



AP Exam Preparation

AP Exam

When: Monday June 7, 2021 at 4PM Eastern Standard Time.

At home. On Digital Testing App.

Preparing:

- 1) Download! <https://download.app.collegeboard.org/>
- 2) Practice with Example Questions in the Digital Testing App.
- 3) **1–3 Days Before Exam Day**, Complete Exam Setup for Each Digital Exam. **You cannot take the exam without completing this step!**
- 4) On Exam Day: 30 Minutes Before the Exam | Check In to the Exam

On exam day, you must check in **30 minutes before the official start time of the exam**—at 11:30 a.m. EDT for 12 p.m. exams and 3:30 p.m. EDT for 4 p.m. exams—to complete final pre-exam checks.

AP Exam

Section I: End-of-Course Multiple-Choice Exam

70 Multiple-Choice Questions | 120 Minutes | 70% of Score | 4 answer options

- 57 single-select multiple-choice
- 5 single-select with reading passage about a computing innovation
- 8 multiple-select multiple-choice: select 2 answers
- Note: On the digital exam, there are 59 single-select multiple-choice questions, 5 single-select multiple-choice questions with reading passage, and 6 multiple-select multiple-choice questions.

Section II: Create Performance Task

30% of Score

Topics

The AP Multiple Choice end-of-year covers 5 Big Ideas:

- 1) Creative Development(10%-13%)
- 2) Data(17%-22%)
- 3) Algorithms and Programming(30%-35%)
- 4) Computer Systems and Networks(11%-15%)
- 5) Impact of Computing(21%-26%)

AP Exam Review

Up to this point, I have been teaching Computer Science principles rather than teaching to the AP test.

This lecture slides is teaching to the AP test. It will provide review material for the AP test that we might not have covered during the course of the year.

We will go over each of the 5 ideas in summary.

Big Idea I

1.1 Collaboration

1.C Explain how collaboration affects the development of a solution.

6.A Collaborate in the development of solutions (*not assessed*).

1.2 Program Function and Purpose

1.A Investigate the situation, context, or task.

3.A Generalize data sources through variables.

4.A Explain how a code segment or program functions.

1.3 Program Design and Development

1.B Determine and design an appropriate method or approach to achieve the purpose.

1.C Explain how collaboration affects the development of a solution.

4.A Explain how a code segment or program functions.

6.C Acknowledge the intellectual property of others (*not assessed*).

1.4 Identifying and Correcting Errors

1.B Determine and design an appropriate method or approach to achieve the purpose.

4.C Identify and correct errors in algorithms and programs, including error discovery through testing.

Big Idea I

Please read Chapter 2: Big Idea I: Creative Development. of the AP Barron's book. The following slides provide a summary of the material covered in this chapter.

A **computing innovation** includes a program as an integral part of its function.

A computing innovation can be physical (e.g., self-driving car), nonphysical computing software (e.g., picture editing software), or a nonphysical computing concept (e.g., e-commerce).

Hardware is the physical components of a computing device, while software is the instructions in a programming language to the computing device. A computing innovation can have hardware components. However, the computing innovation is about the software, not the hardware.

Big Idea 1

Computing hardware has gotten smaller and more powerful over the years. Moore's law predicts that the size of transistors halves every two years while the cost also halves every two years. Computers went from taking up 1,800 square feet and weighing almost 50 tons to being able to fit in your pocket.

| Software | Hardware |
|--|------------------|
| Operating systems | Motherboard |
| Driverless vehicle software to avoid crashes | Self-driving car |
| Dual-monitor programs for Windows | Monitor |
| Compiler | Transistor |

Big Idea I

Collaboration helps people learn from each other. Collaboration that includes diverse perspectives helps to avoid bias in the development of computing innovations.

For example, if females play video games at the same percentage as males, a game company might not avoid bias if it employed males to write the code for the games. Bringing in female coders could bring additional perspectives that might not have been achieved otherwise.

Programming companies often hire people who not only are good programmers but also have interpersonal skills needed to collaborate effectively. Effective collaboration can help one gain insight and knowledge by applying multiple perspectives, experiences, and skill sets.

Big Idea I

Collaboration is a learned skill. That skill includes but is not limited to:

- Communication
- Consensus building
- Conflict resolution
- Negotiation

Collaboration with others can make the programmer more self-aware.

Group programming can match up your weaknesses with someone else's strengths, which results in a better product and leads to insight and knowledge not obtainable when working alone.

Big Idea I

Collaboration is not limited by location. Current computing tools allow people in different physical locations to share data.

Online collaboration tools, such as Google Docs, Zoom, Slack, Yammer, and—by the time you read this—dozens of other tools, allow programmers to collaborate from home or from anywhere that has internet access.

A **program** is a collection of program statements that performs a specific task when run by a computer. A program is often referred to as software.

A **code segment** refers to a collection of program statements that are part of a program.

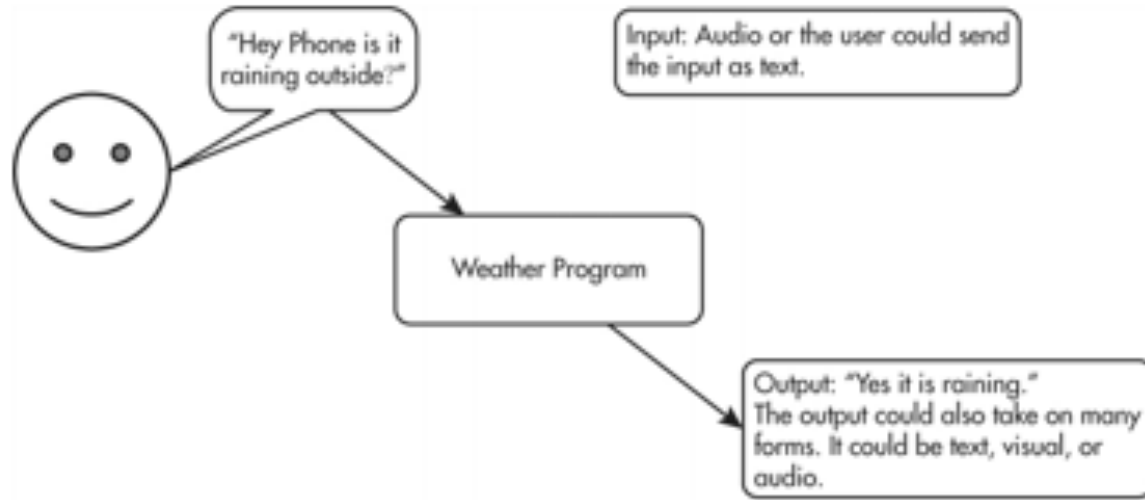
Big Idea I

Program input is data sent to a computer for processing by a program. Input can come in a variety of forms, such as tactile, audio, visual, or text. For example, a cell phone can convert voice (audio) to text to send a message.

A weather program on your phone could take input in many forms. This weather app was triggered by the user saying (audio) “Hey Phone...,” which would be an example of audio input.

