

# 1E - Computer Programming and Methods of Software Development

**Bimal R. Desai, MD, MBI, FAAP, FAMIA**

Children's Hospital of Philadelphia

# Clinical Informatics Subspecialty Delineation of Practice (CIS DoP)

## Domain 1: Fundamental Knowledge and Skills (no Tasks are associated with this Domain which is focused on fundamental knowledge and skills)

### Clinical Informatics

K001. The discipline of informatics (e.g., definitions, history, careers, professional organizations)  
K002. Fundamental informatics concepts, models, and theories  
K003. Core clinical informatics literature (e.g., foundational literature, principle journals, critical analysis of literature, use of evidence to inform practice)  
K004. Descriptive and inferential statistics  
K005. Health Information Technology (HIT) principles and science

### K006. Computer programming fundamentals and computational thinking

K007. Basic systems and network architectures  
K008. Basic database structure, data retrieval and analytics techniques and tools  
K009. Development and use of interoperability/exchange standards (e.g., Fast Health Interoperability Resources [FHIR], Digital Imaging and Communications in Medicine [DICOM])  
K010. Development and use of transaction standards (e.g., American National Standards Institute X12)  
K011. Development and use of messaging standards (e.g., Health Level Seven [HL7] v2)  
K012. Development and use of ancillary data standards (e.g., imaging and Laboratory Information System [LIS])  
K013. Development and use of data model standards  
K014. Vocabularies, terminologies, and nomenclatures (e.g., Logical Observation Identifiers Names and Codes [LOINC], Systematized Nomenclature of Medicine –Clinical Terms [SNOMED-CT], RxNorm, International Classification of Diseases [ICD], Current Procedural Terminology [CPT])  
K015. Data taxonomies and ontologies  
K016. Security, privacy, and confidentiality requirements and practices  
K017. Legal and regulatory issues related to clinical data and information sharing  
K018. Technical and non-technical approaches and barriers to interoperability  
K019. Ethics and professionalism

### The Health System

K020. Primary domains of health, organizational structures, cultures, and processes (e.g., health care delivery, public health, personal health, population health, education of health professionals, clinical research)  
K021. Determinants of individual and population health  
K022. Forces shaping health care delivery and considerations regarding health care access  
K023. Health economics and financing  
K024. Policy and regulatory frameworks related to the healthcare system  
K025. The flow of data, information, and knowledge within the health system

## Domain 2: Improving Care Delivery and Outcomes

K026. Decision science (e.g., Bayes theorem, decision analysis, probability theory, utility and preference assessment, test characteristics)  
K027. Clinical decision support standards and processes for development, implementation, evaluation, and maintenance  
K028. Five Rights of clinical decision support (i.e., information, person, intervention formats, channel, and point/time in workflow)  
K029. Legal, regulatory, and ethical issues regarding clinical decision support  
K030. Methods of workflow analysis  
K031. Principles of workflow re-engineering  
K032. Quality improvement principles and practices (e.g., Six Sigma, Lean, Plan-Do-Study-Act [PDSA] cycle, root cause analysis)  
K033. User-centered design principles (e.g., iterative design process)  
K034. Usability testing  
K035. Definitions of measures (e.g., quality performance, regulatory, pay for performance, public health surveillance)  
K036. Measure development and evaluation processes and criteria  
K037. Key performance indicators (KPIs)  
K038. Claims analytics and benchmarks  
K039. Predictive analytic techniques, indications, and limitations  
K040. Clinical and financial benchmarking sources (e.g., Gartner, Healthcare Information and Management Systems Society [HIMSS] Analytics, Centers for Medicare and Medicaid Services [CMS], Leapfrog)  
K041. Quality standards and measures promulgated by quality organizations (e.g., National Quality Forum [NQF], Centers for Medicare and Medicaid Services [CMS], National Committee for Quality Assurance [NCQA])  
K042. Facility accreditation quality and safety standards (e.g., The Joint Commission, Clinical Laboratory Improvement Amendments [CLIA])  
K043. Clinical quality standards (e.g., Physician Quality Reporting System [PQRS], Agency for Healthcare Research and Quality [AHRQ], National Surgical Quality Improvement Program [NSQIP], Quality Reporting Document Architecture [QRDA], Health Quality Measure Format [HQMF], Council on Quality and Leadership [CQL], Fast Health Interoperability Resources [FHIR] Clinical Reasoning)  
K044. Reporting requirements  
K045. Methods to measure and report organizational performance  
K046. Adoption metrics (e.g., Electronic Medical Records Adoption Model [EMRAM], Adoption Model for Analytics Maturity [AMAM])  
K047. Social determinants of health  
K048. Use of patient-generated data  
K049. Prediction models  
K050. Risk stratification and adjustment  
K051. Concepts and tools for care coordination  
K052. Care delivery and payment models

## Domain 3: Enterprise Information Systems

K053. Health information technology landscape (e.g., innovation strategies, emerging technologies)  
K054. Institutional governance of clinical information systems  
K055. Information system maintenance requirements  
K056. Information needs analysis and information system selection  
K057. Information system implementation procedures  
K058. Information system evaluation techniques and methods  
K059. Information system and integration testing techniques and methodologies  
K060. Enterprise architecture (databases, storage, application, interface engine)  
K061. Methods of communication between various software components  
K062. Network communications infrastructure and protocols between information systems (e.g., Transmission Control Protocol/Internet Protocol [TCP/IP], switches, routers)  
K063. Types of settings (e.g., labs, ambulatory, radiology, home) where various systems are used  
K064. Clinical system functional requirements  
K065. Models and theories of human-computer (machine) interaction (HCI)  
K066. HCI evaluation, usability engineering and testing, study design and methods  
K067. HCI design standards and design principles  
K068. Functionalities of clinical information systems (e.g., Electronic Health Records [EHR], Laboratory Information System [LIS], Picture Archiving and Communication System [PACS], Radiology Information System [RIS] vendor-neutral archive, pharmacy, revenue cycle)  
K069. Consumer-facing health informatics applications (e.g., patient portals, mobile health apps and devices, disease management, patient education, behavior modification)  
K070. User types and roles, institutional policy and access control  
K071. Clinical communication channels and best practices for use (e.g., secure messaging, closed loop communication)  
K072. Security threat assessment methods and mitigation strategies  
K073. Security standards and safeguards  
K074. Clinical impact of scheduled and unscheduled system downtimes  
K075. Information system failure modes and downtime mitigation strategies (e.g., replicated data centers, log shipping)  
K076. Approaches to knowledge repositories and their implementation and maintenance  
K077. Data storage options and their implications  
K078. Clinical registries  
K079. Health information exchanges  
K080. Patient matching strategies  
K081. Master patient index  
K082. Data reconciliation  
K083. Regulated medical devices (e.g., pumps, telemetry monitors) that may be integrated into information systems  
K084. Non-regulated medical devices (e.g., consumer devices)  
K085. Telehealth workflows and resources (e.g., software, hardware, staff)

