

http://bit.ly/2kvxDxa

Overview

- Learning Objectives
- Flow and Structure
- Tools and material

Learning Objectives

At the end of the program, participants will have:

- Acquired the appropriate background to code and extend their programming skills
- Developed a project portfolio containing at least three full-stack applications and various exercises
- Learnt to use the latest tools and concepts in application development
- Worked in a team to design and develop a larger-scale application

Outcome

Building on the acquired knowledge, participants will be able to:

- Develop full-stack applications using the C# or Java programming languages
- Start working as full-stack entry-level application developers

In other words...

- Learn how to **guess**
- Learn how to **search**

Program Structure

- Basic Coding Skills (Foundation Level)
  - 7 weeks – Foundation Level Certificate
- Additional Coding Skills (Advanced Level)
  - 5 weeks – Practitioner Level Certificate

Technologies

Java	C# .net	In Common
• Java • Maven/ Tomcat • MySQL / HSQLDB • Servlets / JSP • Spring MVC / DI / AOP • JPA / Hibernate	• C# • ASP .NET MVC • SQL Server • SQL Reporting Services • ADO .NET/ Entity Framework • Visual Studio • Dapper	• REST architecture • Web servers (Apache, nginx) • xUnit • Cloud-based services and deployment (AWS, Google Cloud Platform) • JavaScript • AngularJS / React • Python • MongoDB and other NoSQL databases

Curriculum & Syllabus (Part I)

- Curriculum (Educational Material)

Topic
Introduction to Software Design and Development
Introduction to Object Oriented Programming
Object Oriented Programming
Relational Databases
Web Design and Development Fundamentals (Front End)
Web Application Development
Web Application Development, MVC and other Frameworks (Basics)
Individual Project (Windows application)

- Syllabus (Exam Material)
  - Part I – PeopleCert Developer Skills, Foundation Level

Exam in Java or C#, depending on your selected stream

Curriculum & Syllabus (Part II)

- Curriculum (Educational Material)

Topic
Web Application Development, MVC and other Frameworks (Advanced)
Web Design and Development Advanced Topics
Object Oriented Programming – Advanced Topics
Advanced Application Development and Other Topics
Testing & Debugging - Advanced
Maintainable Code
Developer Soft Skills, Team Work and Project Documentation
Team/Group Project (Web application)

- Syllabus (Exam Material)
  - Part II – PeopleCert Developer Skills, Practitioner Level

Exam in Java or C#, depending on your selected stream

Software Design and Development

(Syllabus 1.1 – 1.4)

Knowledge/Skill Set	Ref	Task Item	Lecture Hour
1.1 Key Concepts	1.1.1	Define the term computing	09:00-10:30
	1.1.2	Define the terms code and program/application	
	1.1.3	Define the terms programming language, compiler and software execution	
	1.1.4	Recognise typical activities in the creation of a program: analysis, design, programming, debugging/testing, maintaining/enhancement	
	1.1.5	Define the term "full-stack development"	
1.2 Software Architectures	1.2.1	Understand the notion of software architecture	
	1.2.2	Understand the notion of the client-server model	
	1.2.3	Understand typical web-application architecture	
	1.2.4	Understand three-tier architecture	
	1.2.5	Understand the end-to-end workflow of a web request	
	1.2.6	Understand the software-as-a-service concept	
	1.2.7	Distinguish between centralized versus distributed application architectures: peer-to-peer, microservices etc.	
	1.2.8	Understand the architecture of popular applications	
	1.2.9	Understand mobile application architectures	
1.3 Software Development Methodologies	1.3.1	Understand the application development process	
	1.3.2	Describe the main software development methodologies, like: Waterfall, prototyping, rapid application, agile code and fix. Distinguish between these methodologies and when they are applied.	
	1.3.3	List the main software development related activities as per software methodology: Requirements, Design, Construction, Testing, Debugging, Deployment, Maintenance	
	1.3.4	Understand how code reviews work	
	1.3.5	Understand how DevOps works	
	1.3.6	Understand how Scrum works	
1.4 Requirements Capturing and Software Design	1.4.1	Understand the process of requirements capturing and know how to complete it	10:45-12:15
	1.4.2	Understand the process from requirements capturing to software design and apply it to a simple case	
	1.4.3	Create a mock-up design	
	1.4.4	Use Unified Modelling Language (UML) to create a Use-case diagram	
	1.4.5	Use Unified Modelling Language (UML) to create a Class diagram	
	1.4.6	Use Unified Modelling Language (UML) to create a Sequence diagram, State transition diagram	
	1.4.7	Use a GUI Designer and its basic functions	

Alliance For Digital Employability

1.1 Key Concepts

Key Concepts

- Computing
- Code
- Program/Application
- Programming language
- Compiler
- Software execution

Basic Concepts & Terms

- Software
- Computer program
- Algorithm
- System software vs. Application software
- Compiler and Interpreter
- Software architecture
- Full-stack application development

Software

- Computer software, or simply software, is that part of a computer system that consists of encoded information or computer instructions
- Computer hardware and software require each other and neither can be realistically used on its own
- A collection of computer programs, libraries and related data are referred to as software

Computer program

- A computer program is a collection of instructions that performs a specific task when executed by a computer
- A computer executes the program's instructions in a central processing unit
- A computer program is usually written by a computer programmer in a programming language
- From the program in its human-readable form of source code, a compiler can derive machine code
- Alternatively, a computer program may be executed with the aid of an interpreter

Algorithm

- A part of a computer program that performs a well-defined task is known as an algorithm
- An algorithm is a self-contained step-by-step set of operations to be performed
- Algorithms perform calculation, data processing, and/or automated reasoning tasks

Class Exercise

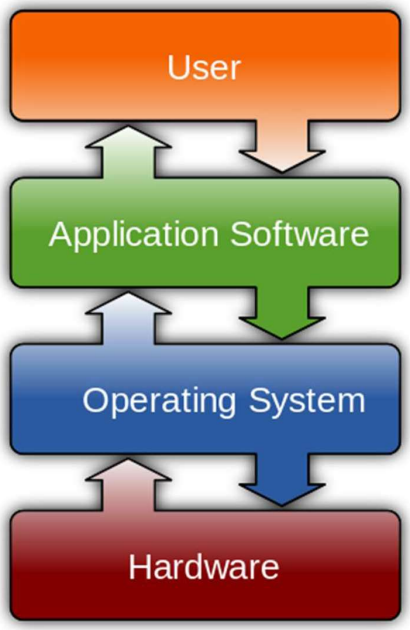
Write an **algorithm** for the following problem:

Problem: Given a list of positive numbers, return the largest number on the list.

Inputs: A list L of positive numbers. This list must contain at least one number. (Asking for the largest number in a list of no numbers is not a meaningful question.)

Outputs: A number n, which will be the largest number of the list.

Software, system software and hardware



Application Software

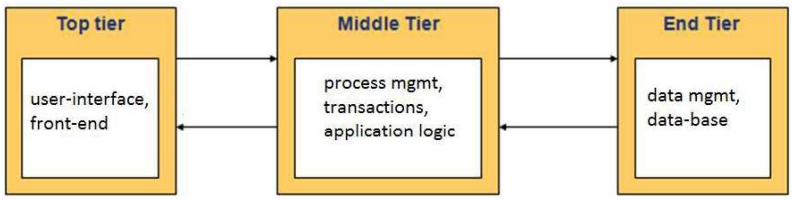
- App or application for short
- A computer program designed to perform a group of coordinated functions, tasks, or activities for the benefit of the user
- Examples: a word processor, a spreadsheet, an accounting application, a web browser etc.
- The collective noun "application software" refers to all applications collectively (contrasts with system software)
- Apps built for mobile platforms are called mobile apps

Software Architecture

- Software architecture refers to the fundamental structures of a software system
- Each structure comprises software elements, relations among them, and properties of both elements and relations
- The architecture of a software system is a metaphor, analogous to the architecture of a building

Full Stack Development

- Refers to software development covering all the layers of a 3-tier architecture:



Software development process

Core Activities:

- Requirements capturing (also called *Requirements analysis*)
- Design
- Construction (also called *Software implementation*)
- Software Testing and Debugging
- Deployment
- Maintenance

Typical activities in the creation of a program

- Analysis
- Design
- Programming
- Debugging/testing
- Maintaining/enhancement

1.1 Key Concepts

Q&A

Exercises and Review Questions

Sample Question #1

Computing is defined as:

A. Any goal-oriented activity requiring, benefiting from, or creating computers

B. An algorithm that uniquely represents symbols from some source alphabet, which may be in some other target alphabet

C. A deliberate process that transforms one or more inputs into one or more results, with variable change

D. A process of discovering and resolving defects that prevent correct operation of computer software or a system

Sample Question #1 - Answer

Computing is defined as:

A. **Any goal-oriented activity requiring, benefiting from, or creating computers**
B. An algorithm that uniquely represents symbols from some source alphabet, which may be in some other target alphabet
C. A deliberate process that transforms one or more inputs into one or more results, with variable change
D. A process of discovering and resolving defects that prevent correct operation of computer software or a system

Sample Question #2

Which of the following is a typical activity in the creation of a program?

A. Prototyping
B. Project Management
C. Rapid Application
D. Debugging

Sample Question #2 - Answer

Which of the following is a typical activity in the creation of a program?

A. Prototyping

B. Project Management

C. Rapid Application

D. Debugging

Sample Question #3

The fundamental structures of a software system are referred to as:

A. Computing

B. Algorithm

C. Software Architecture

D. Application Software

Sample Question #3 - Answer

The fundamental structures of a software system are referred to as:

A. Computing

B. Algorithm

C. Software Architecture

D. Application Software

1.2 Software Architectures

Software Architecture

- The **high level structures** of a software system, the **discipline** of creating such structures, and the **documentation** of these structures.
- These structures are needed to reason about the software system.
- Each structure comprises
  - **software** elements,
  - **relations** among them, and
  - **properties** of both elements and relations
- The **architecture** of a software system is a metaphor, analogous to the architecture of a building

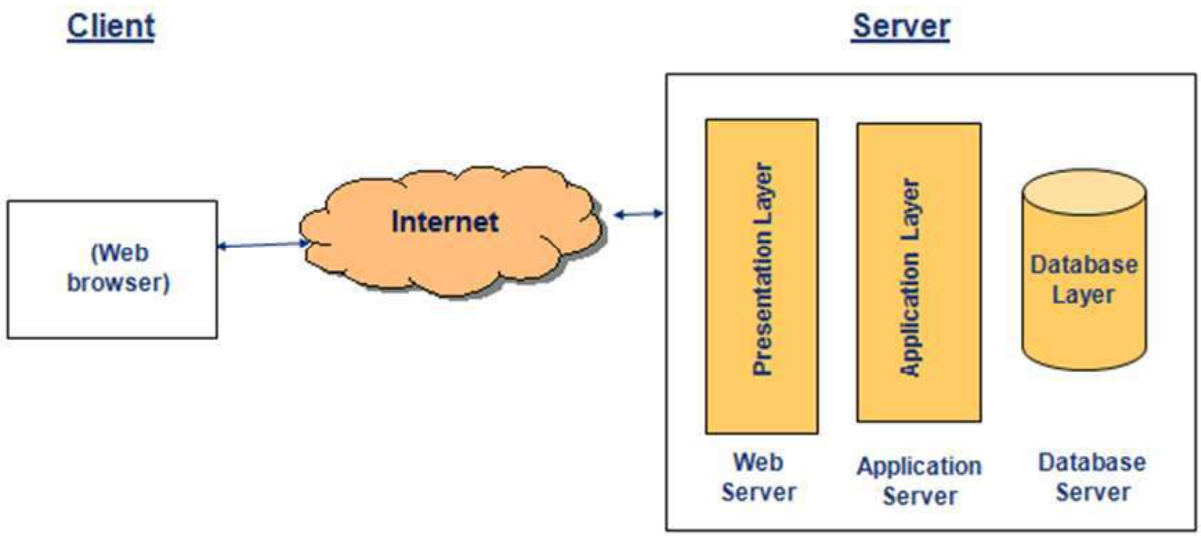
Software Architecture (2)

- Making fundamental **structural choices** which are **costly** to change once implemented ; specific structural options from possibilities in the design of software.
- **Documenting** software architecture facilitates communication between stakeholders, captures early decisions about the high-level design, and allows reuse of design components between projects

Software architecture (3)

- Architectural design
- Detailed Software design
- Client-server architecture
- 3-tier architecture
- Typical web application architecture

Architectural Design

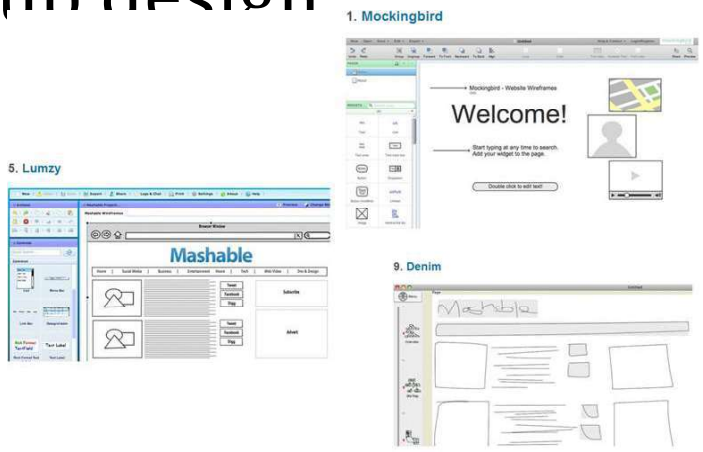


- Also called *High Level Architecture*
- High-level view of a system

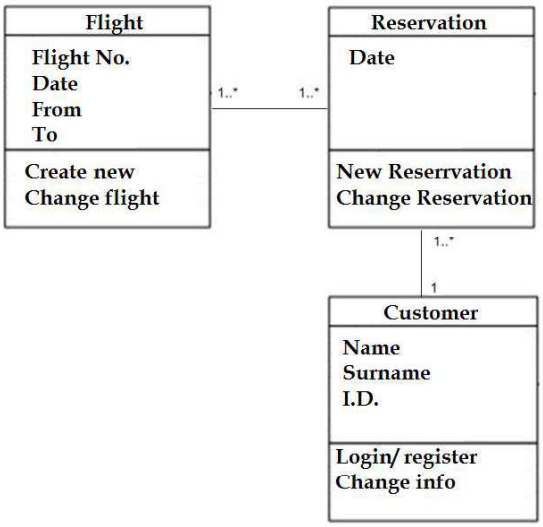
Detailed Software design

- Depicts relationships among software elements/ components at lower level
- E.g. UI design, class diagram, database schema, component diagram