This triggering is called an **event**. The event is the action that supplies input data to a program. Events can be generated when a key is pressed, a mouse is clicked, a program is started, or by any other defined action that affects the flow of execution.

Big Idea 1



Example of input/output

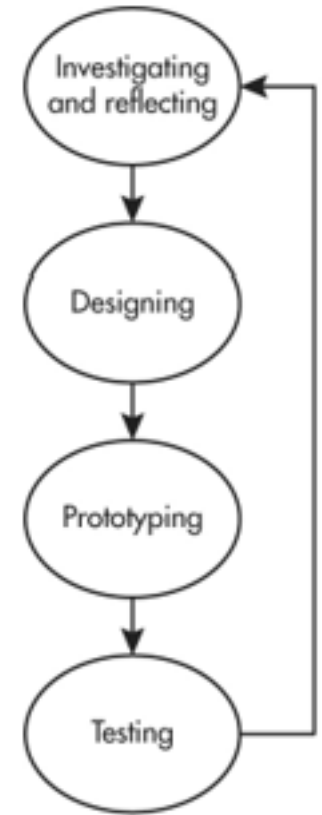
Program outputs are any data sent from a program to a device. Program output can come in a variety of forms, such as tactile, audio, visual, or text. Program output is usually based on a program's input or prior state (e.g., internal values).

Big Idea 1

A **development process** can be ordered and intentional, or exploratory in nature.

A development process that is **incremental** is one that breaks the problem into smaller pieces and makes sure each piece works before adding it to the whole.

A development process that is **iterative** requires refinement and revision based on feedback, testing, or reflection throughout the process. This may require revisiting earlier phases of the process.



Big Idea I

The design of a program incorporates investigations to determine its requirements. Most programs are designed to be used by people other than the programmers.

To meet the needs of the users, the investigation must identify the program constraints as well as the concerns and interests of people who will use the program.

Some ways investigations can be performed are as follows:

- Collecting data through surveys
- User testing
- Interviews
- Direct observations

Big Idea I

The design phase of a program may include:

- Brainstorming
- Planning and storyboarding
- Organizing the program into modules and functional components
- Creating diagrams that represent the layouts of the user interface
- Developing a testing strategy for the program

Program documentation is a written description of the function of a code segment, event, procedure, or program and how it was developed.

Program documentation helps in developing and maintaining correct programs when working individually or in collaborative programming environments.

Big Idea I

Programmers should document a program throughout its development.

Documentation helps the programmer remember what he or she was thinking or the collaborative partners were thinking at the time they were programming.

Comments are a form of program documentation written into the program that do not affect how the program runs. Comments do not affect the run speed of a program. Python, for example, uses `#` for comments.

Big Idea I

Three types of program errors can occur:

■ **Logic error**—This is a mistake in the algorithm or program that causes it to behave incorrectly or unexpectedly. (incorrect implementation of algorithm)

■ **Syntax error**—This is a mistake in the program where the rules of the programming language are not followed. (missing parenthesis, incorrect indentation, misspelling name of function calls)

■ **Runtime error**—This is a mistake in the program that occurs during the execution of a program. Programming languages define their own runtime errors. (divide by 0, accessing out-of-bounds index of a list)

Big Idea 1

Read the code below. Assume that `myList` is a nonempty list of numbers. Identify the error. What kind of error is it?

```
Line 1: Procedure getTotal(myList)
Line 2: {
Line 3:   total ← myList[1]
Line 3:   FOR EACH item IN myList
Line 4:   {
Line 5:     total ← total + item
Line 6:   }
Line 7: RETURN(total)
Line 8: }
```

Logic Error. The code adds the first number twice in the sum.

Big Idea 2: Data

2.1 Binary Numbers

1.D Evaluate solution options.

2.B Implement and apply an algorithm.

3.C Explain how abstraction manages complexity.

2.2 Data Compression

1.D Evaluate solution options.

2.3 Extracting Information from Data

5.B Explain how knowledge can be generated from data.

5.D Describe the impact of gathering data.

2.4 Using Programs with Data

2.B Implement and apply an algorithm.

5.B Explain how knowledge can be generated from data.

Big Idea 2: Data

A bit is shorthand for a single binary digit and is either 0 or 1. A byte is 8 bits. For example, the binary sequence 01101111 contains 8 bits or 1 byte.

Binary sequences can be used to represent all digital data. Binary sequences can represent colors, Boolean logic, lists, and so on. Anything that can be stored on a computer can be represented by binary sequences.

Some data take many bits to represent it. For example, a single 10 MP (1 MP is one million pixels) picture uses 10,000,000 pixels.

Each pixel (16-bit mode RGB) contains $16 * 3 = 48$ bits = 6 bytes. That means there are $48 \text{ bits} * 10,000,000 = 480,000,000$ bits in a single 10 MP (16-bit mode) picture.

Digital vs. Analog

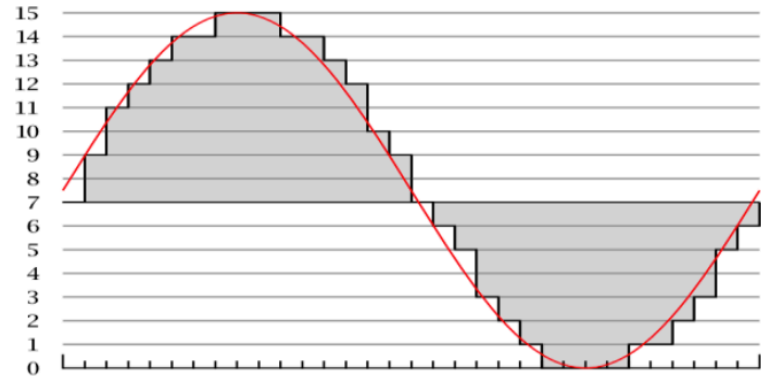
Many physical phenomena can be modeled by analog signal(sound, colors, temperature). Analog signals are continuous signals.

Computers can only understand **digital** signals (discrete or finite signal(0s and 1s).)

- analog signals are continuous and can take on an infinite possible values(the real numbers)
- digital signals are finite.
- For example, 8 bit colors can take on one of 256 discrete, finite possibilities. But actual colors can take on any of an infinite possible values or shades.

Sampling allow computers to approximate analog signals such as sound.

The number of samples is the **sampling rate**, the higher the rate the better the quality.



Big Idea 2: Data

In many programming languages, integers are represented by a fixed number of bits, which limit the range of integer values and mathematical operations on those values.

For example in JAVA, the range of the value of an integer is from $-2,147,483,648$ to $+2,147,483,647$. Trying to store a number bigger than the limits will result in an **overflow error**.

Some languages like Python, integers do not have limits on number size but, instead, expand to the limit of the available memory.

Big Idea 2: Data

The formula to calculate the largest number stored is $2^n - 1$, where n is the number of bits.

Example 1:

With 4 bits, the largest integer that can be stored is $2^4 - 1 = 15$.