## Domain 4: Data Governance and Data Analytics

K086. Stewardship of data  
K087. Regulations, organizations, and best practice related to data access and sharing agreements, data use, privacy, security, and portability  
K088. Metadata and data dictionaries  
K089. Data life cycle  
K090. Transactional and reporting/research databases  
K091. Techniques for the storage of disparate data types  
K092. Techniques to extract, transform, and load data  
K093. Data associated with workflow processes and clinical context  
K094. Data management and validation techniques  
K095. Standards related to storage and retrieval from specialized and emerging data sources  
K096. Types and uses of specialized and emerging data sources (e.g., imaging, bioinformatics, internet of things [IoT], patient-generated, social determinants)  
K097. Issues related to integrating emerging data sources into business and clinical decision making  
K098. Information architecture  
K099. Query tools and techniques  
K100. Flat files, relational and non-relational/NoSQL database structures, distributed file systems  
K101. Definitions and appropriate use of descriptive, diagnostic, predictive, and prescriptive analytics  
K102. Analytic tools and techniques (e.g., Boolean, Bayesian, statistical/mathematical modeling)  
K103. Advanced modeling and algorithms  
K104. Artificial intelligence  
K105. Machine learning (e.g., neural networks, support vector machines, Bayesian network)  
K106. Data visualization (e.g., graphical, geospatial, 3D modeling, dashboards, heat maps)  
K107. Natural language processing  
K108. Precision medicine (customized treatment plans based on patient-specific data)  
K109. Knowledge management and archiving science  
K110. Methods for knowledge persistence and sharing  
K111. Methods and standards for data sharing across systems (e.g., health information exchanges, public health reporting)

## Domain 5: Leadership and Professionalism

K112. Environmental scanning and assessment methods and techniques  
K113. Consensus building, collaboration, and conflict management  
K114. Business plan development for informatics projects and activities (e.g., return on investment, business case analysis, pro forma projections)  
K115. Basic revenue cycle  
K116. Basic managerial/cost accounting principles and concepts  
K117. Capital and operating budgeting  
K118. Strategy formulation and evaluation  
K119. Approaches to establishing Health Information Technology (HIT) mission and objectives  
K120. Communication strategies, including one-on-one, presentation to groups, and asynchronous communication  
K121. Effective communication programs to support and sustain systems implementation  
K122. Writing effectively for various audiences and goals  
K123. Negotiation strategies, methods, and techniques  
K124. Conflict management strategies, methods, and techniques  
K125. Change management principles, models, and methods  
K126. Assessment of organizational culture and behavior change theories  
K127. Theory and methods for promoting the adoption and effective use of clinical information systems  
K128. Motivational strategies, methods, and techniques  
K129. Basic principles and practices of project management  
K130. Project management tools and techniques  
K131. Leadership principles, models, and methods  
K132. Intergenerational communication techniques  
K133. Coaching, mentoring, championing and cheerleading methods  
K134. Adult learning theories, methods, and techniques  
K135. Teaching modalities for individuals and groups  
K136. Methods to assess the effectiveness of training and competency development  
K137. Principles, models, and methods for building and managing effective interdisciplinary teams  
K138. Team productivity and effectiveness (e.g., articulating team goals, defining rules of operation, clarifying individual roles, team management, identifying and addressing challenges)  
K139. Group management processes (e.g., nominal group, consensus mapping, Delphi method)



# Knowledge Statements from the DoP

---

K006. Computer programming fundamentals and computational thinking

# Binary, Bits, and Bytes

## Computers represent data in binary

- Ex: text is encoded in ASCII, which can be represented as 8-bit binary (ISO 8859-1, or ISO Latin-1)
- Contrast ASCII/ISO 8859-1 (8-bit) to Unicode (16-bit), which allows encoding of most of the world's languages
- Because 16 is a power of 2, base-16 (hexadecimal) is a compact way of relaying binary information. An 8-bit binary ASCII-encoded character can be represented as a 2-digit HEX
  - **ASCII:** "informatics"
  - **HEX:** "69 6e 66 6f 72 6d 61 74 69 63 73"
  - **BIN:** "01101001 01101110 01100110 01101111 01110010 01101101 01100001 01101100 01101001 01100011 01110011"

**Bit** = contraction of "binary digit", also convenient for logical and algebraic information (Boolean "true"/"false", pos/neg)

**Byte** = 8 bits



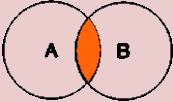
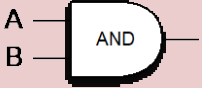
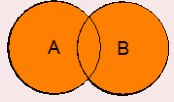
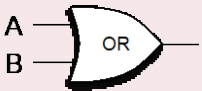
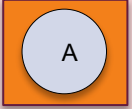
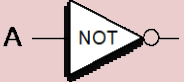
**There are 10 types of  
people in this world: those  
who understand binary,  
and those who don't.**



# From Bits to Boole

George Boole (1815-1864) was an English mathematician, logician

- Invented “Boolean algebra”, basis for digital computation

Boolean Operations	Alternate Notation(s)		Venn Diagram	Logic Gate Diagram
Conjunction (AND)	a AND b $a \bullet b$	$a \wedge b$ $a \& b$		
Disjunction (OR)	a OR b $a + b$	$a \vee b$ $a    b$		
Negation (NOT)	NOT a $\sim a$	$\neg a$ $!a$		

# Digital Logic Gates

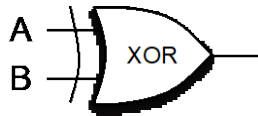
Digital circuits can mimic Boolean algebra

Logic gate diagrams show idealized versions of these circuits

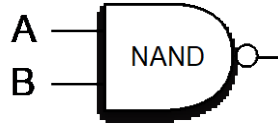
Combining these circuits allows for complex calculation