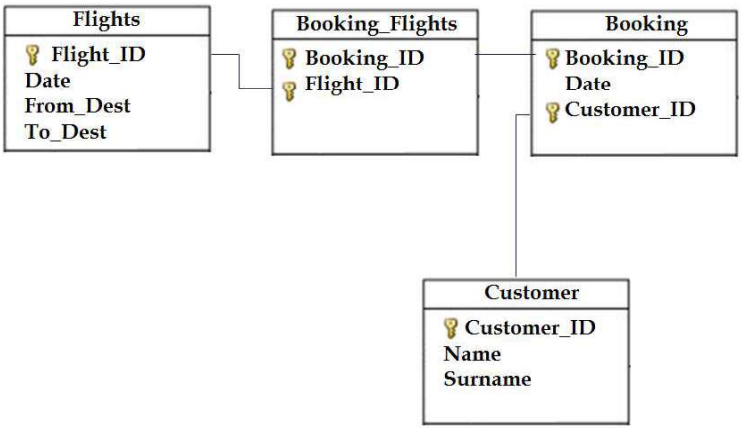
Example: User-interface/ mockup design



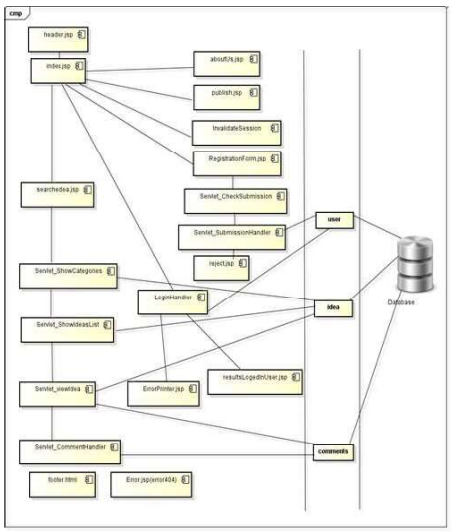
Example: Class Diagram



Example: Database schema

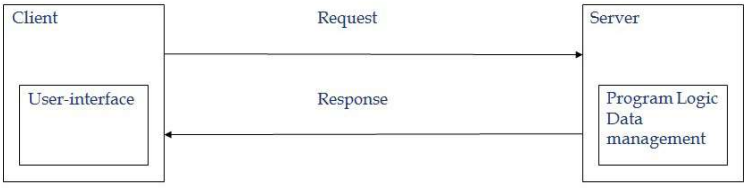


Example: Component diagram

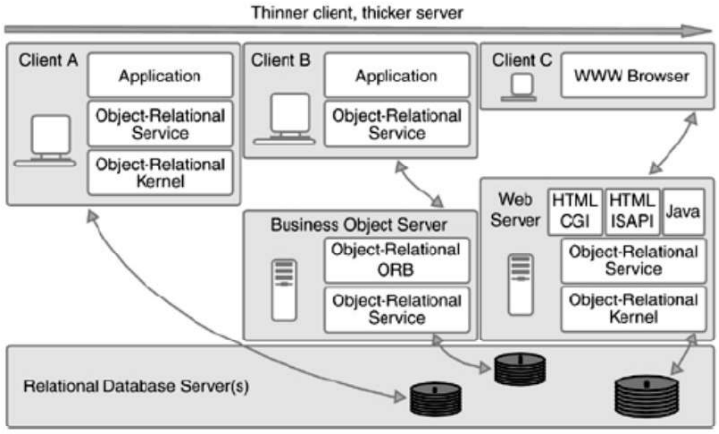


Client-Server architecture

- **Server (host):** A centralized, host computer
- **Clients** (remote processors) request and receive service from a centralized server (host computer).

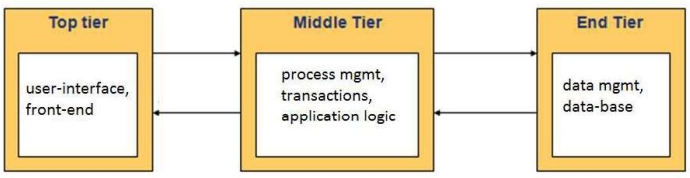


Thick-Thin Client

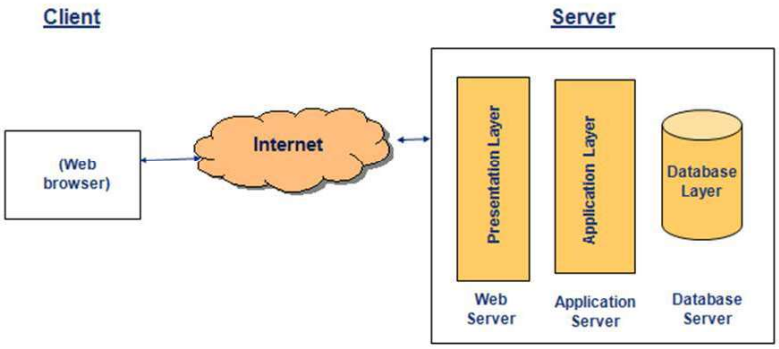


3-tier architecture

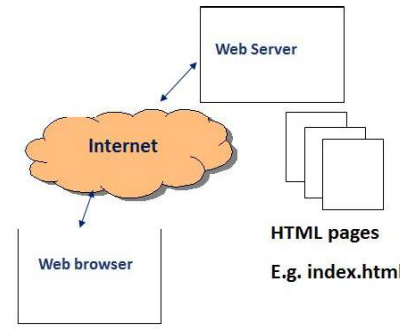
- **Client–server** software architecture pattern
- Developed by **John J. Donovan** in Open Environment Corporation (OEC), a tools company he founded in Cambridge, Massachusetts.
- Indented to allow any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology.
- **Presentation tier** (Top tier): it is a layer which users can access directly (such as a web page, or an operating system's GUI).
- **Application tier** (business logic, logic tier, or middle tier): Controls an application's functionality by performing detailed processing.
- **Data tier** (end tier): includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates
improved scal



Typical Web application architecture



Static Web site

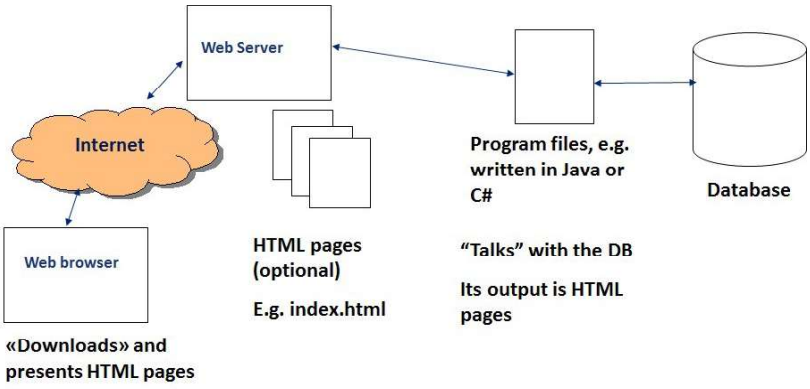


HTML pages

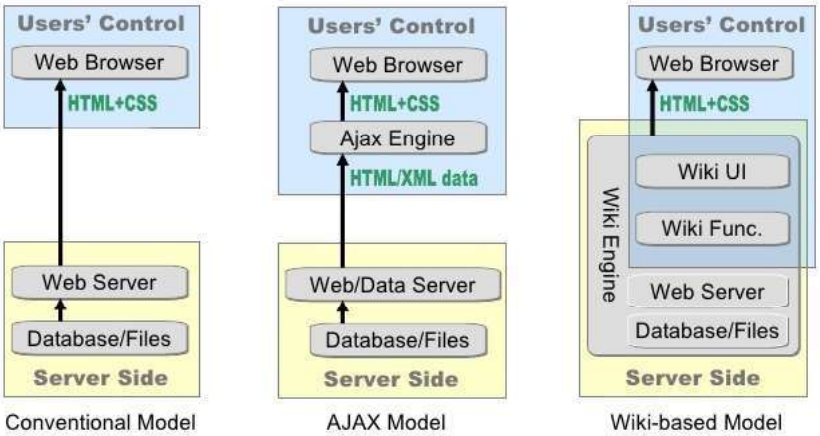
E.g. index.html + content html pages

«Downloads» and presents HTML pages

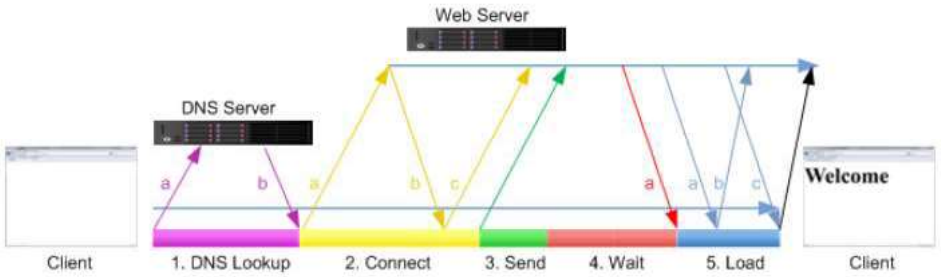
Dynamic Web site



Web application model



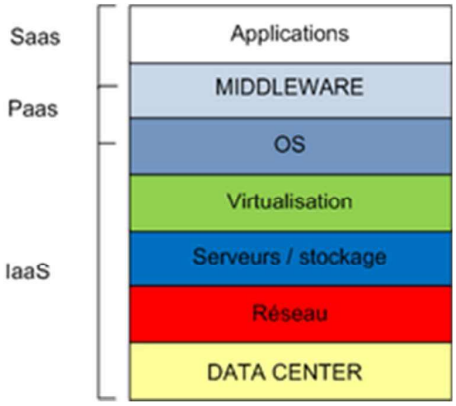
End-to-end workflow of a web request



DNS Lookup: The client tries to resolve the domain name for the request.
Client sends DNS Query to local ISP DNS server.
DNS server responds with the IP address for hostname.com
Connect: Client establishes TCP connection with the IP address of hostname.com
Client sends SYN packet.
Web server sends SYN-ACK packet.
Client answers with ACK packet, concluding the three-way TCP connection establishment.
Send: Client sends the HTTP request to the web server.
Wait: Client waits for the server to respond to the request.
Web server processes the request, finds the resource, and sends the response to the Client. Client receives the first byte of the first packet from the web server, which contains the HTTP Response headers and content.
Load: Client loads the content of the response.
Web server sends second TCP segment with the PSH flag set.
Client sends ACK. (Client sends ACK every two segments it receives. from the host)
Web server sends third TCP segment with HTTP_Continue.
Close: Client sends a a FIN packet to close the TCP connection.

Software-as-a-service (SaaS) concept

- The term "Software as a Service" (SaaS) is considered to be part of the nomenclature of cloud computing
- A software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted
- SaaS has become a common delivery model for many business applications, including office software, messaging software, DBMS software, CRM software and many more



Centralized vs Distributed application architectures

- Distinguish between
  - peer-to-peer,
  - microservices etc

A **peer-to-peer (P2P) network** in which interconnected nodes ("peers") share resources amongst each other without the use of a centralized administrative system

A network based on the client-server model, where individual clients request services and resources from centralized servers

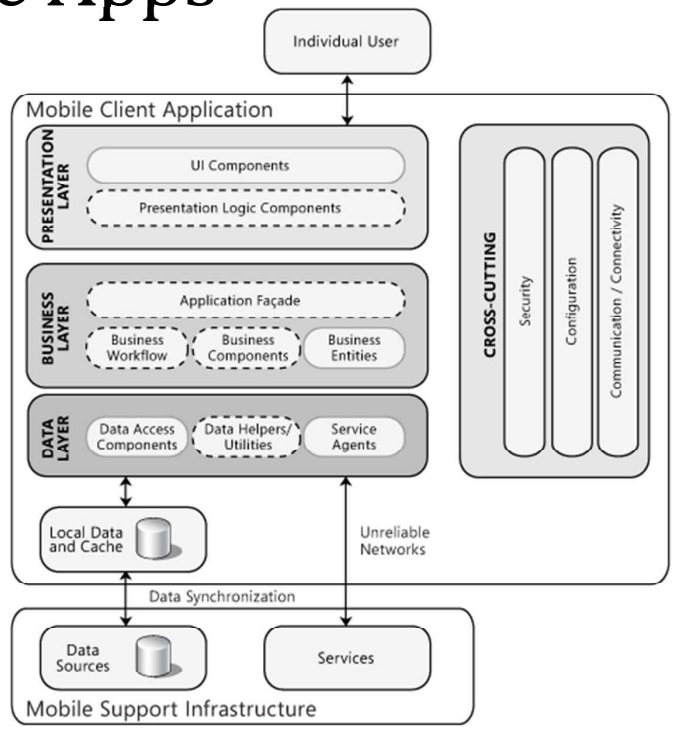
Microservices

- **Variant** of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.
- In a microservices architecture, services should be **fine-grained** and the protocols should be **lightweight.**
- The **benefit** of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop and test.
- Parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.
- Allows the architecture of an individual service to emerge through continuous refactoring.
- Microservices-based architectures enable continuous delivery and deployment

Mobile application architectures

- **Scalability –** A Mobile Architecture must be able to be utilized with all recovery requirements on both large and small scale.
- **Secure –** Encryption is important, transmission protocols must support encryption (SSL) via secure transit such as HTTPS
- **Reliable –** Reliability is always important in all technologies and mobile architecture is no different.

Mobile Apps



The architecture of popular applications

- Examples
  - Amazon is a web based application
  - Angry Birds on your phone is a mobile app
  - An ATM's application is client server
- Other examples

1.2 Software Architectures

Q&A

Exercises and Review Questions

Sample Question #1

The capability provided to the consumer to use a provider's applications running on a **cloud infrastructure** is also known as:

A. Platform as a Service (PaaS)

B. Software as a Service (SaaS)

C. Infrastructure as a Service (IaaS)

D. Communication as a Service (CaaS)

Sample Question #1 - Answer

The capability provided to the consumer to use a provider's applications running on a **cloud infrastructure** is also known as:

A. Platform as a Service (PaaS)

B. Software as a Service (SaaS)

C. Infrastructure as a Service (IaaS)

D. Communication as a Service (CaaS)

Sample Question #2

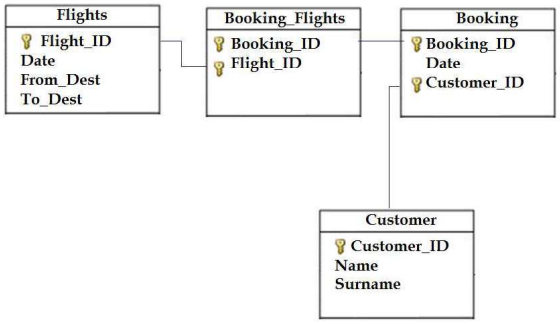
The image on the right shows a:

A. Class Diagram

B. Database Schema

C. Component Diagram

D. User Interface Mockup



Sample Question #2 - Answer

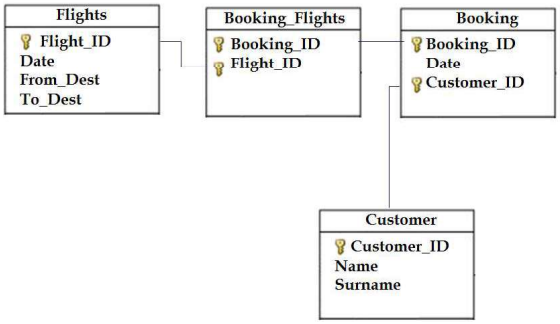
The image on the right shows a:

A. Class Diagram

B. Database Schema

C. Component Diagram

D. User Interface Mockup



Sample Question #3

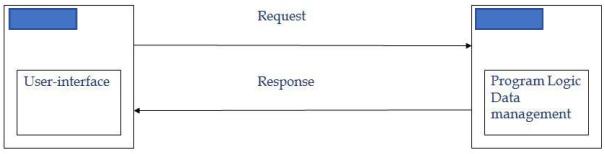
What kind of architecture
is shown on the image on the right?