Example 2:

A 4-bit integer can any value in $\{0, 1, 2, \dots, 15\}$. Thus storing the value of $10 + 6 = 16$ would cause an overflow error since the $16 = 10000$ requires at least 5 bits.

Example 3:

If x is a 3-bit integer, then $x = 111 + 111$ will cause an overflow error since the sum is 1110 which requires at least 4 bits.

Big Idea 2: Data

$1/3$ does not always equal $1/3$. A **roundoff error** occurs when decimals (real numbers) are rounded.

One computer might calculate $1/3$ as 0.333333. Another computer might calculate $1/3$ as 0.3333333333. In this case, $1/3$ on one computer is not equal to $1/3$ on a second computer.

Big Idea 2: Data

Number bases, including binary, decimal, and hexadecimal, are used to represent and investigate digital data.

On your AP exam, you will be expected to convert binary to decimal and decimal to binary only.

$$11011_{\text{BIN}} = ?_{\text{DEC}}$$

$$(16 * 1) + (8 * 1) + (4 * 0) + (2 * 1) + (1 * 1) = 27_{\text{DEC}}$$

Big Idea 2: Data

$30_{\text{DEC}} = ?_{\text{BIN}}$

| | | | | | |
|----|----|---|---|---|---|
| 32 | 16 | 8 | 4 | 2 | 1 |
| | | | | | |

There is one 16 in 30.

| | | | | | |
|----|----|---|---|---|---|
| 32 | 16 | 8 | 4 | 2 | 1 |
| | 1 | | | | |

What is left is $30 - 16 = 14$. There is one 8 in 14 so:

| | | | | | |
|----|----|---|---|---|---|
| 32 | 16 | 8 | 4 | 2 | 1 |
| | 1 | 1 | | | |

$$14 - 8 = 6$$

There is one 4 in 6 so:

| 32 | 16 | 8 | 4 | 2 | 1 |
|----|----|---|---|---|---|
| | 1 | 1 | | | |

$6 - 4 = 2$. And there is a 2 in 2 so

| 32 | 16 | 8 | 4 | 2 | 1 |
|----|----|---|---|---|---|
| | 1 | 1 | 1 | 1 | |

$2 - 2 = 0$. Thus

| 32 | 16 | 8 | 4 | 2 | 1 |
|----|----|---|---|---|---|
| | 1 | 1 | 1 | 1 | 0 |

Answer: 11110_{BIN}

Big Idea 2: Data

INFORMATION EXTRACTED FROM DATA

People generate significant amounts of digital data daily. Some always-on devices are collecting geographic location data constantly, while social media sites are collecting premium data based on your usage.

People can use computer programs to process information as well as to gain insight and knowledge. Information is the collection of facts and patterns extracted from data.

Gaining insight from this valuable data involves a combination of statistics, mathematics, programming, and problem solving.

Big Idea 2: Data

Large data sets may be analyzed computationally to reveal patterns, trends, and associations.

These trends are powerful predictors of future behaviors. Investors are constantly reviewing trends in past pricing to influence their future investment decisions.

However, sometimes trends can be misinterpreted and result in business disasters. Digitally processed data may show correlation between variables.

A correlation found in data does not necessarily indicate that a causal relationship exists. Additional research is needed to understand the exact nature of the relationship.

Big Idea 2: Data

Depending on how the data were collected, the information may not be uniform. For example, if users entered data into an open field, the way they chose to abbreviate, spell, or capitalize something may vary from user to user. Data sets pose challenges regardless of size, such as:

- The need to clean data
- Incomplete data
- Invalid data
- The need to combine data sources

Cleaning data is a process that makes the data uniform without changing their meaning.

One example is replacing all equivalent abbreviations with the same word. This can also be done with various spellings and with different capitalizations.

Big Idea 2: Data

Data can get too large for traditional data-processing applications. The ability to process data depends on the capabilities of the users and their tools. Social media activity generates an enormous amount of data.

Some data sets are difficult to process using a single computer and may require parallel systems.

Problems of **bias** are often created by the types and sources of data being collected. Bias is not eliminated by simply collecting more data. A large amount of data is generated by humans. Algorithms that use this data will reflect this bias.

Despite the advantages of big data, a large sample size can magnify the bias associated with the data being used. Data can have little value if the sample is not representative of the population to which the results will be generalized.

Big Idea 2: Data

Predicting algorithms use information collected from big data to influence our daily lives. For example:

- A credit card company can use purchasing patterns to identify when to extend credit or flag a purchase for possible fraud.
- Social media sites can use patterns to target advertising based on viewing habits.
- An online store analyzing customers' past purchases can suggest new products the customer may be interested in buying.
- An entertainment application may recommend an additional movie to watch based on the viewer's interests.

Big Idea 2: Data

Using appropriate visualizations when presenting digitally processed data can help one gain insight and knowledge. Although big data is a powerful tool, the data will lose their value if they cannot be presented in a way that can be interpreted.

Visualization tools can communicate information about data. Column charts, line graphs, pie charts, bar charts, XY charts, radar charts, histograms, and waterfall charts can make complex data easier to interpret.

Python's pandas library can be used to explore, process and visualize data. See the optional lecture slides "pandas for Tabular data". Python's pandas library is a powerful alternative to Excel.

Big Idea 2: Data

Privacy concerns arise through the mass collection of data. The content of the data may contain personal information and can affect the choice in storage and transmitting.

Anything done online is likely to lead to sharing of private data. Using Gmail to order a pair of shoes from Clarks could result in ads for shoes showing up in your search engine.

Big Idea 2: Data

Metadata are data that describe your data—for example, a picture of you standing in front of a waterfall is data. The location and time the picture was taken are metadata.

Metadata are used for finding, organizing, and managing information. Metadata can increase the effective use of data or data sets by providing additional information about various aspects of that data.

Changes and deletions made to metadata do not change the primary data.

Big Idea 3: Programming and Algorithms

The AP exam will use a language-agnostic syntax for programming and algorithm questions.

Please see the following reference sheet(also available during the actual exam) for more details about the syntax.

<https://apcentral.collegeboard.org/pdf/ap-computer-science-principles-exam-reference-sheet.pdf>

Big Idea 3: Programming and Algorithms

In computer science, an **abstraction** is a way to represent essential features without including the background details or explanations. Abstractions reduce complexity and allow for efficient design and implementation of complex software systems.

Abstractions become a necessity as systems become more complex. For example, anytime you check your stories on Instagram, you are using a bunch of processes in the background that you have no control over.

Without these abstractions, it would be difficult to send a message to a friend. With the use of abstractions, you can focus on content, not the technical details of how the application works.

Big Idea 3: Programming and Algorithms

Programmers also use abstractions. The purpose of abstraction is to hide coding details so the programmer can focus on the current problem.

Computers can understand only binary machine code. Machine code is a strictly numerical language that runs fast but is hard to use.

The code on the right is written in machine code to outputs "Hello World" to the screen.