Let's look at a few more Boolean operations and logic gate diagrams

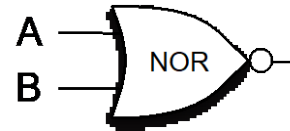
- **XOR (exclusive OR)** – returns TRUE if either A or B is true, but not both
- **NAND (NOT AND)** – returns TRUE unless both A and B are TRUE
- **NOR (NOT OR)** – returns TRUE only when both A and B are FALSE



[https://en.wikipedia.org/wiki/XOR\\_gate](https://en.wikipedia.org/wiki/XOR_gate)



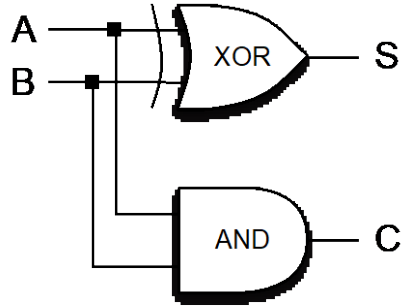
[https://en.wikipedia.org/wiki/NAND\\_gate](https://en.wikipedia.org/wiki/NAND_gate)



[https://en.wikipedia.org/wiki/NOR\\_gate](https://en.wikipedia.org/wiki/NOR_gate)



# How Computers Add Binary Numbers



A	B	C (A AND B)	S (A XOR B)	Binary Addition
1	1	1	0	1 + 1 = 10
1	0	0	1	1 + 0 = 01
0	1	0	1	0 + 1 = 01
0	0	0	0	0 + 0 = 00

- Imagine A and B are 1-bit binary values. We can model the output of this logic gate circuit with a Truth Table.
- If C (carry) and S (sum) represent the  $2^1$  and  $2^0$  positions of a 2-bit binary number, note that this series of logic gates, with one “XOR” gate and one “AND” gate, is performing a simple addition function.
- **This digital circuit is called a “half-adder”. There are, of course, more complicated versions that can perform more complicated math.**





# Transistors and Integrated Circuits

---

**Transistors**, invented in 1947, replaced vacuum tubes as the fundamental building block of electronic devices

Make use of silicon **semiconductors** to make a switch with no moving parts that can be controlled with a small current

Can be combined to build **logic gates** - a single logic gate might need 20 transistors

These days, transistors, resistors, diodes and capacitors are prepackaged onto **integrated circuits (ICs)**

Your smartphone has 100 million transistors, your PC has over a billion, and each is only 22 nanometers wide

At some point, will reach theoretical size limit. The end of Moore's law? Not quite – there are a few tricks left, like 3D instead of planar transistors

# Why More Transistors is Better

**Moore's law** predicts that the number of transistors on an integrated circuit doubles every two years, and therefore so will speed of the device.

Allows computers to do billions of calculations per second

**FLOP** is measure of computational speed = **Floating-point Operations Per Second**

A **giga-FLOP (GFLOP)** is a billion floating point operations

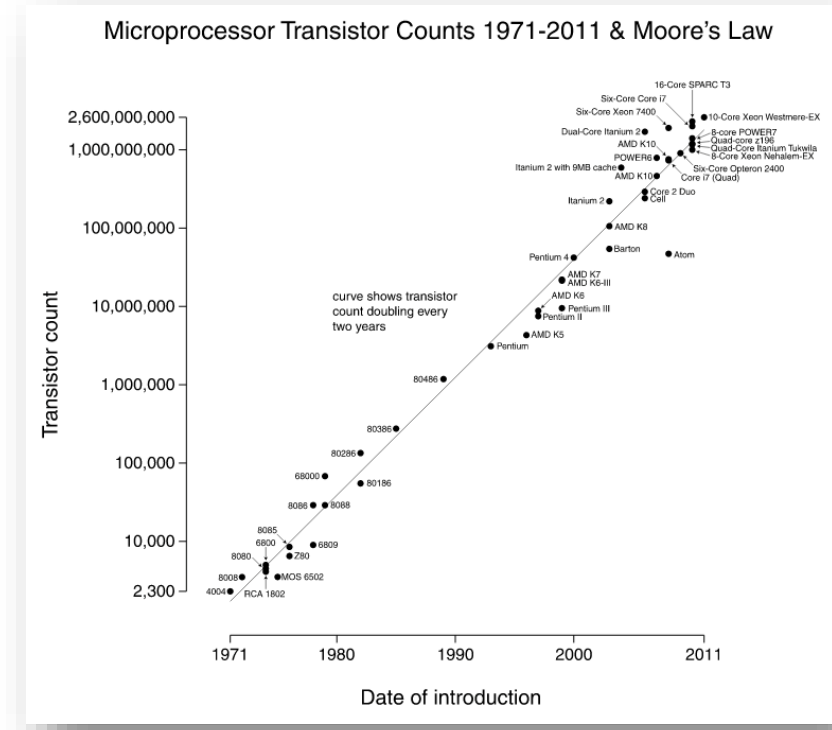


Image credit: [https://upload.wikimedia.org/wikipedia/commons/0/00/Transistor\\_Count\\_and\\_Moore%27s\\_Law](https://upload.wikimedia.org/wikipedia/commons/0/00/Transistor_Count_and_Moore%27s_Law)

# Computers Are Faster and Cheaper Than Ever

A GFLOP is 5.2 trillion-times cheaper than it was 60 years ago

Date	Approximate cost per GFLOPS	Adjusted per GFLOPS (2018 \$US)	Platform
1961	<b>\$18.7 billion</b>	<b>\$156.8 billion</b>	<b>IBM 7030 at \$7.78M each</b>
1984	\$18,750,000	\$45,220,000	Cray X-MP/48
1997	\$30,000	\$47,000	Two 16-processor Beowulfclusters with Pentium Promicroprocessors <sup>[64]</sup>
Apr-00	\$1,000	\$1,480	Bunyip Beowulf Cluster
May-00	\$640	\$944	KLAT2
Aug-03	\$82	\$112	KASY0
Aug-07	\$48	\$58	Microwulf
Mar-11	\$1.80	\$2.03	HPU4Science
Aug-12	\$0.75	\$0.82	Quad AMD Radeon 7970 GHz System
Jun-13	<b>\$0.22</b>	<b>\$0.24</b>	<b>Sony PlayStation 4</b>
Nov-13	\$0.16	\$0.17	AMD Sempron 145 & GeForce GTX 760 System
Dec-13	\$0.12	\$0.13	Pentium G550 & Radeon R9 290 System
Jan-15	\$0.08	\$0.08	Celeron G1830 & Radeon R9 295X2 System
Jun-17	\$0.06	\$0.06	AMD Ryzen 7 1700 & AMD Radeon Vega Frontier Edition
Oct-17	<b>\$0.03</b>	<b>\$0.03</b>	<b>Intel Celeron G3930 CPU + AMD RX Vega 64 GPUs (\$2000 computer that performs at 75 TFLOPS)</b>

# Low Level Programming Languages

---

## Machine Code

- Unique to the CPU (why code for SPARC, ARM, and X86 are fundamentally different)
- Series of binary digits – not intended to be human interpretable
- Ex: “10110000 01100001” or “B0 61” in hexadecimal

## Assembly language

- Shortcut mnemonics for machine language routines
- Still tedious to use, not “readable”
- Ex: “MOV AL, 61h” → instruction to move contents of a memory register from one location to another

# High Level Programming Languages

---

Written in English-like phrases

Compiled before execution (converted from source code to object code, usually assembly language or machine code)

Ex: C/C++, Java, VisualBasic, etc.