A. Client-Server

B. Thick Client

C. 3-tier architecture

D. Web application



Sample Question #3 - Answer

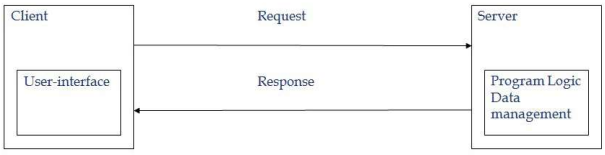
What kind of architecture
is shown on the image on the right?

A. Client-Server

B. Thick Client

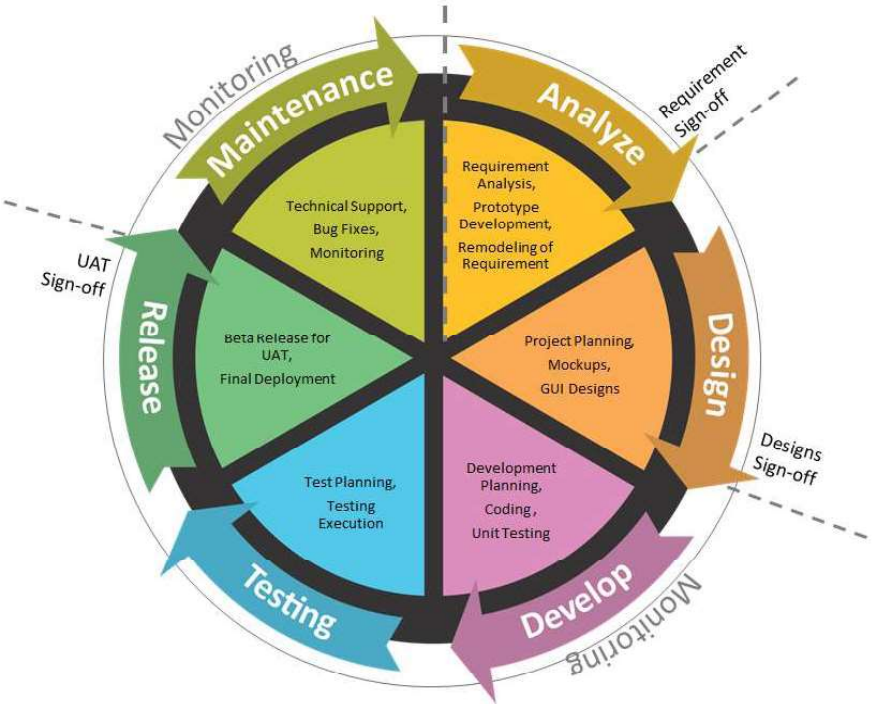
C. 3-tier architecture

D. Web application



1.3 Software Development Methodologies

Application development process



Software development methodologies

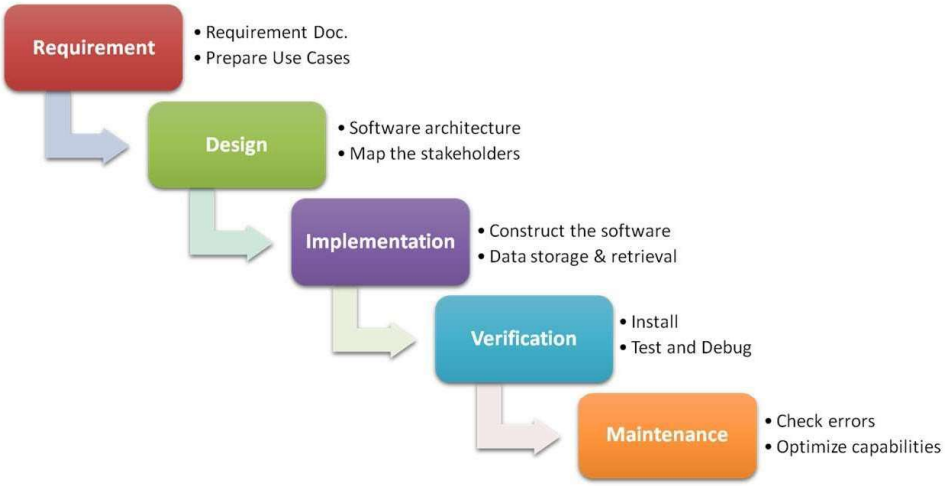
- Waterfall
- Prototyping
- Rapid application
- Agile code and fix.
- Distinguish between these methodologies and when they are applied

Methodologies, paradigms and models

- Software engineering
- Waterfall
- Prototyping
- Incremental
- V-Model, Dual Vee Model
- Spiral
- Iterative and incremental development (IID)
- Agile (2001)
- Lean (2003)
- DevOps (2008)

Waterfall

- Requirements, Design, Construction, Testing, Debugging, Deployment, Maintenance



Waterfall model phases

- Requirements analysis and definition
  - System's services, constraints and goals = System specification
- System and software design
  - Partitions the requirements to either SW or HW systems. System architecture
- Implementation and unit testing
  - The SW design is realised as a set of programs or program units
- Integration and system testing
- Operation and maintenance

- **The result of each phase is one or more documents which are approved ("signed off")**

Waterfall model problems

- Inflexible partitioning of the project into distinct stages
- This makes it difficult to respond to changing customer requirements
- Therefore, this model is only appropriate when the requirements are well-understood

- **The drawback of the waterfall model is the difficulty of accommodating change after the process is underway**

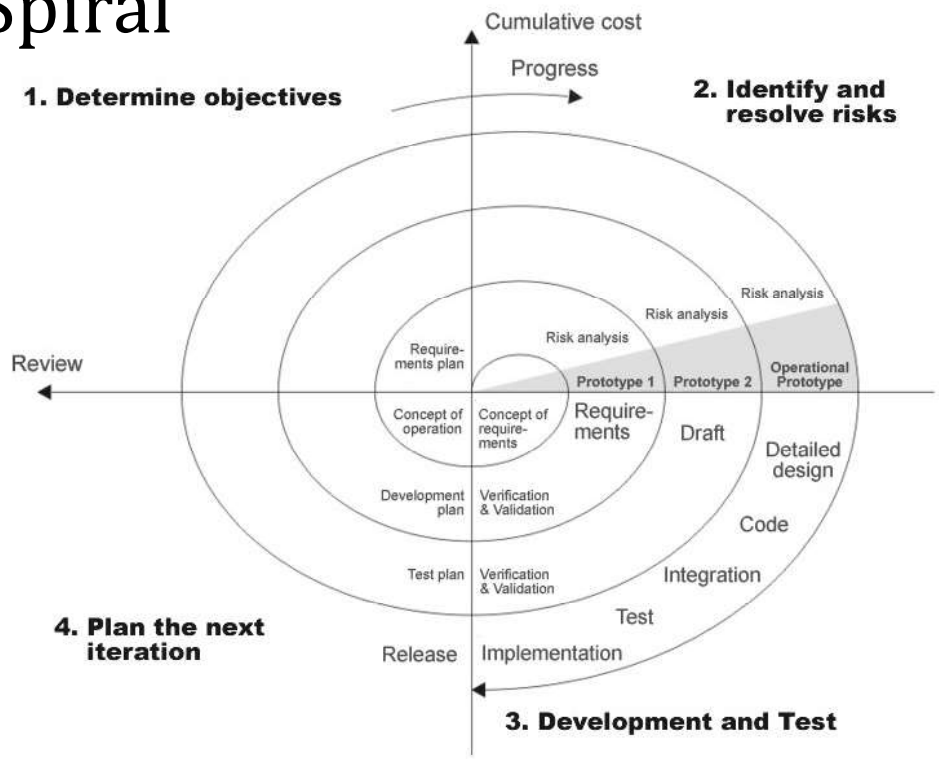
Prototyping

- Software prototyping is the activity of creating prototypes of software applications, i.e., incomplete versions of the software program being developed

Spiral development (Boehm, 1988)

- Process is represented as a spiral rather than as a sequence of activities with backtracking
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required
- Risks are explicitly assessed and resolved throughout the process

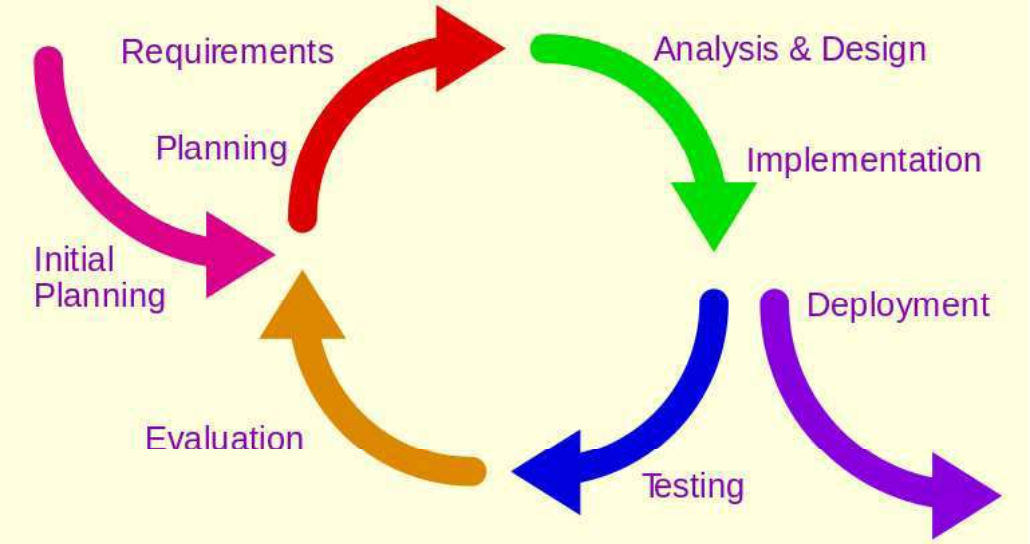
Spiral



Spiral model sectors

- Objective setting
  - Specific objectives for the phase are identified
- Risk assessment and reduction **(distinguishes it from other models)**
  - Risks are assessed and activities put in place to reduce the key risks
- Development and validation
  - A development model for the system is chosen which can be any of the generic models
- Planning
  - The project is reviewed and the next phase of the spiral is planned

Iterative and incremental development (IID)



Process iteration

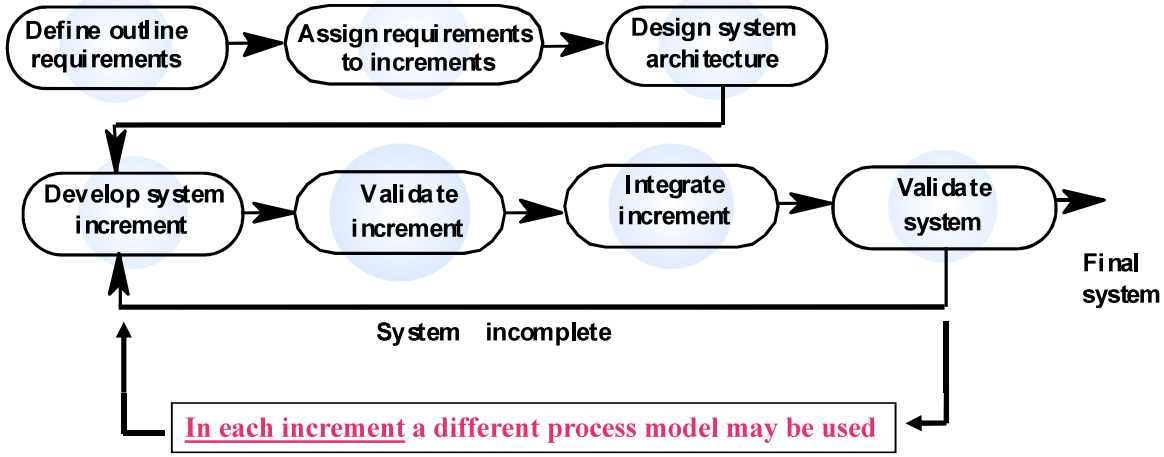
- System requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large systems
- Iteration can be applied to any of the generic process models
- Two (related) approaches
  - **Incremental development**
  - **Spiral development**

Incremental development

(Mills, 1980)

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality
- User requirements are prioritised and the highest priority requirements are included in early increments
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve

Incremental development



Incremental development advantages

- Customer value can be delivered with each increment so system functionality is available earlier
- Early increments act as a prototype to help elicit requirements for later increments
- Lower risk of overall project failure
- The highest priority system services tend to receive the most testing

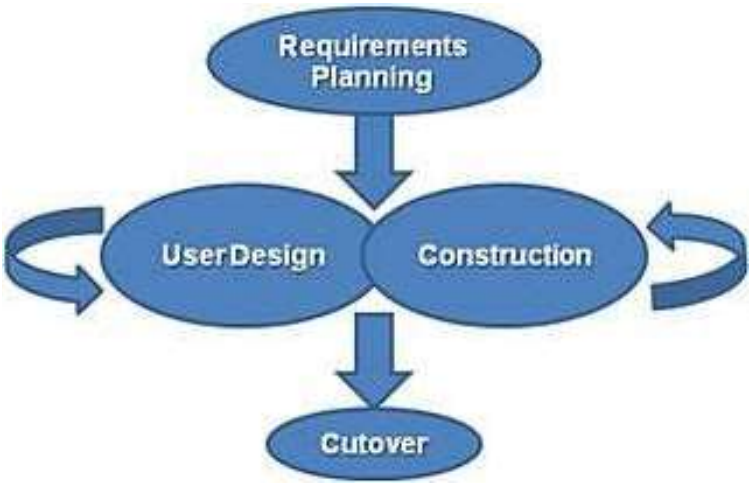
Incremental development problems

- Increments should be relatively small (< 20.000 LOC)
- Each increment should deliver some functionality
- Difficult to map customer's requirements onto increments of the right size

Rapid Application Development

- **Rapid-application development** (**RAD**) is both a:
  - general term used to refer to alternatives to the conventional waterfall model of software development
  - the name for James Martin's approach to rapid development.
- RAD approaches to SD put less emphasis on planning and more emphasis on process.
- RAD approaches emphasize adaptability and the necessity of adjusting requirements in response to knowledge gained as the project progresses.
- Prototypes are often used in addition to or sometimes even in place of design specifications.
- RAD is especially well suited for (although not limited to) developing software that is driven by user interface requirements.
- Other approaches to rapid development include Agile methods and the spiral model