In Python it can be done using the `print()` abstraction:

`print("Hello World")`

```
2  _10111000 _00100001 _00001010 _00000000 _00000000
3  _10100011 _00001100 _00010000 _00000000 _00000110
4  _10111000 _01101111 _01110010 _01101100 _01100100
5  _10100011 _00001000 _00010000 _00000000 _00000110
6  _10111000 _01101111 _00101100 _00100000 _01010111
7  _10100011 _00000100 _00010000 _00000000 _00000110
8  _10111000 _01001000 _01100101 _01101100 _01101100
9  _10100011 _00000000 _00010000 _00000000 _00000110
10 _10111001 _00000000 _00010000 _00000000 _00000110
11 _10111010 _00010000 _00000000 _00000000 _00000000
12 _10111011 _00000001 _00000000 _00000000 _00000000
13 _10111000 _00000100 _00000000 _00000000 _00000000
14 _11001101 _10000000
15
16 _10111000 _00000001 _00000000 _00000000 _00000000
17 _11001101 _10000000
```

Big Idea 3: Programming and Algorithms

Abstractions allow for programmers to use semihuman language to program(Python, Java, etc...).

Rarely will programmers deal directly in machine code. Machine code is a base language where no abstractions are implemented. Programmers have worked to hide details by using abstractions.

Different program languages offer different levels of abstractions. High-level programming languages provide more abstractions than do lower-level languages.

Coding in a programming language is often translated into code in another low-level language that the computer can execute.

Big Idea 3: Programming and Algorithms

Abstraction Examples Used on the AP Exam:

DISPLAY(expression) is an abstraction that is used on your AP exam to display a value of expression followed by a space. The input parameter for the DISPLAY abstraction is expression.

Another abstraction used on your AP exam is RANDOM(a, b), which evaluates to a random number from a to b inclusive. The input parameters in this abstraction are a and b.

An abstraction generalizes functionality with input parameters that allow software reuse. Being aware of and using multiple levels of abstractions in developing programs helps to apply available resources and tools more effectively to solve problems.

| Operator | Meaning | Example |
|----------|----------------|------------------------|
| + | Addition | $5 + 7 = 12$ |
| - | Subtraction | $2 - 1 = 1$ |
| * | Multiplication | $3 * 3 = 9$ |
| / | Division | $3/2 = 1.5$ |
| MOD | Modulus | $3 \text{ MOD } 2 = 1$ |

Operator Precedence (Order of Operations)

First: Parentheses

Second: MOD, *, /

Third: +, -

Big Idea 3: Programming and Algorithms

What is the value of a after the expression is evaluated?

$a \leftarrow 26 \text{ MOD } 2$

Since 2 goes into 26 a total of 13 times and since 26 minus 26 is 0, a is equal to 0.

What is the value of a after the expression is evaluated?

$a \leftarrow 17 \text{ MOD } 2$

Since 2 goes into 17 a total of 8 times and since 16 minus 17 is 1, a is equal to 1.

Big Idea 3: Programming and Algorithms

1. If the divisor is a multiple of the dividend, it will divide evenly with no remainder, resulting in a modulus calculation of 0.
 - $4 \text{ MOD } 2 = 0$
2. If the dividend is less than the divisor, the resulting modulus calculation will equal the value of the dividend.
 - $3 \text{ MOD } 4 = 3$
3. A zero to the right of MOD results in a DIVIDE BY ZERO error.
 - $6 \text{ MOD } 0 = \text{DIVIDE BY ZERO error}$
4. A zero to the left of MOD is feasible and results in a modulus calculation of 0.
 - $0 \text{ MOD } 6 = 0$

Big Idea 3: Programming and Algorithms

What will the following program display?

$a \leftarrow 5 + 4 * 2$

$b \leftarrow 4 \text{ MOD } 5$

DISPLAY $a + b$

Answer: 17

Big Idea 3: Programming and Algorithms

What will the following program display if the INPUT function reads an even number such as 4?

```
a ← INPUT( )
```

```
a ← a MOD 2
```

```
DISPLAY(a)
```

Answer: 0

What will the following program display?

```
a ← 3
```

```
b ← 3 MOD 5
```

```
DISPLAY a = b
```

a is initialized to 3; b is initialized to the remainder when 3 is divided by 5. Since 3 is equal to 3, the program will display true.

Answer: True

Big Idea 3: Programming and Algorithms

`a ← 3`

`b ← 14`

`c ← 5`

`a ← c`

`b ← a`

`DISPLAY(a)`

`DISPLAY(b)`

`DISPLAY(c)`

| <i>a</i> | <i>b</i> | <i>c</i> |
|--------------|---------------|----------|
| 3 | 14 | 5 |
| 5 | 5 | |

Answer: 5 5 5

If the below code was executed several times, what is the percentage of times “false” would be expected to be displayed?

```
a ← RANDOM (1, 4)
```

```
DISPLAY (a = 3)
```

The numbers 1, 2, 3, and 4 are all possible selections. The chance of 3 being selected is 1/4 or 25%. The chance of 3 not being selected is 3/4, which is 75%.

Answer: 75%

If the below code was executed several times, what is the percentage of times “true” would be expected to be displayed?

```
a ← RANDOM (1, 10)
```

```
DISPLAY (a ≤ 3)
```

Answer: 30%

Lists

Lists are an organized and formatted way of storing and retrieving data. Each element in a list can be accessed by its index.

Unlike some common programming languages, indexes start at 1 on the AP exam, not 0. Trying to access an index that does not exist will result in an index out of bounds error.

```
scores ← [11, 35, 6, 75, 37]
```

```
scores[1] = 11
```

```
scores[2] = 35
```

```
scores[3] = 6
```

```
scores[4] = 75
```

```
scores[5] = 37
```

Lists

```
namesOfMyDogs ← [ "Waffles", "Novack the 3rd", "Benji" ]  
                index      1           2           3
```

An element is an individual value in the list that is assigned a unique index. For all list operations, if a list index is less than 1 or greater than the length of the list, an error message is produced and the program will terminate.

```
namesOfMyDogs ← [ "Waffles", "Novack the 3rd", "Benji" ]  
newList = namesOfMyDogs  
  
a ← newList [0]
```

This code causes an error. This index will be out of bounds since newList has only the indexes 1, 2, and 3.

The `INSERT(list, i, item)` will insert the item at index `i` and shift right items at index `i` or higher.



The diagram consists of two rounded rectangular boxes. The top box contains the text `words ← "The", "little", "Frog", "Jumping"`. The bottom box contains the text `INSERT words, 3, "Green"`. The boxes are arranged vertically, with the bottom box positioned below the top box.

The data structure `words` will now contain the following after the `INSERT` method is used.

```
words["The", "Little", "Green", "Frog", "Jumping"]
```

The APPEND(list, item) will add the item to the end of the list.

```
words ← ["The"]  
INSERT(words, 1, "Green")  
APPEND(words, "Fox")  
APPEND(words, "Pig")  
APPEND(words, "Rhino")  
INSERT(words, 1, "Elephant")
```

The data structure words will now contain the following after the INSERT and APPEND methods are used.

```
words["Elephant", "Green", "The", "Fox", "Pig", "Rhino"]
```

The REMOVE(list, i) will remove the item at index i and shift left items at index i or higher.