Fourth-generation languages

- Further abstraction of complex tasks
- May automate report generation, interaction with a database
- Often are very task or domain-specific
  - Ex: Mathematica, SPSS

# Programming Paradigms

---

Distinct styles of programming that influence performance, maintainability, length of code, memory requirements, etc.

Key examples:

- Imperative / Procedural
- Object-Oriented

# Imperative and Procedural Programming

---

Ex: FORTRAN, BASIC, Pascal, C, C++

Programs are a list of tasks, subroutines

Like a recipe: each step is a sequenced instruction

Procedural languages allow programmer to define functions, subroutines (procedures) and reuse throughout the program

# Object-Oriented Programming

---

Ex: Java, Objective-C, VisualBasic.NET

Programs are a collection of interacting objects

**Objects** can have independent functions, characteristics, and states

A **Class** is a “blueprint” for an object – it describes the functions and characteristics that all objects in that class share in common

- Ex: “the class of all dogs”

A specific member of a class is an **Instance**

- Ex: “a beagle is an instance of a dog”



# OOP Terminology

---

Objects have **Attributes** (adjectives) and **Methods** (verbs)

**Instantiation** involves creating a new object and setting initial parameters for the **Attributes** and **Methods**

Ex: the class of “Dog” may have attributes for “fur color”, “name”, “breed” and methods for “bark” and “roll over”

Classes can demonstrate **Encapsulation**: can keep their attributes and methods private; control access to other parts of the program; either allow or restrict external invocation / modification

# OOP Terminology

---

## Composition

- Objects can be composed from smaller objects
- Ex: if you have a class that defines a “Point” as an X,Y coordinate, you can create a class for “Line” by reusing two “Point” objects

## Inheritance

- Objects can inherit their structure from parent objects and extend their functionality
- Ex: Two classes for “Manager” and “Staff” may inherit the same basic structure of a parent class “Employee”

# OOP Terminology

---

## Polymorphism

- Objects can override their parent attributes/methods
- Details of the subclass implementation will determine which attribute/method is invoked
- Ex: class “Animal” may have a “makeNoise” method. SubClasses of “Animal” (Dog, Cat, Mouse) will all have a “makeNoise” method, but may implement it differently and can override the parent method

**Accessors** (“getters”) are methods used to retrieve variable state

**Mutators** (“setters”) are methods used to change variable state

# OOP Class Example (Java)

```
public class Dog extends Mammal
{
    private String name;
    private String furColor;
    private String breed;
    private String noise = "Woof!";

    public void setName(string n)
        {name = n;}
    public void setFurColor(string f)
        {furColor = f;}
    public void setBreed(string b)
        {breed = b;}
```

```
//continued
    Public String getName()
        {return name;}
    Public String getFurColor()
        {return furColor;}
    public void bark();
        {System.out.println(noise);}
}
```

- **Inheritance:** Dog extends Mammal class
- **Encapsulation:** no way to see or modify name, furColor, breed without using “get” and “set” methods
- **Composition:** Dog uses “String” class and “System.out” class
- **Polymorphism:** Subclasses of Dog do not have to say “Woof!”

# OOP Class Example (Java)

```
public static void main(String[] args) {  
  
    Dog myDog = new Dog();  
  
    myDog.setName("Amia");  
  
    myDog.setFurColor("brown");  
  
  
    System.out.println(  
  
        "My dog's name is "+myDog.getName());  
  
  
    myDog.bark();  
  
}
```

- **Instantiation** – create a new object of the class "Dog"
- **Mutators** – set value of String "Name" using the setName() method
- **Accessors** – get value of String "Name" using the getName() method

# Types of Qualitative Data

---

## Nominal aka “Categorical”

- Think “data with names”
- Mutually exclusive, unordered, discrete categories of data, such as patient smoking status
- **Mode** (most frequently occurring value) is the only measure of central tendency

## Ordinal

- Think “ordered data”
- Data that have a natural ordering
- Ex: “Small”, “Medium”, “Large”
- Ex: Asthma severity (“Intermittent”, “Mild Persistent”, “Moderate Persistent”, “Severe Persistent”)
- Ex: Likert scales of patient satisfaction, Pain scales
- **Median** (middle-ranked value) and **Mode** can be used to measure central tendency



# Types of Quantitative Data

---

## Interval

- Data where the intervals between values represent the same distance
- Example: Year, Temperature. The difference between 32° F and 33° F is the same as the difference between 76° F and 77° F
- **Arithmetic mean, median, and mode** can be used as measures of central tendency

## Ratio

- Allows for additional comparison because “zero” means something and is not arbitrarily chosen
- Ex: Area, distance. Temperature in Kelvin is a ratio (because 300K has twice as much heat as 150K), but temperature in Fahrenheit is not (because zero Fahrenheit does not mean zero heat)
- **Geometric mean** ( $n$ th root of the product of  $n$  values), **arithmetic mean, median and mode** are allowed



# Data Transformations

---

## Interval → Ratio

- Your application stores a value for “year diagnosed with cancer” – that’s an **Interval**
- In your software, you convert this to “years since diagnosis of cancer” – that’s a **Ratio**

## Interval → Interval or Ratio → Ratio

- Example: Image processing, scaling an image: take every 2x2 cluster of pixels and calculate the average color value, assign it to a single 1x1 pixel.