James Martin's approach to **RAD**



RAD – Pros and Cons

Advantages (Pros)

- Better quality
- Risk control
- More projects completed on time and within budget

Disadvantages (Cons)

- The risk of a new approach
- Requires time of scarce resources
- Less control
- Poor design
- Lack of scalability

AGILE Software Development

- Agile software development describes a set of principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams
- First coined in 2001, in the Manifesto for Agile Software Development

Manifesto for Agile SD

- **Individuals and Interactions** more than processes and tools
- **Working Software** more than comprehensive documentation
- **Customer Collaboration** more than contract negotiation
- **Responding to Change** more than following a plan

Agile Development – 12 Principles

- Customer satisfaction by early and continuous delivery of valuable software
- Welcome changing requirements, even in late development
- Working software is delivered frequently (weeks rather than months)
- Close, daily cooperation between business people and developers
- Projects are built around motivated individuals, who should be trusted
- Face-to-face conversation is the best form of communication (co-location)
- Working software is the primary measure of progress
- Sustainable development, able to maintain a constant pace
- Continuous attention to technical excellence and good design
- Simplicity—the art of maximizing the amount of work not done—is essential
- Best architectures, requirements, and designs emerge from self-organizing teams
- Regularly, the team reflects on how to become more effective, and adjusts accordingly

Agile Development - Pitfalls

- Lack of overall product design
- Adding stories to an iteration in progress
- Lack of sponsor support
- Insufficient training
- Product owner role is not properly filled
- Teams are not focused
- Excessive preparation/planning
- Problem-solving in the daily standup
- Fixed time, resources, scope, and quality
- Attempting to take on too much in an iteration
- Allowing technical debt to build up

Lean

- Lean software development (LSD) is a translation of lean manufacturing and lean IT principles and practices to the software development domain
- Adapted from the Toyota Production System, it emerged from within the Agile community

Lean principles

- Eliminate waste
- Amplify learning
- Decide as late as possible
- Deliver as fast as possible
- Empower the team
- Build integrity in
- See the whole

How code reviews work

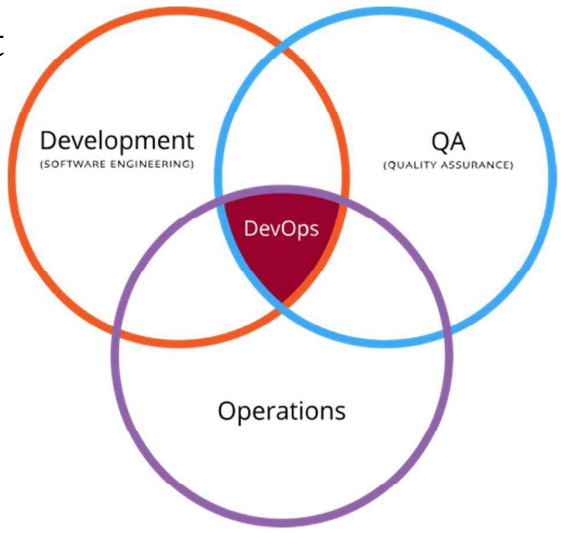
- **Code review** is "a process where two or more developers visually inspect a set of program code, typically, several times"
- Objectives:
  - Best Practice
  - Error Detection
  - Vulnerability Exposure
  - Malware Discovery
- Two main categories: **formal** code review and **lightweight** code review
- **Code Review Saves Time and Cost!!!**

Code review resources

- http://www.methodsandtools.com/archive/archive.php?id=66
- http://www.ganssle.com/inspections.pdf

How DevOps works

- DevOps (a clipped compound of development and operations) is a culture, movement or practice that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes



How Scrum works

- Scrum is an iterative and incremental agile software development framework for managing product development
- It defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal"
- It challenges assumptions of the "traditional, sequential approach" to product development
- It enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines involved

1.3 Software Development Methodologies

Q&A

Exercises and Review Questions

Sample Question #1

_____ defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal"

A. Scrum

B. DevOps

C. Agile

D. Project Management

Sample Question #1 - Answer

_____ defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal"

A. Scrum

B. DevOps

C. Agile

D. Project Management

Sample Question #2

RAD stands for:

A. Rapid Agile Development

B. Random Application Design

C. Rapid Application Development

D. Random Agile Design

Sample Question #2 - Answer

RAD stands for:

A. Rapid Agile Development

B. Random Application Design

C. Rapid Application Development

D. Random Agile Design

Sample Question #3

A splitting of software development work into distinct phases (or stages) containing activities with the intent of better planning and management describes:

A. Requirements capturing
B. Project management
C. Software as a Service
D. Application development

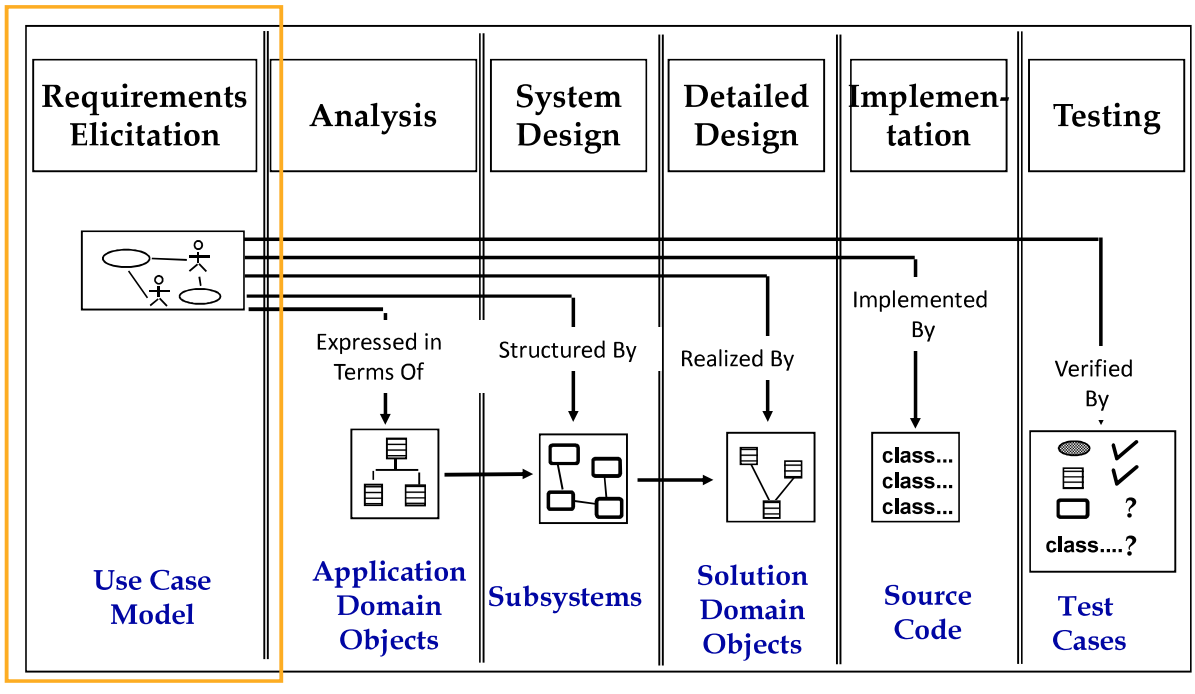
Sample Question #3 - Answer

A splitting of software development work into distinct phases (or stages) containing activities with the intent of better planning and management describes:

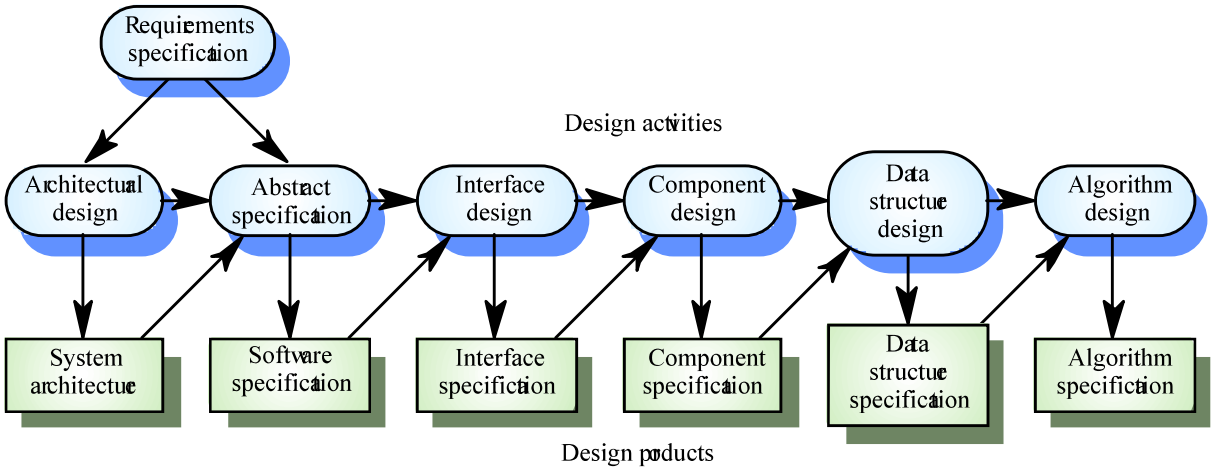
A. Requirements capturing
B. Project management
C. Software as a Service
D. Application development

1.4 Requirements Capturing and Software Design

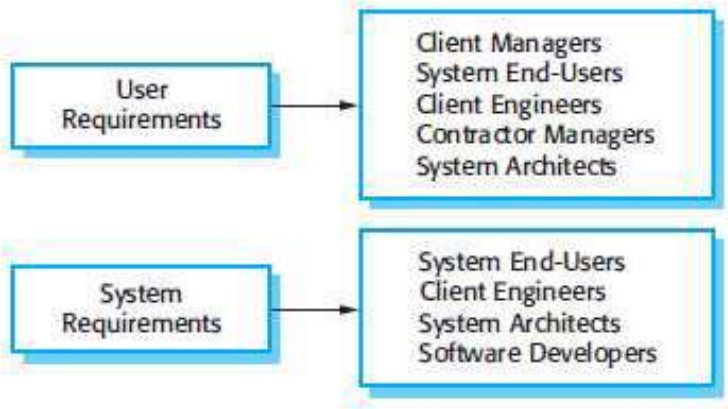
Software Lifecycle Activities



The software design process

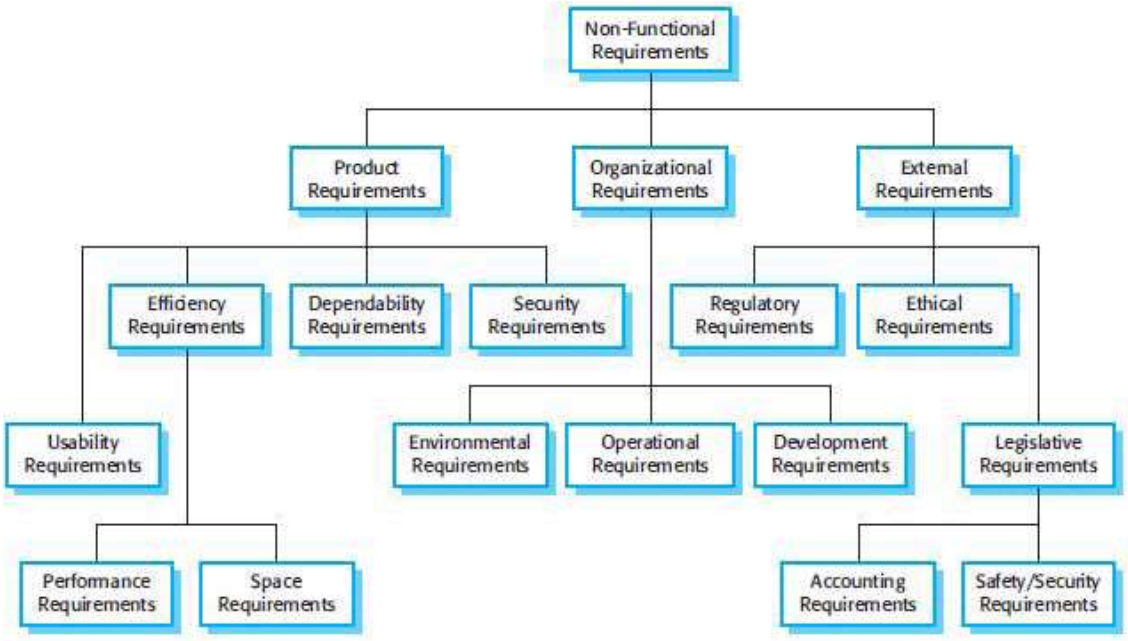


Requirements capturing

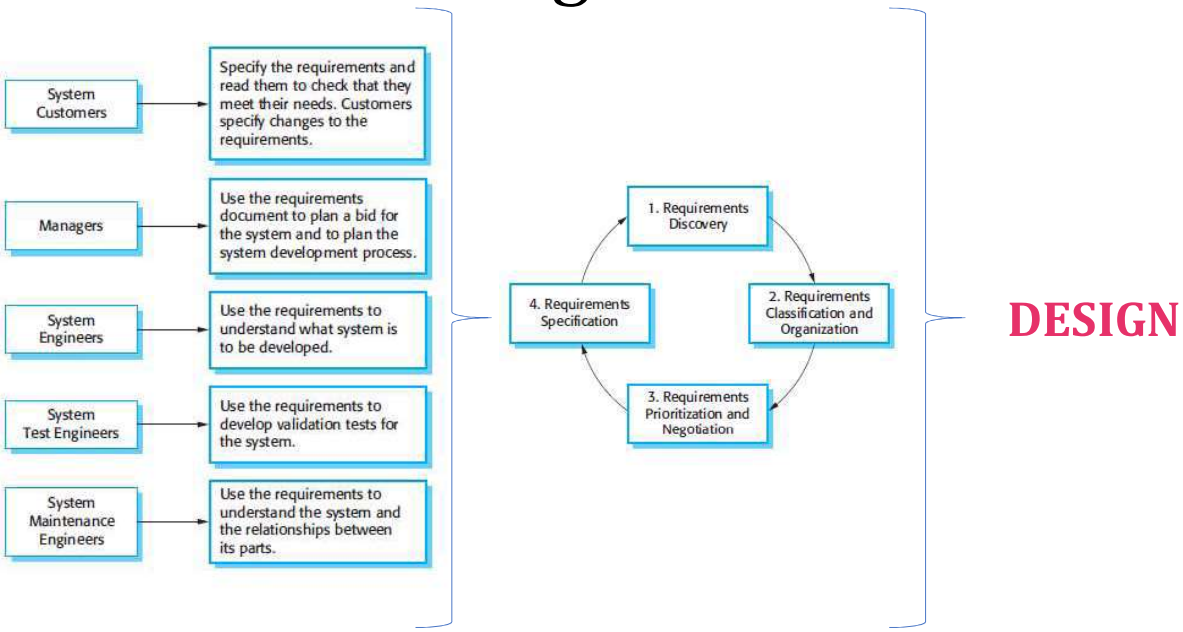


- Determine the needs or conditions to meet taking account of the possibly conflicting requirements of the various stakeholders, **analyzing, documenting, validating and managing** software or system requirements