Line 1: words \leftarrow ["Elephant", "Green", "The", "Fox", "Pig", "Rhino", "Fox"]

Line 2: DISPLAY(LENGTH(words)) // answer 7

Line 3: REMOVE(words, 1)

Line 4: REMOVE(words, 3)

Line 5: DISPLAY(words)

Answer: ["Green", "The", "Pig", "Rhino", "Fox"]

scores ← 96, 93, 90, 100, 92, 90

For EACH item IN scores

DISPLAY item

Output: 96 93 90 100 92 90

PROCEDURES

A procedure is a set of code that is referred to by name and can be called (invoked) at any point in a program simply by utilizing the procedure's name. In some languages, a procedure could be called a *method* or *subroutine*.

```
Line 1: PROCEDURE doubling(list)
Line 2: {
Line 3:   count ← 1
Line 3:   REPEAT LENGTH(list) TIMES
Line 4:   {
Line 5:     list[count] ← list[count] * 2
Line 6:     count ← count + 1
Line 6:   }
Line 7: }
```

What does the code do?

Answer: Double each value in the list.


```
Line 1: PROCEDURE keepPositive(aList, bList)
Line 2: {
Line 3:   FOR EACH item IN aList
Line 4:   {
Line 5:     IF(item < 0)
Line 6:       APPEND(bList, item)
Line 7:   }
```

What does the code do?

Answer: Add negative numbers from alist to the end of bList.

Line 1: scores \leftarrow [90, 89, 98, 100, 90]

Line 2: total \leftarrow findTotal(scores)

Line 3: DISPLAY(total)

Line 4:

Line 5: PROCEDURE findTotal(scores)

Line 6: {

Line 7: sum = 0

Line 8: FOR EACH item IN scores

Line 9: {

Line 10: sum \leftarrow sum + item

Line 11: }

Line 12: RETURN sum

Line 13: }











Output

467









Big Idea 3: Programming and Algorithms

ROTATE_RIGHT will rotate the robot 90 degrees clockwise.

| Initial Robot Direction | Command | Ending Robot Direction |
|---|----------------|---|
|  | ROTATE_RIGHT() |  |
|  | ROTATE_RIGHT() |  |
|  | ROTATE_RIGHT() |  |
|  | ROTATE_RIGHT() |  |

Big Idea 3: Programming and Algorithms

ROTATE_LEFT will rotate the robot 90 degrees counterclockwise.

| Initial Robot Direction | Command | Ending Robot Direction |
|--|---------------|--|
|  | ROTATE_LEFT() |  |
|  | ROTATE_LEFT() |  |
|  | ROTATE_LEFT() |  |
|  | ROTATE_LEFT() |  |

MOVE_FORWARD()



robot moves to



MOVE_FORWARD()



robot moves to



MOVE_FORWARD()



robot moves to



MOVE_FORWARD()



robot ERROR



To prevent the robot from moving off the grid and resulting in an error, use the `CAN_MOVE(direction)` abstraction.

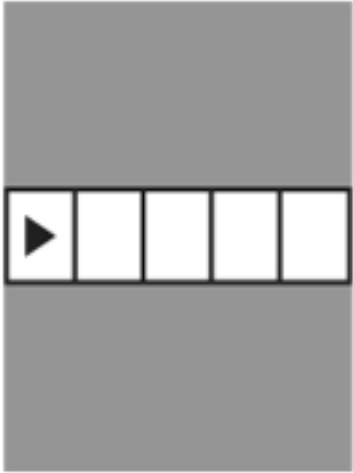
`CAN_MOVE(forward) = FALSE`



`CAN_MOVE(forward) = TRUE`



Robot starting location and direction shown.



Robot code:

```
Line 1: REPEAT_UNTIL(CAN_MOVE(forward) = false)
```

```
Line 2: {
```

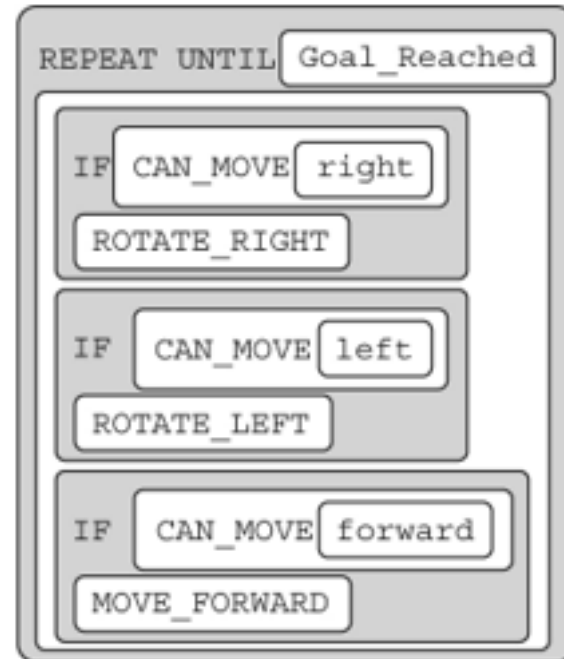
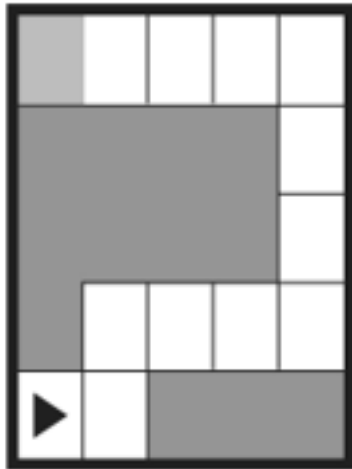
```
Line 3:   MOVE_FORWARD()
```

```
Line 4: }
```

What is the result of executing the above code?

Answer: Robot moves forward 4 steps.