## Interval → Ordinal

- Can group ranges of variable using “**Binning**” techniques, commonly used to smooth effects of minor observation errors
- Take a continuous variable like “age in years”, group them into categories such a “Neonate”, “Infant”, “Toddler”, “Child”, “Adolescent”, “Adult” (implied order)

**Be aware what you gain or lose when you transform data.**

**When you compress image / audio files by downsampling, you can lose data.**

**When you bin continuous data into ordinal buckets, you lose data.**





# Data Structures

---

## Primitives

- BOOLEAN (TRUE / FALSE)
- CHAR (character string of predetermined length)) and VARCHAR (variable length character string)
- FLOAT / DOUBLE (Floating-point number is a representation of a number with decimals / exponents, up to 32 bits long; Double is up to 64bits long)
- INT (Integer)

## Array & Composite structures (e.g. multidimensional arrays)

- Array is a collection of elements identified by position / key / index  

```
var presidents = ["Washington", "Adams", "Jefferson", "Madison", "Monroe"]  
presidents.length → 5  
presidents[0] → "Washington"
```

## Strings

### Date

- most programming languages have a special class for dates
- most RDBMS have a special data type for dates
- Could store as a string, but that could impact reuse
- Ex: Oracle SQL – define a column of type VARCHAR and store value "01/13/12". How would you perform math on this value?

### Time

**[“hip”, “hip”]**



# Control Structures

---

## Primitive

- Labels (line numbers)
- GOTO statements – instructs program to “jump” to another label
- Subroutine – GOSUB label → [do some stuff] → RETURN

## Choice

- IF – THEN – ELSE  
IF (age > 12 && sex = female) THEN (show pregnancy risk warning) ELSE (do nothing)
- CASE – WHEN statements

CASE

WHEN (age < 8) → (recommend amoxicillin instead of doxycycline)

WHEN (age >= 8 && age < 60) → (recommend doxycycline)

WHEN (age >= 60) → (recommend cephalosporin)

END



# Control Structures

---

## Loop

- **Count-controlled loop with iterator (FOR – NEXT)**

```
FOR (i=0; i<20; i=i+1){ DO SOMETHING; }
```

- **Condition-controlled loops (WHILE)**

```
WHILE (i<20) {  
    DO SOMETHING;  
    i=i+1;  
}
```

- **Collection-controlled loops (FOR EACH IN...)**

```
var Patients=["John","Martha","Alice","Carlos"];  
for(var i in Patients){  
    DO SOMETHING;  
}
```

# Apple Basic Example

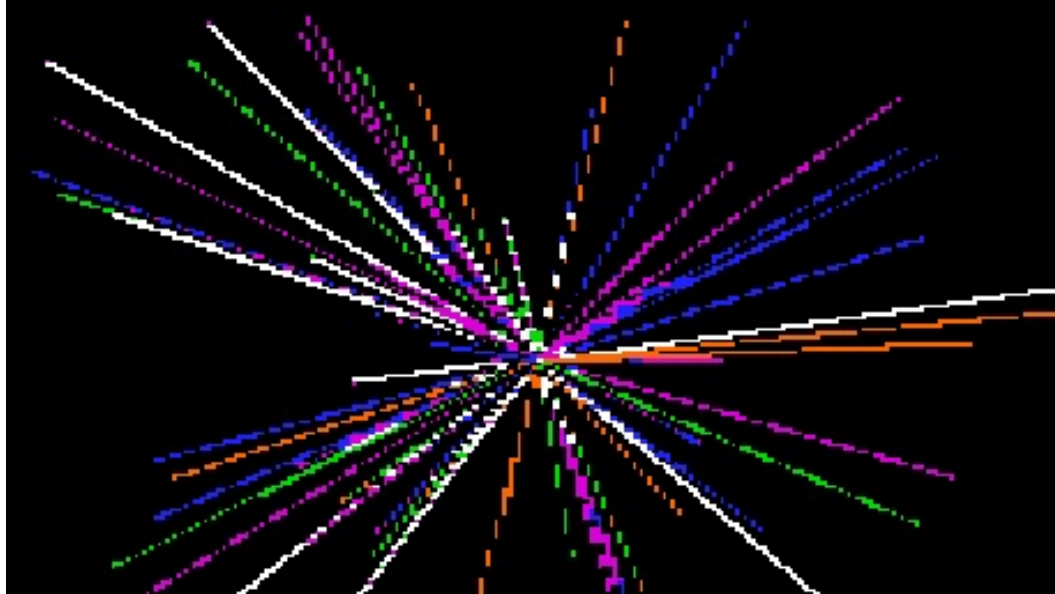
```
10  FOR T = 1 TO 10
20  HOME : HGR
30  FOR N = 1 TO 100
40  X = INT (280 * RND (1))
50  Y = INT (192 * RND (1))
60  C = INT (7 * RND (1))
70  HCOLOR= C
80  HPLOT (280 / 2),(192 / 2) TO
   X,Y
90  NEXT N
100 NEXT T
110 TEXT : HOME
```

- Set a variable “T” and iterate it from 1 to 10, start the loop at T=1
- Clear the screen, change to high resolution graphics mode (“HGR”)
- Set a variable “N” and iterate it from 1 to 100, start at N=1
- Define variables X, Y, and C using the random number function (“RND(1)”)
- Set the pen color to the value of variable C
- Plot a line from the center of the screen to the coordinates X,Y
- Increment N by 1 and repeat steps 40 through 80 until N = 100
- Increment T by 1 and repeat steps 20 through 90 until T=10
- Exit HGR mode, go back to TEXT mode, and clear the screen

Credit: “Programming in AppleSoft Basic”, <http://www.evaluation.com/>

# Apple Basic Example

---



**So, this program will draw a random starburst pattern of “N” lines a total of “T” times**

Credit: “Programming in AppleSoft Basic”, <http://www.evaughan.com>

# Pseudocode & Trace Tables

---

The last example was easy enough that we could “do it in our heads”

But often, you want to keep track of variable assignments, loops, and operators.

You can use **pseudocode** for this and check your code using **trace tables**

Pseudocode is not a real programming language (but there is a “standard”)

You use normal English expressions to represent common computer functions

- Variable assignment
- Control / loop structures
- Logic functions
- Inputs / Outputs / Database functions

Goal is to represent an algorithm in a way that can be implemented in any computer program, simply to convey program flow and logic

It's also a way to check algorithms before you start coding



# Pseudocode & Trace Tables

```
1  x = 10
2  y = 2
3  print x+y
4  print x/y
5  for i = 1 to 3
6  x = x-1
7  print x
8  next i
9  print "I love AMIA!"
```

	line	x	y	i	output
	1				
	2				
	3				
	4				
	5				
loop	6				
	7				
	8				
loop	6				
	7				
	8				
loop	6				
	7				
	9				





# Pseudocode and Trace Tables

```
1  x = 10
2  y = 2
3  print x+y
4  print x/y
5  for i = 1 to 3
6  x = x-1
7  print x
8  next i
9  print "I love AMIA!"
```

	line	x	y	i	output
	1	10			
	2	10	2		
	3	10	2		12
	4	10	2		5
	5	10	2	1	
loop	6	9	2	1	
	7	9	2	1	9
	8	9	2	2	
loop	6	8	2	2	
	7	8	2	2	8
	8	8	2	3	
loop	6	7	2	3	
	7	7	2	3	7
	9	7	2		I love AMIA!