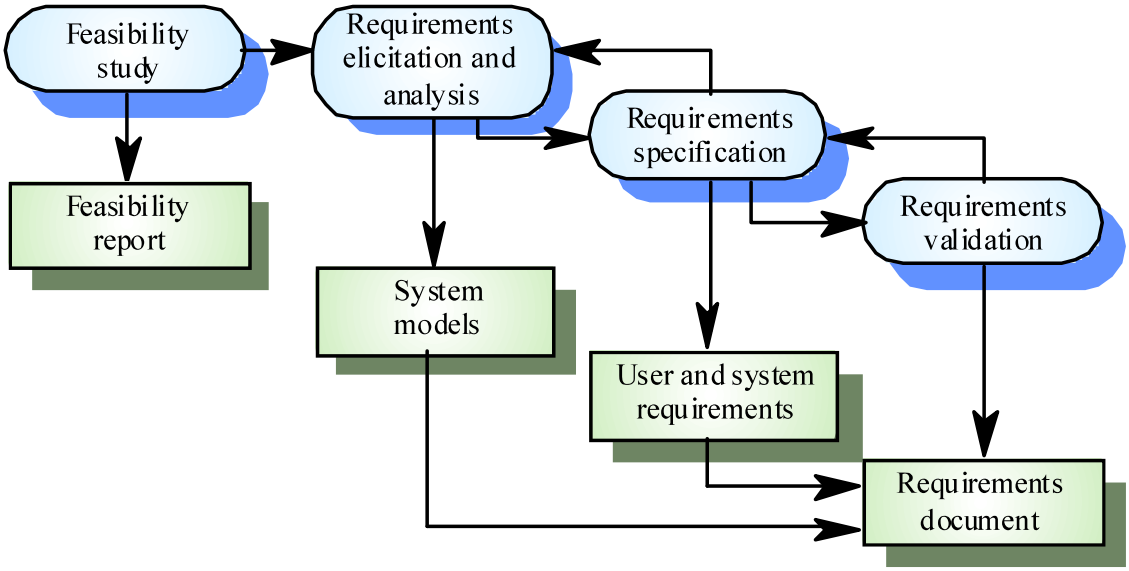
Requirements capturing



From requirements capturing to software design



The requirements engineering process



5 Principles to Good Requirements

1. **Communicate Input to Design**
   - What are we solving?
   - Why is this function important?
   - Clarity to Cross-Functional Team
2. **Measurable & Testable**
   - Verification and Validation are Possible
   - Subjective Requirements cannot be Verified
3. **Requirements are Focused**
   - Audience for Requirement is known
4. **Provide Value to Development**
   - Based on Need: Answer WHY?
5. **Free of Specific Design Content**

Class discussion

- Consider a simple game, like tic-tac-toe
  - **Draw a flowchart** showing how the game works
  - What are some basic **requirements**?

Unified Modelling Language (UML)

- Use Case Diagram
- Class Diagram
- Sequence Diagram
- State Transition Diagram

Use Case Diagram

Figure 5.3 Transfer-data use case

- A **use case diagram** is "a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved".

Class Discussion / Exercise

Using the previous diagram as an example, draw a **Use Case Diagram** Showing:

- A medical receptionist
- 4 Use Cases
  - Register patient
  - Unregister Patient
  - View Patient Info
  - Transfer Data
  - Contact Patient

Use Case Diagram (Discussion)

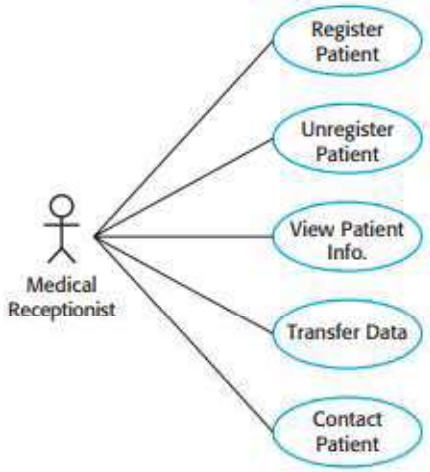


Figure 5.5 Use cases involving the role 'medical receptionist'

Use Case Diagram

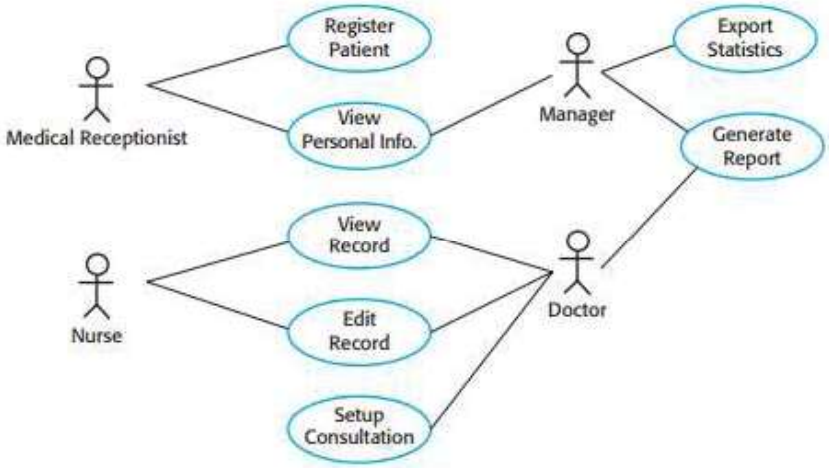


Figure 4.15 Use cases for the MHC-PMS

Class Diagram

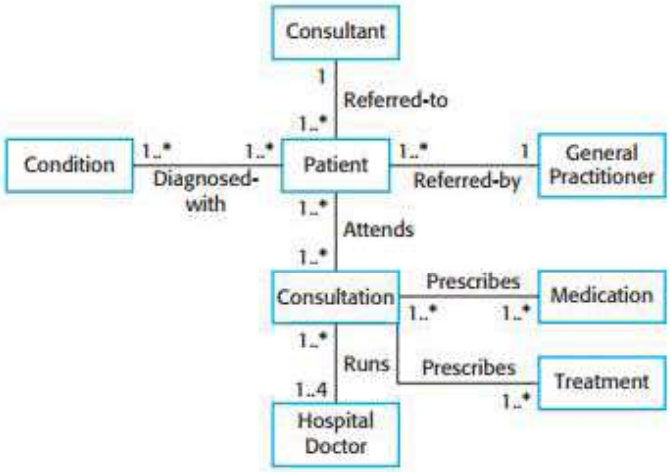


Figure 5.9 Classes and associations in the MHC-PMS

Sequence Diagram

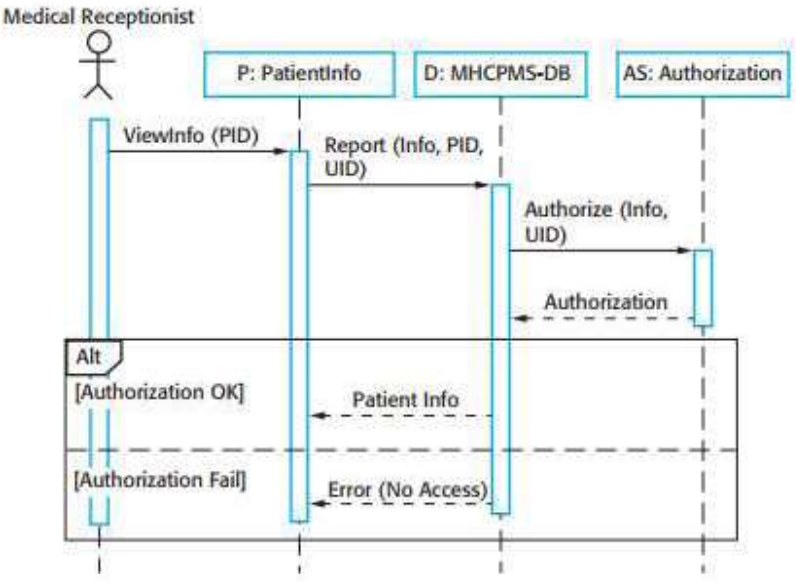


Figure 5.6 Sequence diagram for View patient information

- **A sequence diagram** is an interaction diagram that shows how objects operate with one another and in what order

State Transition Diagram

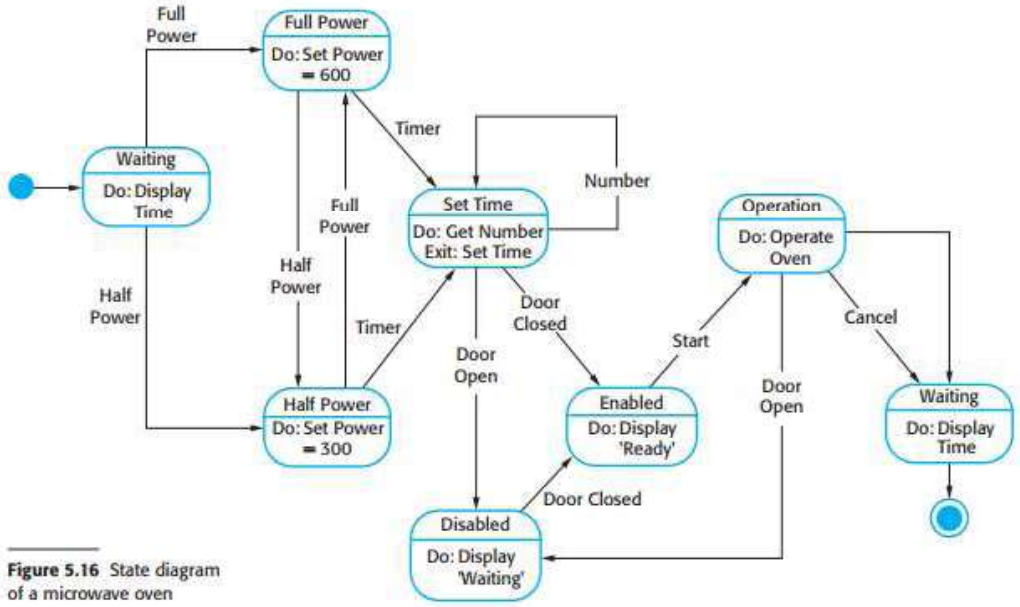


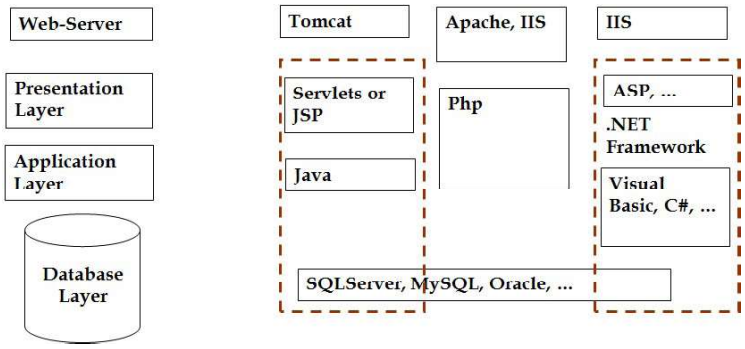
Figure 5.16 State diagram of a microwave oven

- A **state diagram** is a type of diagram used in computer science and related fields to describe the behavior of systems.

Software Development Tools

- Application Development Frameworks
- Content Management Systems (CMS)
- Popular Content management platforms
- Open source e-Commerce platforms
- Software development tools

Application Development Frameworks



Content Management Systems (CMS)

- A content management system, or CMS, is a web application designed to make it easy for non-technical users to add, edit and manage a website
- Automatically generate navigation elements
- Making content searchable and indexable
- Keeping track of users, their permissions and security settings
- and much, much more

Popular CMS platforms

Alliance For Digital Employability

Joomla!

WordPress.org

Drupal

Open source e-Commerce platforms

Alliance For Digital Employability

PRESTASHOP

cubecart

AGORACart

Magento

osCommerce

zen cart the art of e-commerce

OPENCART Open Source Shopping

Software Development Tools

- IDE (Integrated development environment)
- Netbeans
- Eclipse
- .NET
- Version Repositories
- CVS (Concurrent Versions System)
- IDEs with CVS support: Emacs , Anjuta, Dev-C++, Eclipse, NetBeans, IntelliJ IDEA, wxDev-C++, Kdevelop, Aqualogic, Xcode, PhpED
- and many, many more

1.4 Requirements Capturing and Software Design

Q&A

Exercises and Review Questions

Sample Question #1

"An interaction diagram that shows how objects operate with one another and in what order" is called a:

A. Flowchart
B. Sequence Diagram
C. State Transition Diagram
D. Class Diagram

Sample Question #1 - Answer

"An interaction diagram that shows how objects operate with one another and in what order" is called a:

A. Flowchart
B. Sequence Diagram
C. State Transition Diagram
D. Class Diagram

Sample Question #2

The process of collecting the user needs to solve a problem or an issue and achieve and object is known as:

A. Requirements analysis
B. System analysis
C. State Transition Diagram
D. Class Diagram

Sample Question #2 - Answer

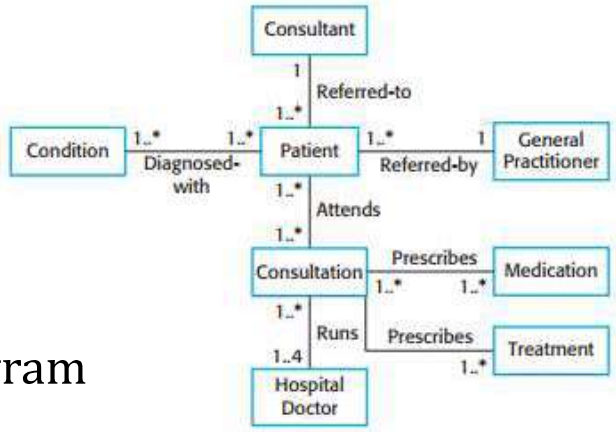
The process of collecting the user needs to solve a problem or an issue and achieve and object is known as:

A. Requirements analysis
B. System analysis
C. State Transition Diagram
D. Class Diagram

Sample Question #3

The image on the right shows a:

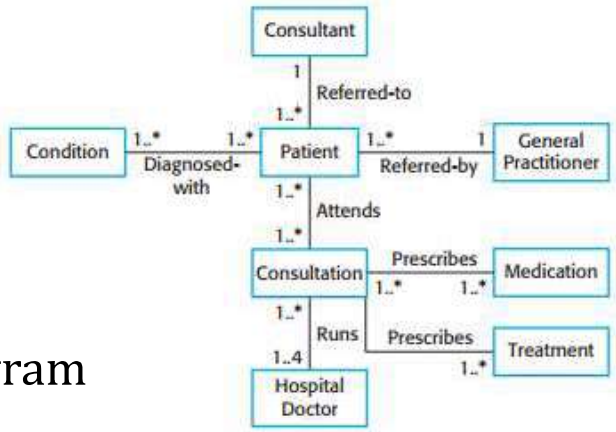
A. Sequence diagram
B. Class Diagram
C. Use Case Diagram
D. State Transition Diagram



Sample Question #3 - Answer

The image on the right shows a:

A. Sequence diagram
B. Class Diagram
C. Use Case Diagram
D. State Transition Diagram



Introduction to Programming

Syllabus 2.1 – 2.3

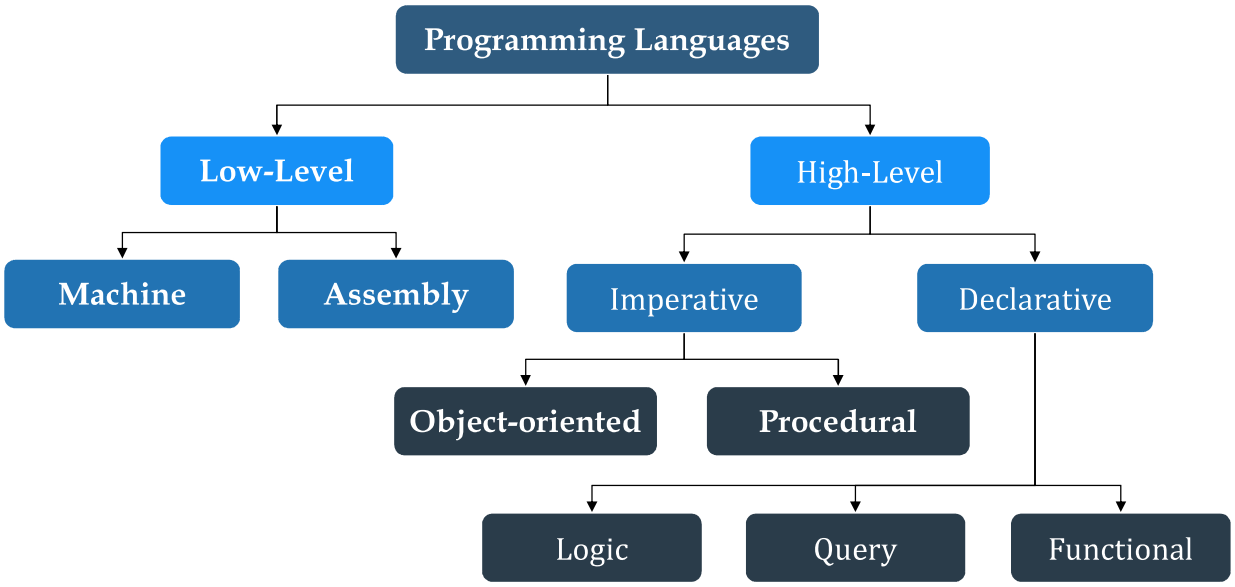
2.1 Developer Tools	2.1.1	Understand and use editors and compilers
	2.1.2	Perform static analysis (e.g. FindBugs)
	2.1.3	Understand and use the Unix command line and its tools
	2.1.4	Use build tools (e.g. make, Maven)
	2.1.5	Understand and use IDE
	2.1.6	Understand and use Github
	2.1.7	Understand and use GIT-Version Control
	2.1.8	Understand and perform debugging
	2.1.9	Understand and perform optimisation
2.2 Programming Languages and Paradigms	2.2.1	List the different types of programming languages like: procedural, object-oriented, data oriented, scripting. Distinguish between compiled and interpreted languages
	2.2.2	Distinguish between Back-end and front-end software development
2.3 Programming Basics	2.3.1	Define variables, write and execute a simple program
	2.3.2	List and use Command-line basics, like: editing, execution, stopping, paging, redirecting, piping
	2.3.3	Use Strings and Console Output, including creating string literals, calling a variety of string methods
	2.3.4	Use conditionals, loops and control flow
	2.3.5	Define functions
	2.3.6	Define and use lists, list comprehensions, dictionaries
	2.3.7	Handle input-output
	2.3.8	Connect a database to the HTML pages to display data via Python

Alliance For Digital Employability

2.1 Developer Tools

Alliance For Digital Employability

Classification of Programming Languages



Programming Languages

- The **three major families** of languages are:
  - Machine languages
  - Assembly languages
  - High-Level languages

```
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122
; otherwise leaves it unchanged.

SUB32  PROC        ; procedure begins here
       CMP  AX,97  ; compare AX to 97
       JL   DONE   ; if less, jump to DONE
       CMP  AX,122 ; compare AX to 122
       JG   DONE   ; if greater, jump to DONE
       SUB  AX,32  ; subtract 32 from AX
DONE:  RET         ; return to main program
SUB32  ENDP        ; procedure ends here
```

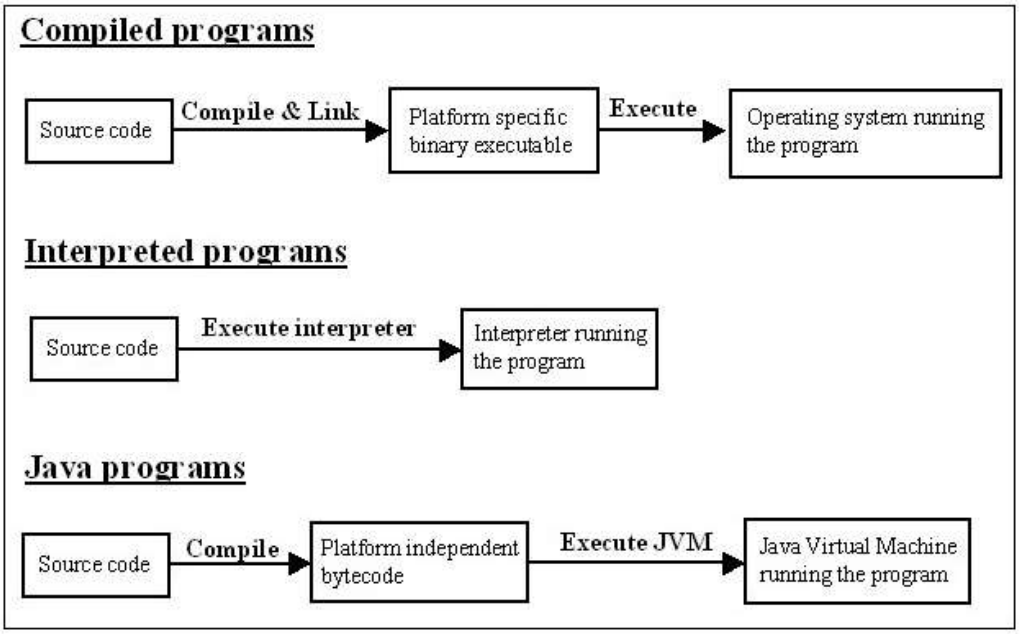
```
public void processData()
{
  do
  {
    int data = getData();
    if(data < 0)
      performOperation1(data);
    else
      performOperation2(data);
  }
  while(hasMoreData());
}
```

Machine Language

- Comprised of 1s and 0s
- The “native” language of a computer
- Difficult to program – one misplaced 1 or 0 will cause the program to fail
- Code example:

```
1110100010101
111010101110
10111010110100
10100011110111
```

Compiled vs. Interpreted Languages



Perform static analysis

- **Static program analysis** is the analysis of computer software that is performed without actually executing programs
- Analysis performed on executing programs is known as dynamic analysis).[
- [n most cases the analysis is performed on some version of the source code, and in the other cases, some form of the object code.

Unix command line and its tools

- Shell programming
  - Command — Execute a simple command
  - Echo — Write arguments to standard output
  - Printf — Write formatted output
- File System
  - cd — Change the working directory
  - cp — Copy files
  - df — Report free disk space
  - ls — List directory contents
  - mkdir — Make directories

Class activity

- Open your command prompt and try various commands
  - cd
  - dir
  - etc…

Build tools

- A build tool is a programming utility that is used when building a new version of a program.

For example:

- make is a popular open source build tool that uses makefile, another build tool, to ensure that source files that have been updated (and files that are dependent on them) will be compiled into a new version (build) of a program
- MSBuild: The build system for Microsoft and Visual Studio.

IDE

- **Integrated Development Environment (IDE)**
- A software application that provides comprehensive facilities to computer programmers for software development

Consists of a

- source code editor
- build automation tools and
- a debugger

Class Activity

- Allow students to browse the Internet and locate IDE examples

Github

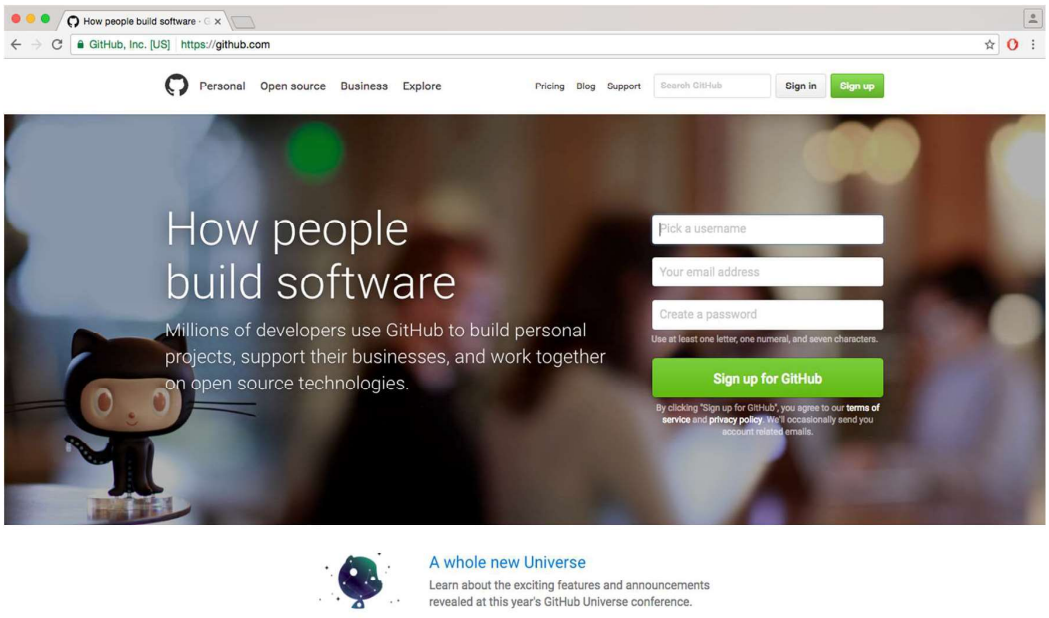
- GitHub is a web-based Git repository hosting service.
- Offers all of the distributed version control and source code management (SCM) functionality.
- Provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

- Why use Github?

  You are going to upload and edit your excercises on your personal GitHub page

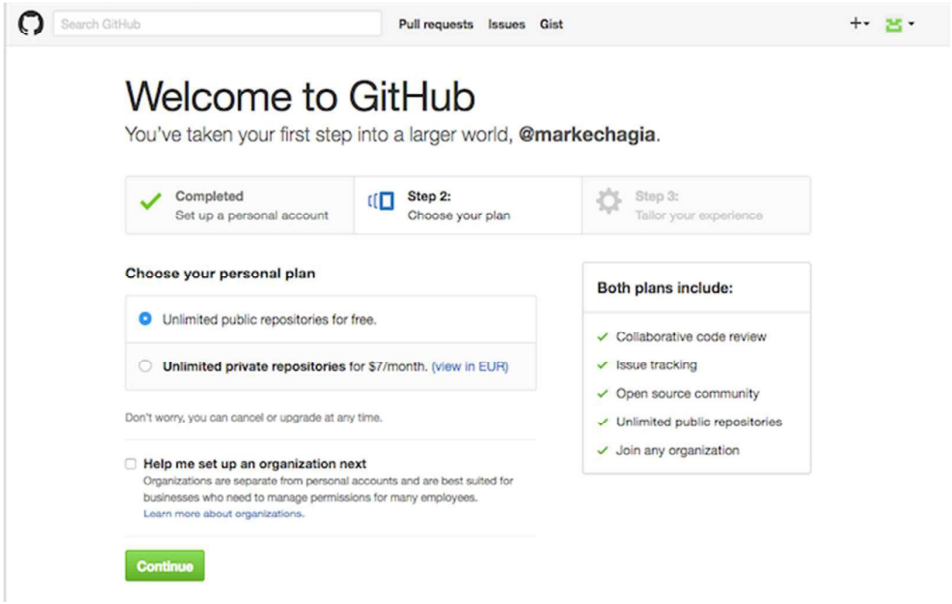
Signup!