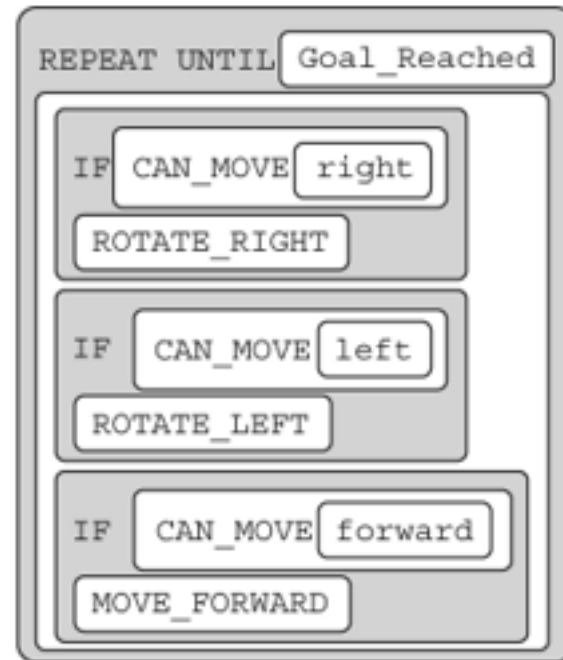
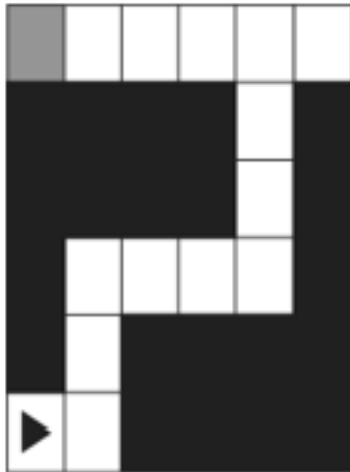
For the following grid, the program below is intended to move the robot to the gray square. The program uses the procedure `Goal_Reached()`, which returns “true” if the robot is in the gray square and returns “false” otherwise.



Does the code work as intended?

Yes, as an exercise, trace the code to convince yourself that the robot will reach destination.

For the following grid, the program below is intended to move the robot to the gray square. The program uses the procedure `Goal_Reached()`, which returns “true” if the robot is in the gray square and returns “false” otherwise.



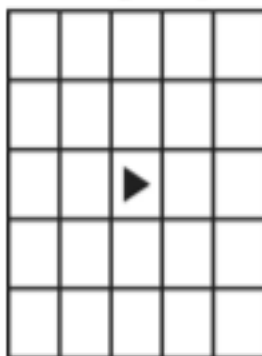
Does the code work as intended?

No! Robot get stuck in an infinite loop at the top right corner.

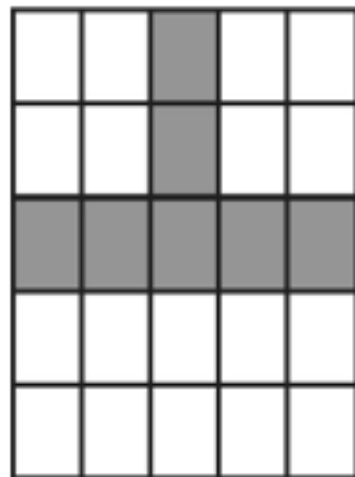
What are the possible robot landing spots when running the procedure below?

```
REPEAT ( RANDOM ( 0 , 2 ) )  
{  
  ROTATE_LEFT ( )  
}  
REPEAT ( RANDOM ( 0 , 2 ) )  
{  
  MOVE_FORWARD ( )  
}
```

Starting Map



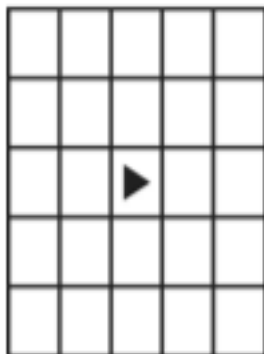
Answer:



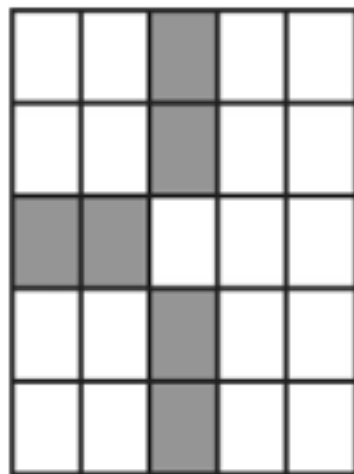
What are the possible robot landing spots when running the procedure below?

```
REPEAT ( RANDOM ( 1 , 3 ) )  
{  
  ROTATE_LEFT ( )  
}  
REPEAT ( RANDOM ( 1 , 2 ) )  
{  
  MOVE_FORWARD ( )  
}
```

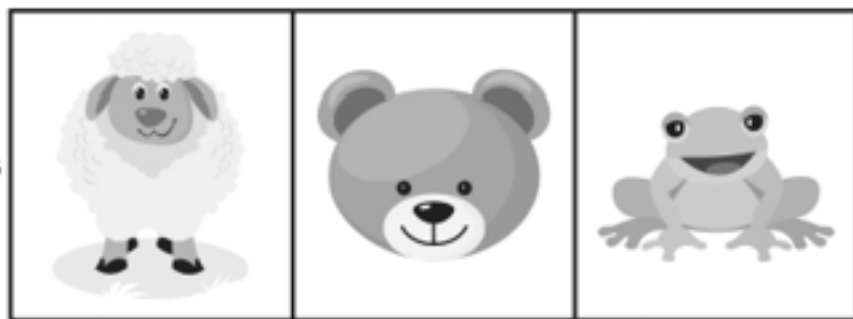
Starting Map



Answer:



Animals



A common algorithm is the swap. In the above animal data structure, we want to swap the sheep with the frog. Currently, the list contents are the following:

```
animals[1] = sheep
```

```
animals[2] = bear
```

```
animals[3] = frog
```

If the swap is successful, the animals' data structure will swap the sheep with the frog.

What steps are necessary for a successful algorithm that swaps the first data structure into the second data structure shown above?

Step 1: Create a temporary variable, and use it to store the value of the first item in the list.

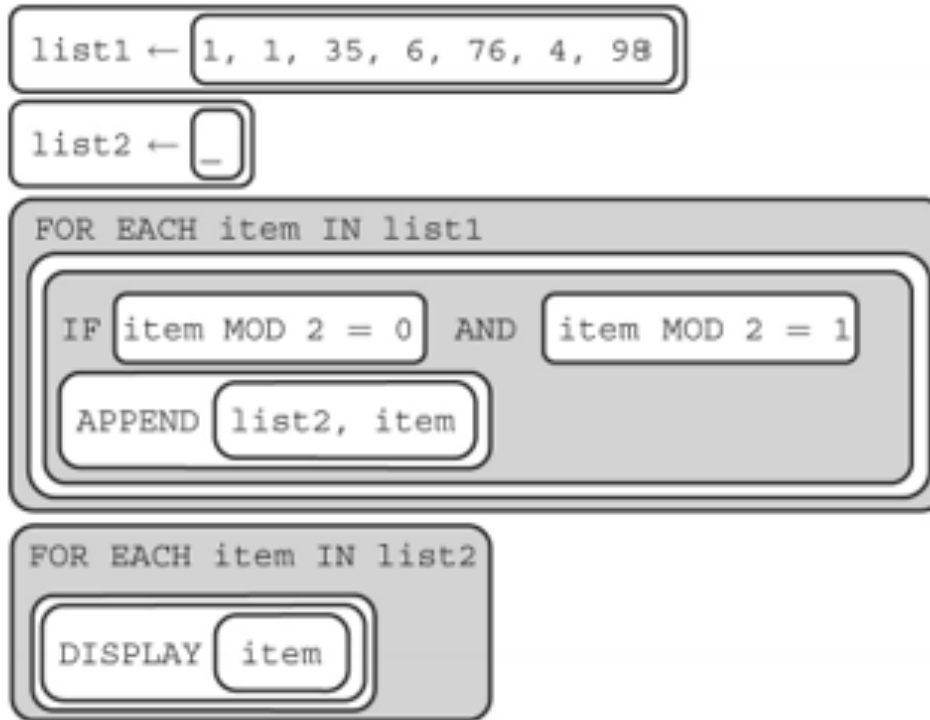
```
temp = animals[1]
```

Step 2: Replace the first item in the list with the third item in the list.

```
animals[1] = animals[3]
```

Step 3: Replace the third item in the list with the item that was stored in the temporary variable.

```
animals[3] = temp
```



What is the output?