# Class Exercise

```
// pseudocode example
01 let n = 2
02 while (n < 10) do the following:
03     t= "true"
04     let i = 2; for every number i < n:
05         if (n mod i) is not 0 then increase i by 1
06         else t = "false" and then exit for loop
07     if t is still "true" then print n
08 increase n by 1
```

•The “**modulo**” function returns the remainder when you divide two numbers.

•Think “clock math”.

- Ex:  $23 \bmod 12 = 11$  (“2300 = 11pm”)
- Ex:  $17 \bmod 12 = 5$  (“1700 = 5pm”)
- Ex:  $13 \bmod 5 = 3$  ( $13 \div 5$  has remainder 3)

Let’s make a trace table for the program on the left...

## Class Questions:

- How many numbers will it output?
- What does this program do?
- What do the output numbers have in common?





# Class Exercise

```
// pseudocode example
```

```
01 let n = 2
```

```
02 while (n < 10) do the following:
```

```
03   t= "true"
```

```
04   let i = 2; FOR every number where i < n:
```

```
05     if (n mod i) is not 0 then increase i by 1
```

```
06     else t = "false" and then exit FOR loop
```

```
07   if t is still "true" then print n
```

```
08 increase n by 1
```

\*Note that "t" and "i" get reset in each loop

\*Note that "x mod y = 0" means x is evenly divisible by y

n	n<10?	i	i<n?	n mod i	t	Output
2	yes				true	
2	yes	2	no		true	2
3	yes	2	yes	1	true	
3	yes	3	no		true	3
4	yes	2	yes	0	false	
5	yes	2	yes	1	true	
5	yes	3	yes	2	true	
5	yes	4	yes	1	true	
5	yes	5	no		true	5
6	yes	2	yes	0	false	
7	yes	2	yes	1	true	
7	yes	3	yes	1	true	
7	yes	4	yes	3	true	
7	yes	5	yes	2	true	
7	yes	6	yes	1	true	
7	yes	7	no		true	7
8	yes	2	yes	0	false	
9	yes	2	yes	1	true	
9	yes	3	yes	0	false	



# Supplement: Javascript Prime Number Tester

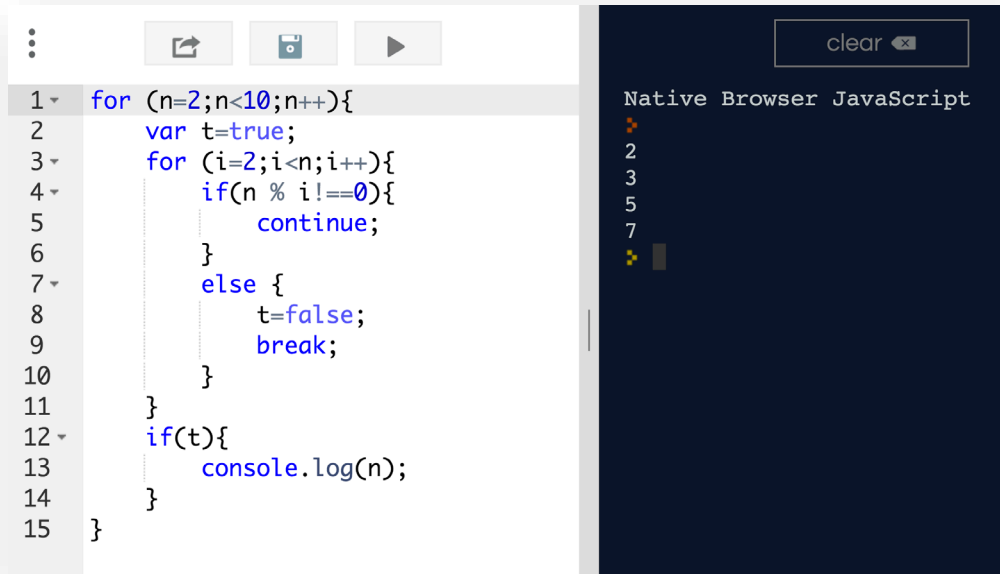
Here's an implementation of the same algorithm in **Javascript**.

The website <http://repl.it/> features a browser-based Javascript console where you can test code.

The left half of the screen is your code. When you press the “play” triangle icon, you can run the program in a virtual console on the right.

Note that the output is the same as our trace table (the numbers 2,3,5, and 7 – all prime numbers less than 10).

Also note the use of “for” loops instead of “while” loops.



```
1 for (n=2;n<10;n++){
2   var t=true;
3   for (i=2;i<n;i++){
4     if(n % i!==0){
5       continue;
6     }
7     else {
8       t=false;
9       break;
10    }
11  }
12  if(t){
13    console.log(n);
14  }
15 }
```

Native Browser JavaScript

2  
3  
5  
7

Bonus question: what line would you change to print all prime numbers less than 100?



# Supplement: Javascript Prime Number Tester

```
1  for (n=2;n<10;n++){
2      var t=true;
3      for (i=2;i<n;i++){
4          if(n % i!==0){
5              continue;
6          }
7          else {
8              t=false;
9              break;
10         }
11     }
12     if(t){
13         console.log(n);
14     }
15 }
```

Some hints to understand this Javascript snippet:

- Best practice to use indentation as a visual cue re: nested loops and controls
- Open/Close brackets must match – use this to tell where a loop ends if you get lost.
- “for...next” loops follow this syntax:  
**for (start; until; increment)**
- “++” is the increment operation → shorthand for “i = i+1”
- New variable declarations start with the word “**var**”
- The percent sign “%” is the **modulo operator**
- “!” means “not”, and “==” is a **test of equality**
  - So “**!=0**” means “is not equal to zero”.
- In contrast, “=” is for **assignment of a value to a variable**
  - So “**t=true**” means the variable t is assigned value “true”
- “**continue**” and “**break**” allow you to iterate within or exit out of a “for” loop
- The variable “t” is a Boolean → so “if(t)” is a Boolean operation