- github.com



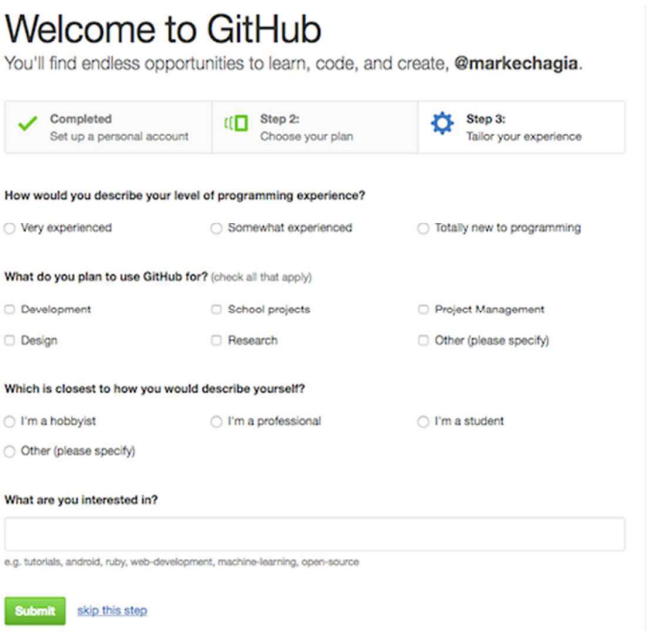
Create your profile

- Select the free repositories option and then press Continue



Create your profile (2)

- Answer if you want or skip this page for now

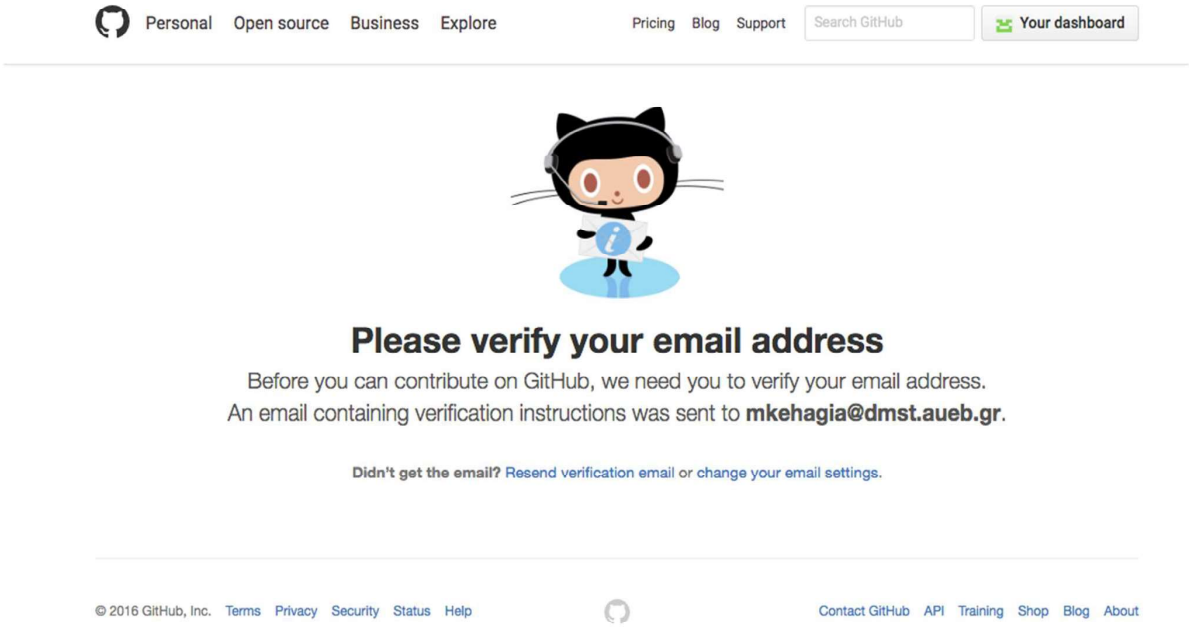


Create your profile (3)

- Press start a project

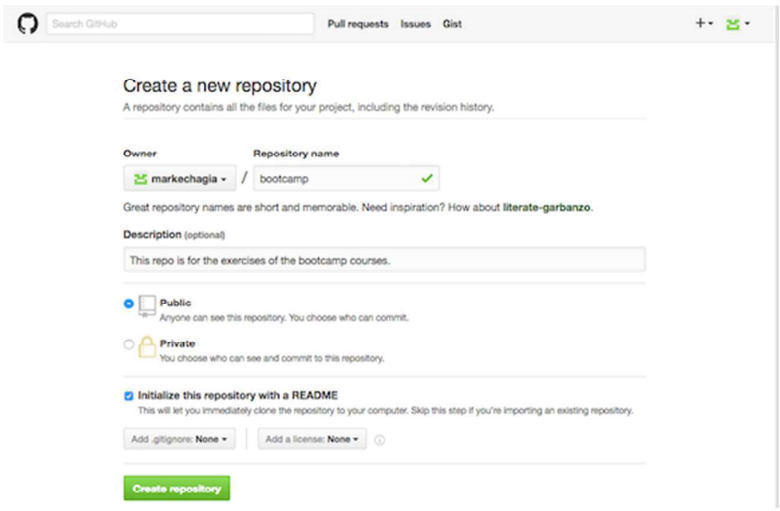
Create your profile (4)

- Check your inbox and verify your email address

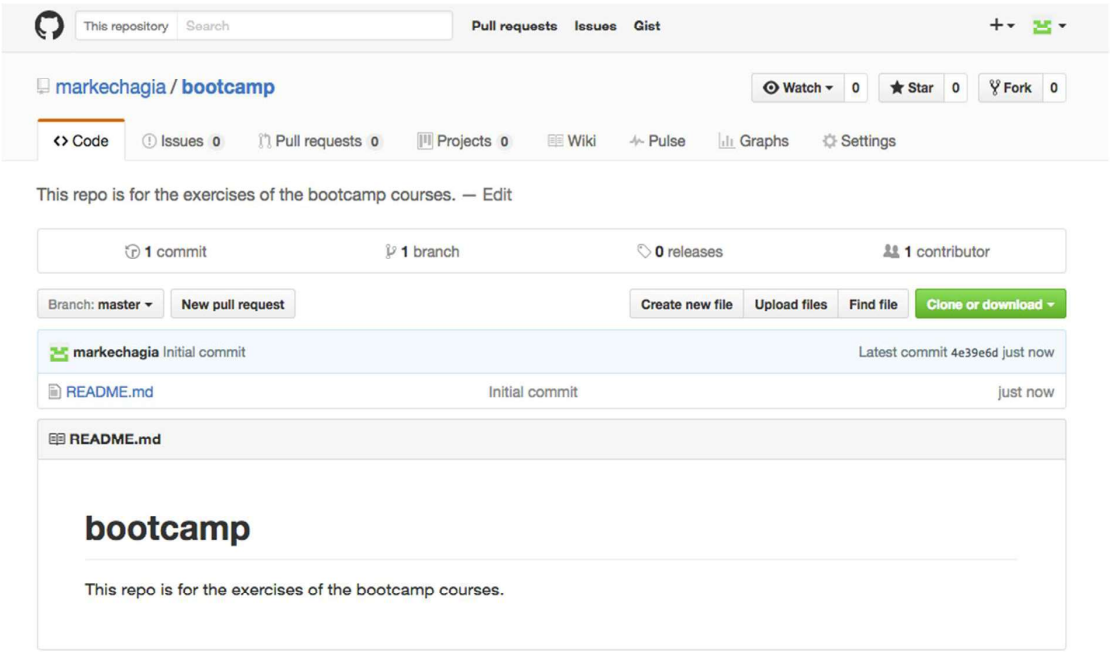


Create a repo

- Give a name to your repo
- Write a description for your repo
- Select to be public
- Select to initialize the repo with a README file

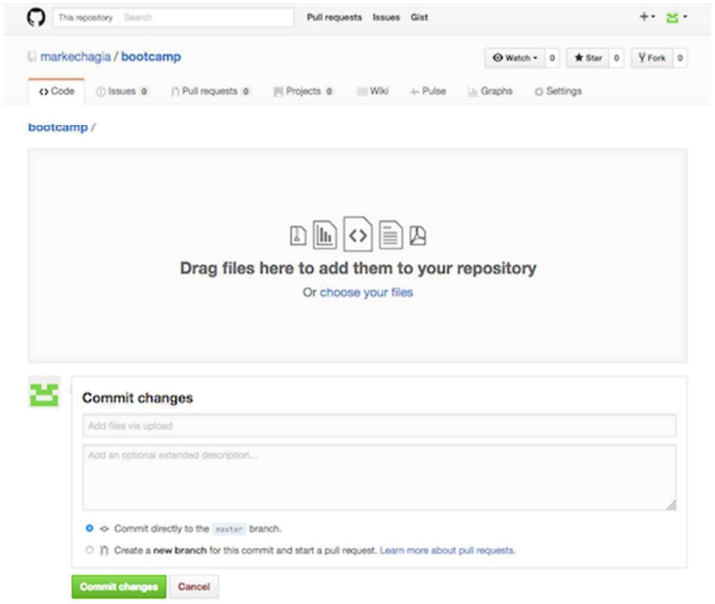


Create a repo (2)



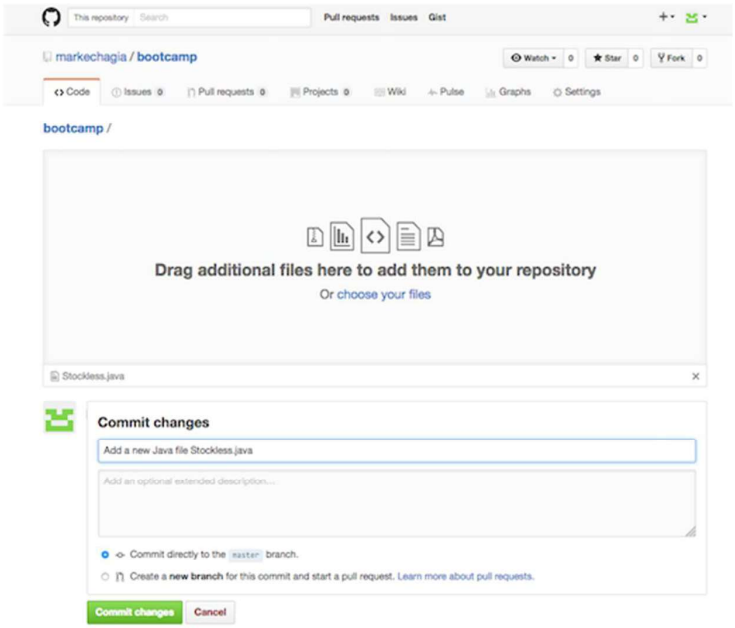
Add a new file to the repo

- Select to upload a new file
- Drag and drop your file

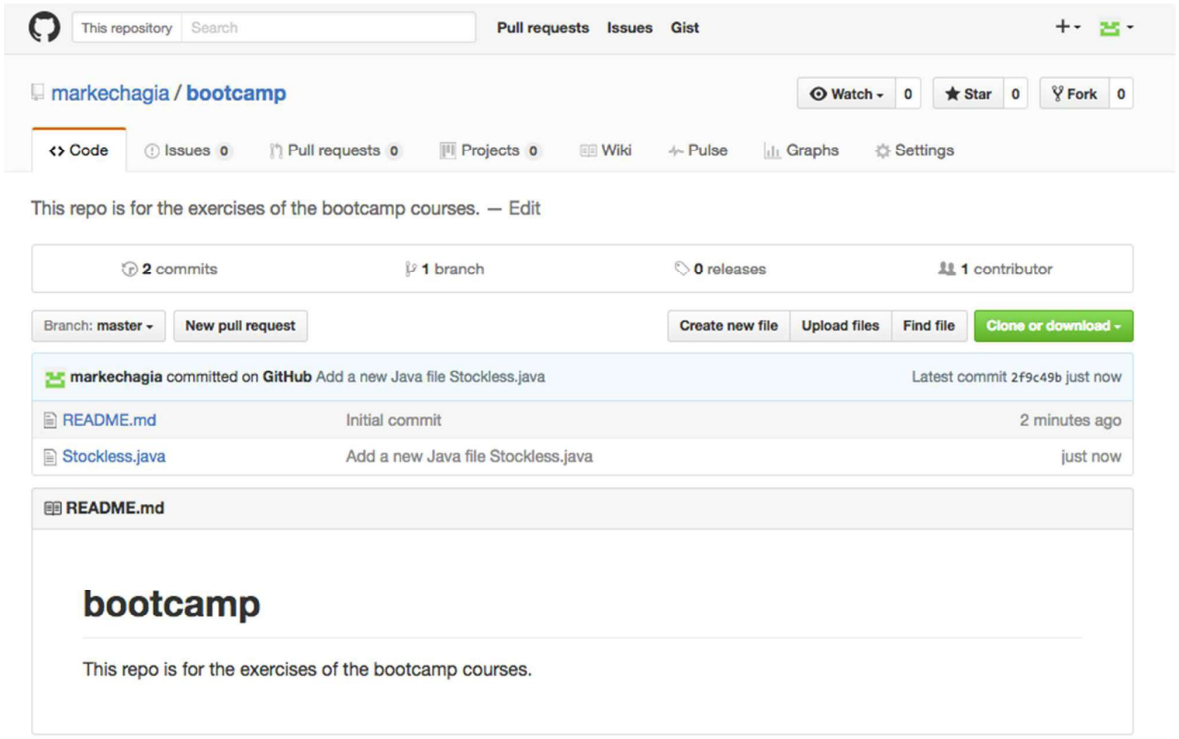


Add a new file to the repo (2)

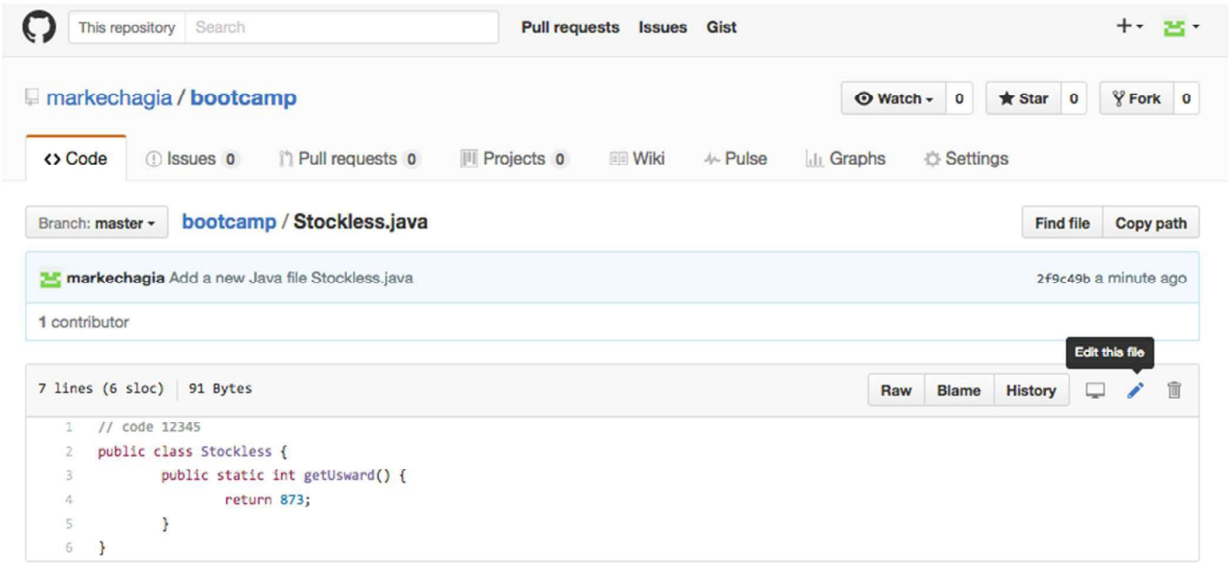
- After you drop the file commit your changes
- Write a message and press commit



Add a new file to the repo (3)