Answer: Empty list, a number cannot be both even and odd. No number is appended to list2.

Linear Search

A linear search (or sequential search) is an algorithm for finding an element in a list. This search starts from the beginning of a list and sequentially checks each element of the list until a match is found or the entire list is searched without finding the element. A linear search can be used for either a **sorted list** or an **unsorted list**.

If a list has n elements, the worst case for the number of searches would be n . For example, if a list has 50 elements, the worst case would be 50 comparisons. However, the best case would be if the element you are looking for was found with the first comparison.

numList ← 11, 35, 2, 1, 56, 76, 3, 33, 90, 180

Using a linear search, how many comparisons would it take to find the number 11?

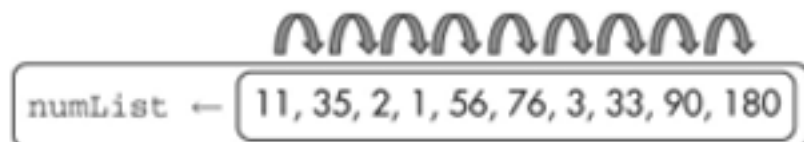


numList ← 11, 35, 2, 1, 56, 76, 3, 33, 90, 180

Answer: 1 comparison. This is the best case for a linear search.

numList \leftarrow 11, 35, 2, 1, 56, 76, 3, 33, 90, 180

Using a linear search, how many comparisons would it take to find the number 180?



Answer: 10 comparisons. This is the worst case for a linear search of a list with 10 elements.

Binary Search

A binary search is a search algorithm that halves the number of elements that need to be searched after every comparison. To use a binary search, **the list must be sorted**. This search compares the middle element of the list to the target value. If they are not equal, the half in which the target cannot lie is eliminated.

```
numList ← 1, 3, 5, 8, 56, 76, 300, 330, 900, 1870, 5444
```

What steps are needed for a binary search to find the number 300?

Step 1: Compare the middle element.

```
numList ← 1, 3, 5, 8, 56, 76, 300, 330, 900, 1870, 5444
```



Since 76 is not equal to the target and 300 cannot be on the left side of the list, we throw out the left side of the list.

```
numList ← 1, 3, 5, 8, 56, 76, 300, 330, 900, 1870, 5444
```

Step 2: Compare the middle element of the remaining numbers.

300, 330, 900, 1870, 5444



Since 900 is not equal to the target and 300 cannot be on the right side of the list, we throw out the right side of the list.

300, 330, ~~900, 1870, 5444~~

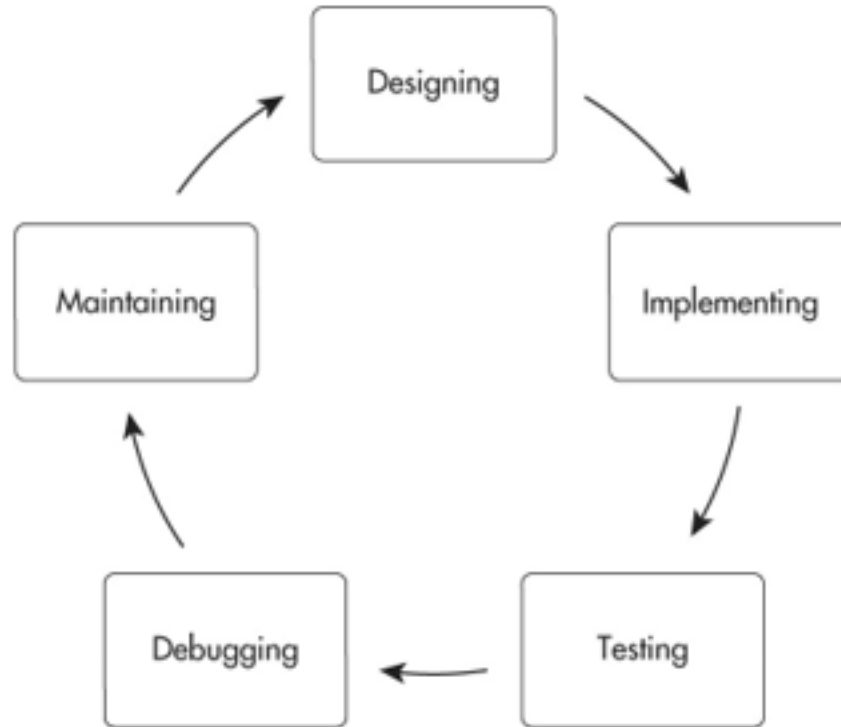
Step 3: Compare the middle element of the remaining numbers (round down).

300, 330







300 = target

Program Design

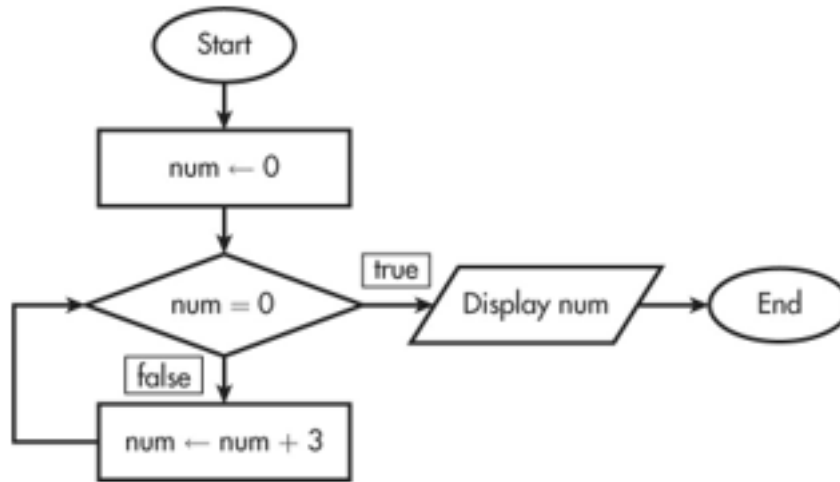


FLOWCHARTS

A flowchart is a way to represent an algorithm visually. The flowcharts below use the following building blocks.