# Programming questions on the exam?

---

## Likely to be Tested

Interpreting a pseudocode algorithm →  
predicting output

Understanding control structures, loops,  
operators, variable assignment

Completing a **truth table**

Completing a **trace table**

## Not likely to be tested

You are not likely to be tested on a specific computer programming language except SQL (in the database lecture)

If you are tested on a real language, the syntax will probably be very “human interpretable” – so treat it just like pseudocode

**A programmer goes to the grocery store.**

**His wife says, “buy a gallon of milk, and if they have eggs, buy a dozen”.**

**He comes home with 13 gallons of milk.**



**There are two types of people in the world:**

**1) Those who can extrapolate from incomplete data**





# Hash Functions

---

Algorithm that maps data of arbitrary length to data of fixed length, an example of “binning”

Returned value is known as “hash value”, “hash code”, “hash sum”, or “checksum”

Uses in data integrity:

- Checking that a file was not tampered with or downloaded incompletely (“MD5 Hash”)
- Checking that a keystroke error did not occur (National Provider Identifier and Luhn Algorithm, where last digit is a checksum)

# Hash Functions

---

```
md5("The patient has evidence of hypertrophy") =  
"65dc36d7011f96c757ca96e87bd0ff9e"
```

```
md5("the patient has evidence of hypertrophy") =  
"611a64303cffc9232c05022500fe4b9a"
```

## Uses in Cryptography

- Websites should not store unencrypted passwords
- Rule of thumb: if you click 'forgot password' on a website and they email you the password, they're probably storing them unencrypted, which is a no-no

## Uses in Indexing (as in databases)

- Takes text strings and, via a hash, places them into "buckets"
- Retrieving info is faster since program only has to look in that bucket



# Software Development Methodologies

---

Aka Software Development Life Cycle (SDLC)

Need for standard approach to

- Determine scope
- Organize programming tasks
- Determine testing requirements
- Manage resources & time commitments
- Deliver software on a reproducible schedule

# Basic Phases of SDLC

---

## Planning

- Gather requirements, determine scope and priority of work

## Software Implementation (what we often refer to as “Software Development or Design”)

- Actual development process

## Testing – Verification vs. Validation

- **Verification** - “are you building the software right?” (does it meet specification, is the code high quality, defect-free, etc.)
- **Validation** - “are you building the right software?” (does it meet customer’s expectation, satisfy their needs, do what it’s supposed to)

## Documentation

- critical for future enhancement, debugging, maintainability

## Deployment (what we often refer to as “Implementation”)

- Install, customize, train, evaluate

## Maintenance

- Process to collect new defects or enhancements, support users in ongoing fashion

# SDLC Methodologies: Waterfall

---

Move to next phase only when prior phase is done

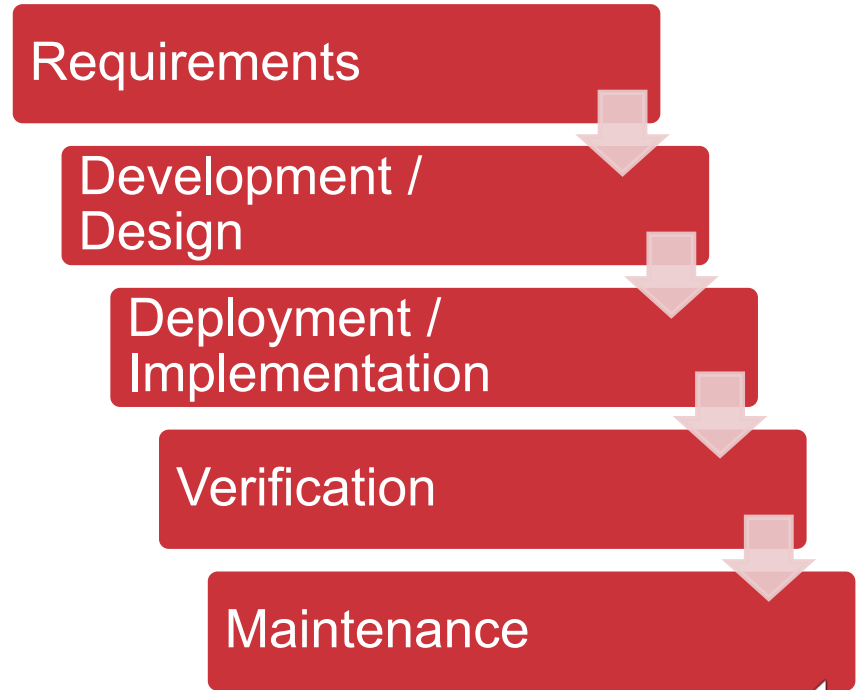
Option to revisit decisions, but usually go through a formal change control process

Highly structured, relatively inflexible

Good for projects with stable requirements

**Advantages:** defects are found sooner, when they're less costly to fix

**Disadvantage:** a late-breaking requirement can be expensive or prohibitive



# SDLC Methodologies: Spiral

---

“Risk oriented” software development

- One example of risk is poorly understood requirements

Development broken into smaller efforts

Each subproject designed to tackle an area of high-risk

Traditional phases

- Determine objectives, alternatives, constraints
- Identify and resolve risks
- Evaluate alternatives
- Develop deliverables for a given iteration and verify they are correct
- Plan the next iteration
- Commit to an approach for subsequent iteration

**Advantages:** highest risk is tackled early on, when change is less expensive



# SDLC Methodologies: Agile

---

High focus on very small steps, frequent loops

Extreme Programming (XP) and Scrum are two popular Agile variations

Feedback provides regular testing and release of software

# SDLC Methodologies: Agile

---

## Scrum Terminology

- **Product Owner** – person who represents client requirements. Writes user stories (brief statement that captures “who”, “what”, “why” of a simple requirement) and adds to backlog.
- **Development team** – 3 - 9 programmers who are cross-functional (can tradeoff coding, testing, documentation)
- **Scrum Master** – rule enforcer and remover of impediments
- **Sprint** – basic unit of development, predefined in duration and scope, chosen from list of backlog
- **Daily Scrum** – what happened yesterday? What will happen today? Any obstacles?