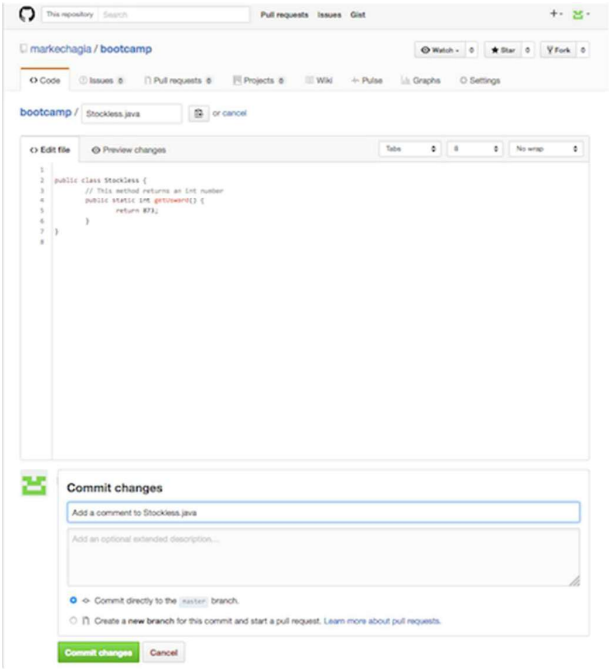


Edit a file

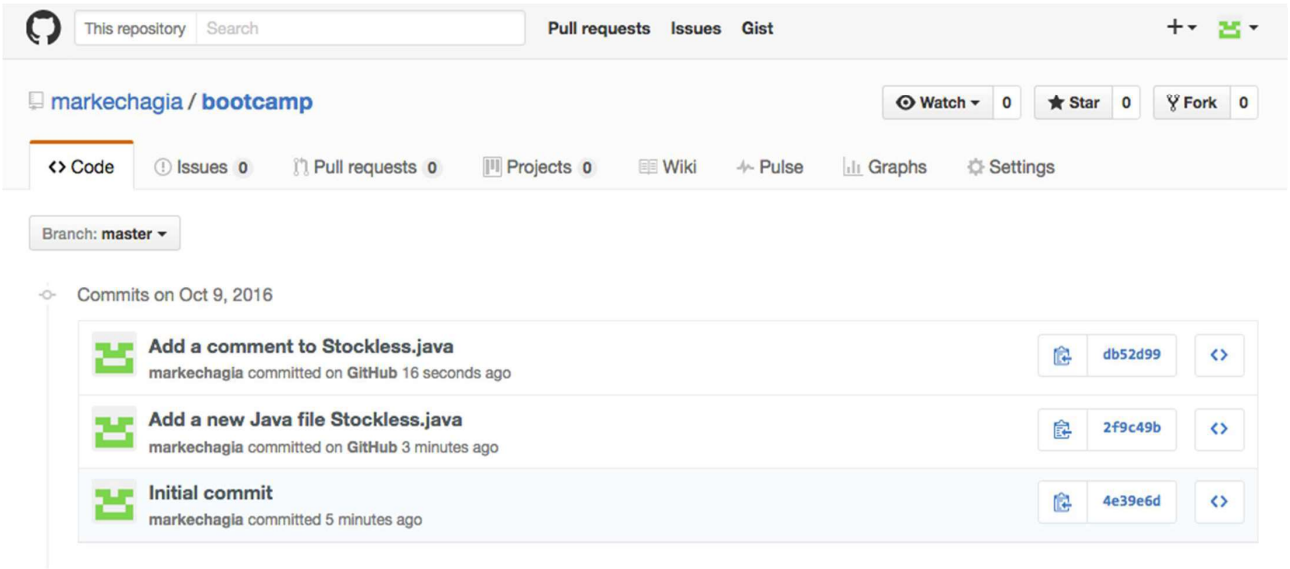


Edit a file (2)

- Make and commit your changes



Repo files



Class Activity

- Students should create their own GitHub accounts!

2.1 Developer Tools

Q&A

Exercises and Review Questions

Sample Question #1

A front-end application which include GUI is created using _____.

A. Compiler

B. IDE

C. Interpreter

D. Command line

Sample Question #1 - Answer

A front-end application which include GUI is created using _____.

A. Compiler

B. IDE

C. Interpreter

D. Command line

Sample Question #2

What does the **sleep** command (UNIX Command line) line?

A. Write arguments to standard output

B. Evaluate arguments as an expression

C. Execute a simple command

D. Suspend execution for an interval

Sample Question #2 - Answer

What does the **sleep** command (UNIX Command line) line?

A. Write arguments to standard output

B. Evaluate arguments as an expression

C. Execute a simple command

D. Suspend execution for an interval

Sample Question #3

What is GitHub?

A. A control repository hosting service

B. An IDE

C. A Compiler

D. A Static Analysis tool for code review

Sample Question #3 - Answer

What is GitHub?

A. A control repository hosting service

B. An IDE

C. A Compiler

D. A Static Analysis tool for code review

2.2 Programming Languages and Paradigms

Different types of programming languages

- Procedural
- Object-oriented
- Data oriented
- Scripting
- Compiled and interpreted languages

Procedural languages

- Procedural programming languages are based on the concept of the unit and scope (the data viewing range of an executable code statement).
- A procedural program is composed of one or more units or modules, either user coded or provided in a code library; each module is composed of one or more procedures, also called a function, routine, subroutine, or method, depending on the language

Examples include: COBOL, Pascal, Python, Fortran

Object Oriented Languages

Object-oriented programming (OOP) is a programming paradigm based on:

- the concept of "**objects**"
- **Code**, in the form of procedures, often known as **methods**.
- In OOP, computer programs are designed by making them out of objects that interact with one another.
- Most popular languages are **class-based**, meaning that objects are instances of classes, which typically also determine their type.
- Examples: C++, Object Pascal, Java, etc.

Data Oriented Languages

- Data-oriented languages provide powerful ways of searching and manipulating the relations that have been described as entity relationship tables which map one set of things into other sets.
- Examples of data-oriented languages include:
  - SQL
  - dBase
  - Clipper
  - Clarion

Scripting Languages

Scripting language has two apparently different, but in fact similar, meanings.

- In a traditional sense, scripting languages are designed to automate frequently used tasks that usually involve calling or passing commands to external programs. Many complex application programs provide built-in languages that let users automate tasks. Those that are interpretive are often called scripting languages.
- Recently, many applications have built-in traditional scripting languages, such as Perl or Visual Basic, but there are quite a few native scripting languages still in use.

Interpreted Languages

- An **interpreted** language is a programming language for which most of its implementations execute instructions directly, without previously compiling a program into machine-language instructions.
- The interpreter executes the program directly, translating each statement into a sequence of one or more subroutines already compiled into machine code.

Compiled Languages

- A **compiled language** is a programming language whose implementations are typically compilers (translators that generate machine code from source code), and not interpreters (step-by-step executors of source code, where no pre-runtime translation takes place).

Compiled vs Interpreted Languages

- In principle, any language can be implemented with a compiler or with an interpreter.
- A combination of both solutions is also common: a compiler can translate the source code into some intermediate form (often called p-code or bytecode), which is then passed to an interpreter which executes it.

Back-end and front-end software development

Refers to the:

1) separation of concerns between the presentation layer (front end), and the data access layer (back end) of a piece of software
2) physical infrastructure or hardware.

Example: Client–server model:

- the client considered the front end
- server is usually considered the back end

Back-end software development

- code that runs on the server
- sending meaningful data to the frontend
- interacts with the database
- Popular backend languages include PHP, Java, C#, Python and Ruby

Example: *A person creates a new Twitter account*

the backend is responsible for saving user to the database.

when saved the backend can then fetch, update and delete this user

Front end software development

- code that runs in the browser
- structure the data that was sent from the backend
- To make it pretty – we use CSS
- To make it interactive we use JavaScript
- combination of HTML, CSS and JavaScript — collectively known as the frontend — creates the User Interface (UI)

Example: *The look on your Twitter account and DM interactivity*

when choosing how your account looks, you are using CSS

when a messaging modal/pop-up appears when another user's name in is clicked, JavaScript is used to add that interactivity

The Database

- has columns and rows
- contains all the important information that your website needs to remember
- Stores information

Example: *Your latest tweet on your Twitter account*

you *tweeted* on your Twitter timeline
you navigated away from the site
when you return to twitter your tweet is on your timeline because it was saved to the database

Class Discussion

- Discuss with students popular web applications, like social media in order to understand how they work; distinguish between their front-end, their back-end and how data is stored and retrieved.

2.2 Programming Languages and Paradigms

Q&A

Exercises and Review Questions

Sample Question #1

Which of the following is an interpreted programming language?

A. Delphi

B. ADA

C. C#

D. JavaScript

Sample Question #1 - Answer

Which of the following is an interpreted programming language?

A. Delphi

B. ADA

C. C#

D. JavaScript

Sample Question #2

Which of the following languages is **NOT** usually used for Back-End Software development?

A. PHP

B. Java

C. JavaScript

D. C#

Sample Question #2 - Answer

Which of the following languages is **NOT** usually used for Back-End Software development?

A. PHP

B. Java

C. JavaScript

D. C#

Sample Question #3

Suppose you log on your Facebook account and look for and locate your last post. Where is that post actually stored?

A. Facebook's database

B. Facebook's front end

C. Facebook's back end

D. On your computer's drive

Sample Question #3 - Answer

Suppose you log on your Facebook account and look for and locate your last post. Where is that post actually stored?

A. Facebook's database

B. Facebook's front end

C. Facebook's back end

D. On your computer's drive

2.3 Programming Basics

Variables

Definition: "a data item that may take on more than one value during the runtime of a program"

Consists of:

- a **storage location** (memory address) paired with
- an associated symbolic name (aka **identifier**)
- and contains some known or unknown quantity of information referred to as a **value**

Python Example:

```
counter = 100          # An integer assignment
miles = 1000.0         # A floating point
name = "John"          # A string print counter print miles print
name
```

Class Exercises

Write a simple program in Python where:

- Height is a variable named **ht**
- Weight is a variable **wg**
- **BMI** is calculated based on the formula:

  BMI = wg / (ht * ht)
- **The BMI is then evaluated as:**

  Underweight = <18.5
  Normal weight = 18.5–24.9
  Overweight = 25–29.9
  Obesity = BMI of 30 or greater

Command-line basics

- Editing
- Execution
- Stopping
- Paging
- Redirecting
- Piping

Strings and Console Output

- The **Console** is a window of the operating system through which users can interact with system programs of the operating system or with other console applications
- most programming languages access the console
- reading text and numbers from the console and printing text and numbers

Conditionals, Loops & Control Flow

Let's remember some basic terms:

Conditionals: Perform different computations or actions depending on whether a condition evaluates to true or false.

i.e. If statement

Loops: executes one or more statements up to a desired number of times, based on conditions

i.e. For, While

Conditionals, Loops & Control Flow

Control Flow:

- Shown in the flow chart of a program
- the order in which individual statements, instructions or function calls of an imperative program are executed or evaluated
- the emphasis on explicit control flow distinguishes an imperative programming language from a declarative programming language
- In an imperative programming language, a control flow statement is a statement the execution of which results in a choice being made as to which of two or more paths to follow

i.e. GoTo statements, count-controlled loops, condition-controlled loops etc.

Class exercise

Write a simply Python program that will transform dog years to human years assuming that:

- 0 years is an impossible number
- The first two years of a dogs life are equivalent of a humans in years
- Every further dog year corresponds to five years in the life of a human being
- A dog can't be over 30 years old (in dog years)

Class exercise – sample answer

```
age = input("Age of the dog: ")
print if age < 0:
        print "That is impossible!"
elif age == 1:
        print "1 human years"
elif age == 2:
        print "2 human years"
elif age > 2:
        human = 2 + (age -2)*5
        print "Human years: ", human
###
raw_input('press Return>')
```

Class discussion

What are the problems with this algorithm?

Could you do it within a loop or using control flow?

Simply discuss options now…we will try this in your Java or C# environment in the upcoming classes!

Functions

- a block of organized, reusable code that is used to perform a single, related action
- provide better modularity for your application
- Provide a high degree of code **reusing**

Function (example)

```
def printme( str1 ):
   "This prints a passed string into this function"
   print str1
   return
```

Call function from another function or directly from the Python prompt#!/usr/bin/python

```
#!/usr/bin/python
# Function definition is here
def printme( str1 ):
     "This prints a passed string into this function“
     print str1
     return;

# Now you can call printme function
printme(“First call to my new ud function!")
printme(“Second call to the same function")
```

When the above code is executed, it produces the following result –

- First call to my new ud function!
- Second call to the same function

Lists, list comprehensions, dictionaries

List comprehensions (Python)

- provide a concise way to create lists
- consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses
- Expressions can be anything, meaning you can put in all kinds of objects in lists
- The result will be a new list resulting from evaluating the expression in the context of the for and if clauses which follow it
- The list comprehension always returns a result list

Lists, list comprehensions, dictionaries (2)

Dictionaries (Python)

- Each key is separated from its value by a colon (:),
- items are separated by commas
- the whole thing is enclosed in curly **bracesAn** empty dictionary without any items is written with just two curly braces, like this: {}.
- **Keys** are **unique** within a dictionary while values may not be
- the values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples

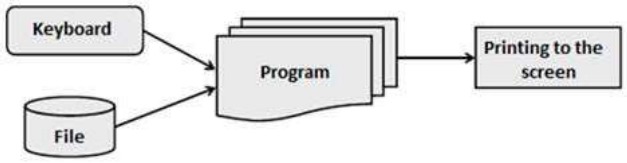
Example: dict = {'Name': 'Anna', 'Age': 27, 'Class': 'Java'}

Input-Output

- The standard input-output also known as "**Standard I/O**" is a system input-output mechanism created since the UNIX operating systems was developed many years ago
- Special peripheral devices for input and output are used, through which data can be input and output.
- When the program is in mode of accepting information and expects action by the user, there is a blinking cursor on the console showing that the system is waiting for command entering.

Input-Output (2)

- Besides the keyboard an application **input** can come from many other places, such as file, microphone, barcode reader and others.
- The **output** of a program may be on the console (on the screen), as well as in a file or another output device, such as a printer



Input-Output (3)

Input

- **The standard input device** is the part of the operating system that controls from where the program will receive its input data.
- Each programming language has a mechanism for reading and writing to the console. The object that controls the standard input stream in **C#**, is **Console.In**.

Alliance For Digital Employability

More on variables, functions, conditionals, loops, control flow, Input-Output and Strings and Console Output in your JAVA or C# classes!!

Alliance For Digital Employability

2.3 Programming Basics

Q&A

Review Questions

Sample Question #1

Which of the following command refers to a loop?

A. While

B. If

C. Else

D. print

Sample Question #1 - Answer

Which of the following command refers to a loop?

A. While

B. If

C. Else

D. print

Sample Question #2

A flow chart can show:

A. Iteration

B. Control flow

C. Conditionals

D. All of the above

Sample Question #2 - Answer

A flow chart can show:

A. Iteration

B. Control flow

C. Conditionals

D. All of the above

2.3 Programming Basics

Let's get ready for our Programming Classes!

Tools Installation

C#

- Visual Studio 15 or 17
- MS SQLserver

Java

- Jdk v8
- Net beans
- Net beans plugins
- Notepad ++
- MySQL server
- Workbench for MySQL
- Tomcat webserver

Download Links

C#

- https://www.visualstudio.com/downloads/
- https://www.microsoft.com/en-us/sql-server/sql-server-downloads

Java

- http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- https://dev.mysql.com/downloads/mysql/
- https://java.com/en/download/
- https://netbeans.org/downloads/
- https://notepad-plus-plus.org/download/
- https://tomcat.apache.org/download-70.cgi

Thank you

Notes

Notes

Notes