**Advantage:** Extremely flexible; assumes that clients will change requirements and is equipped to adapt to unpredicted challenges



# System Integration

---

Process of linking subsystems in software architecture

- Requires combination of software, hardware, interface skills

Facilitated by strong understanding of standards

- HL7, Web Services, Networking

Methods

- Vertical Integration
  - Group systems by function into silos
  - Integrate within a silo, but not necessarily between
- Star Integration
  - Unique connection to each system that requires it
  - High overhead for complex systems
- Horizontal Integration (via Enterprise Service Bus)
  - One connection per system to ESB. ESB handles downstream connections
  - Replacing a single system is much easier



# Quality Assurance / Testing

Goal: to mitigate project risk as early as possible, release good products

Cost of fixing a defect is a function of project phase

Cost to fix a defect		Time detected				
		Requirements	Architecture	Construction	System test	Post-release
Phase of Defect	Requirements	1x	3x	5–10x	10x	10–100x
	Architecture	-	1x	10x	15x	25–100x
	Construction	-	-	1x	10x	10–25x

# THERAC-25

---

Radiation therapy machine

Responsible for 6 accidents, 3 deaths due to massive radiation overdose from 1985-1987

Attributed to failure to detect a “race condition” in software

Poor design of error messages (“MALFUNCTION” followed by a number from 1 to 64)

Personnel didn’t believe patient’s complaints

PATIENT NAME : JOHN DOE

TREATMENT MODE : FIX BEAM TYPE: X ENERGY (MeV): 25

	ACTUAL	PRESCRIBED
UNIT RATE/MINUTE	0	200
MONITOR UNITS	50 50	200
TIME (MIN)	0.27	1.00

GANTRY ROTATION (DEG)	0.0	0	VERIFIED
<b>COLLIMATOR</b> ROTATION (DEG)	359.2	359	VERIFIED
COLLIMATOR X (CM)	14.2	14.3	VERIFIED
COLLIMATOR Y (CM)	27.2	27.3	VERIFIED
WEDGE NUMBER	1	1	VERIFIED
ACCESSORY NUMBER	0	0	VERIFIED

DATE : 84-OCT-26	SYSTEM : BEAM READY	OP.MODE: TREAT AUTO
TIME : 12:55. 8	TREAT : TREAT PAUSE	X-RAY 173777
OPR ID : T25VO2-RO3	REASON : OPERATOR	COMMAND:



# Testing Approaches

---

Static testing – reviewing code itself

Dynamic testing – apply test cases to code

Test case development approaches

- **White Box** – tests inner workings of a program (ex: was an order sent/received correctly between systems)
  - **Code Coverage** is an example of White Box testing – programmer will develop a set of test conditions for all scenarios, all variables, all possible inputs with awareness of the internal design of the system.
- **Black Box** – tests functionality from end-user standpoint (ex: if user enters an order, does EHR display the order is active and signed).
  - Ex: **specification testing**, which may be insufficient to capture all defects

# Levels of Testing

---

## Unit Test

- white box
- code coverage
- developer driven, can be semi-automated

## Integration Test

- white box
- interfaces/API's
- developer-driven

## System Test

- Black box
- Test against documented requirements (verification)

## Acceptance Test

- Black box
- Request for user sign-off (validation)
- Ex: "beta testing", solicit input from a small group of users

# Regression Testing

---

Regression is a new defect revealed with the addition of a new functionality

Ex: you add a new feature to a program, now an older feature stops working as intended – that's a regression

Regression testing: when you add new functionality, go-back and repeat all prior tests, retest all previously-reported and fixed defects

# Optional Pseudocode Exercise: Translating a Clinical Guideline

---

How good is your pseudocode? Informaticists are often asked to translate guidelines into “computable” logic rules (ex: for an EHR alert)

For this self-directed exercise, your task is to write a pseudocode representation of the **MD Anderson Cervical Cancer Screening Guideline**, which you can find here: <https://www.mdanderson.org/prevention-screening/get-screened/cervical-cancer-screening.html>

Make sure your alert:

- 1) Returns the correct recommendation based on patient age
- 2) Takes into account PAP status
- 3) Takes into account HPV status



# Example Pseudocode

(graciously provided by a former CIBRC course attendee!)

---

Extract sex ( =Male, or =female)

If sex=female, then

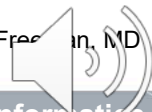
- If age >65, and extract (history of hysterectomy, benign), output “HPV and cervical cancer screening not recommended”)
- If age <21, output “HPV and cervical cancer screening not recommended”
- If age >=21 and <=29, output “Liquid Pap screening is preferred every 3 years.”
- If age >30 and <65, output “Liquid Pap screening is preferred and HPV for high risk screening”

Extract “Pap results” as == (defined as) normal or == abnormal,

Extract HPV results, as == normal or == abnormal, and

- if Pap and/or HPV == abnormal, output “Consult MD Anderson guideline for abnormal Pap/HPV results”
- If Pap results == normal, and HPV not performed, output “Repeat Pap screening and/or HPV screening every 3 years
- If Pap and HPV normal, output “repeat every 5 years”

Used with permission: William David Freeman, MD



---

# End of Lecture

# Suggested Additional Reading

---

[https://en.wikipedia.org/wiki/Control\\_flow](https://en.wikipedia.org/wiki/Control_flow)

[https://en.wikipedia.org/wiki/Comparison\\_of\\_programming\\_paradigms](https://en.wikipedia.org/wiki/Comparison_of_programming_paradigms)

[https://en.wikipedia.org/wiki/Software\\_development\\_process](https://en.wikipedia.org/wiki/Software_development_process)

[https://en.wikipedia.org/wiki/Software\\_testing](https://en.wikipedia.org/wiki/Software_testing)

McConnell S. Rapid Development: Taming Wild Software Schedules. Redmond, WA: Microsoft Press, 1996. 600p.

# Self-Directed Learning

---

“Standard” pseudocode:

[https://www.cs.princeton.edu/courses/archive/spr10/cos116/handouts/Pseudocode\\_Reference.pdf](https://www.cs.princeton.edu/courses/archive/spr10/cos116/handouts/Pseudocode_Reference.pdf)

Repl.it browser-based Javascript console: <https://repl.it/languages/JavaScript>

Javascript tutorial

- <https://www.w3schools.com/js/default.asp>
- Sections of interest:
  - Syntax, Statements, Variables, Operators, Arithmetic, Assignment, Data Types, Functions, Objects, Scope, Strings, Numbers, Arrays, Booleans, Comparisons, Conditions, Switch, Loop For / While

YouTube pseudocode tutorial

- <https://www.youtube.com/watch?v=Rg-f07rDsds>

TraceTable example on YouTube

- <https://www.youtube.com/watch?v=UhziRgmVKuQ>