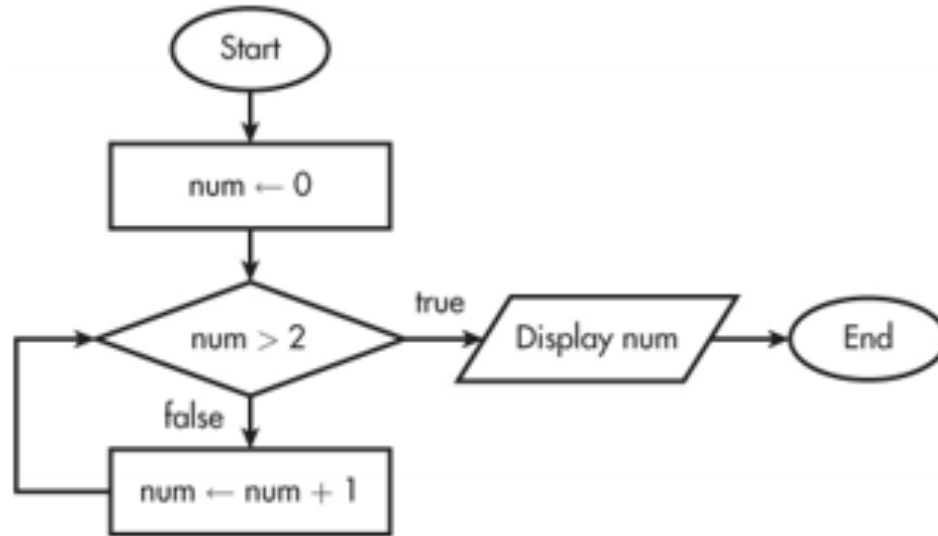
| Block | Explanation |
|---|--|
| Oval  | The start or end of the algorithm |
| Rectangle  | One or more processing steps, such as a statement that assigns a value to a variable |
| Diamond  | A conditional or decision step, where execution proceeds to the side labeled <code>true</code> if the condition is true and to the side labeled <code>false</code> otherwise |
| Parallelogram  | Displays a message |

Trace the code and determine the output of the following program.



Answer: Step 1: Start the program.
 Step 2: Set num = 0.
 Step 3: If n equal to 0, evaluates to true.
 Step 4: Display 0.
 Step 5: End the program.

Trace the code and determine the output of the following program.



Answer: Display 3.

Algorithms to Know

The AP MCQ will ask questions about standard algorithms every programmer should know:

- 1) Find total sum of a list of numbers
- 2) Find average of a list of numbers
- 3) Find maximum/minimum of a list of numbers
- 4) Find word from a list of words

Please know these algorithms and know how to quickly recognize them! If the code for any of the algorithm above is given with the name "mystery_algorithm", you should be able to quickly recognize what it does.

We'll go over each of them.

Find total sum of a list

```
PROCEDURE findTotal(scores)

{

sum = 0

FOR EACH item IN scores

{

sum ← sum + item

}

RETURN sum

}
```

Find average of a list(implementation #1)

Implementation #1: Use for Each Loop.

```
PROCEDURE findAverage(list)
{
    sum ← 0
    FOR EACH item IN list
    {
        sum ← sum + item
    }
    RETURN(sum/LENGTH(list))
}
```

Find average of a list(implementation #2)

Implementation #2: Use REPEAT n times loop.

```
PROCEDURE findAverage(list)
{
  sum ← 0
  count ← 1
  n ← LENGTH(list)
  REPEAT n TIMES
  {
    sum ← sum + list(count)
    count ← count + 1
  }
  RETURN (sum/n)
}
```

Find maximum of a list(implementation #1)

Implementation #1: Use for Each Loop.

```
PROCEDURE findMaximum(list)
{
    max ← list[1]

    FOR EACH item IN list
    {
        IF(item > max)
            max = item
    }
    RETURN(max)
}
```

Find maximum of a list(implementation #2)

Implementation #2: Use REPEAT n times loop.

```
PROCEDURE findMaximum(list)
{
max ← list[1]
n ← LENGTH(list)
count ← 1
REPEAT n TIMES
{
IF(list[count] > max)
    max = list[count]
    count ← count + 1
}
RETURN(max)
}
```

Common Error Example I

The AP exam will try to give you code that is the wrong implementation of an algorithm. **Can you find the error?**

```
PROCEDURE findMaximum(list)
{
  max ← 0
  FOR EACH item IN list
  {
    IF(item > max)
      max = item
  }
  RETURN(max)
}
```

Using the above code with the list $[-1, -1, -35, -6]$ will return 0, not the expected maximum number!

Common Error Example 2

The AP exam will try to give you code that is the wrong implementation of an algorithm. **Can you find the error?**

```
PROCEDURE findMinimum(list)
{
  min ← list[1]
  FOR EACH item IN list
  {
    IF(item < min)
      min = item
    ELSE
      min = 0
  }
  RETURN(min)
}
```

Using the above code with the list [1, 1, 35, 6] will return 0, not the expected minimum number!

Correct findMinimum

Here's the correct implementation of findMinimum.

```
PROCEDURE findMinimum(list)
{
min ← list[1]
FOR EACH item IN list
{
    IF(item < min)
        min = item
}
RETURN(min)
}
```

Find word from list of words

```
PROCEDURE findWord(list, word)
```

```
{
```

```
  index  $\leftarrow$  1
```

```
  FOR EACH item IN list
```

```
  {
```

```
    IF(item = word)
```

```
    {
```

```
      RETURN index
```

```
    }
```

```
    ELSE
```

```
    {
```

```
      index  $\leftarrow$  index + 1
```

```
    }
```

```
  }
```

```
  RETURN("Word not in list")
```

```
}
```

This code also works if we remove the Else block:

```
IF(item = word)
```

```
{
```

```
  RETURN index
```

```
}
```

```
index  $\leftarrow$  index + 1
```

Algorithmic Efficiency

Some problems cannot be solved in a reasonable amount of time because there is no efficient algorithm for solving them. In these cases, approximate solutions are sought.

Algorithms with a polynomial efficiency(constant, linear, square, cube, etc.) are said to run in a **reasonable amount of time**. They can be executed quickly on a modern processor.

However, there exists important and practical problems for which there exists no known polynomial time algorithm. Algorithms with exponential or factorial efficiencies are examples of algorithms that run in an **unreasonable amount of time**.

Algorithmic Efficiency

A **heuristic** is an approach to a problem that produces a solution that is not guaranteed to be optimal but may be used when techniques that are guaranteed to always find an optimal solution are impractical.

For example, a file-organizing algorithm determines the content of a file based on a certain number of bytes in the beginning of the file. This is an approximate solution since only a few bytes are examined. But it is more practical and faster to run than examining every byte of every file.

Programmers break down problems into smaller and more manageable pieces. By creating procedures and leveraging parameters, programmers generalize processes that can be reused.

Procedures allow programmers to draw upon existing code that has already been tested, allowing them to write programs more quickly and with more confidence.

A **software library** contains procedures that may be used in creating new programs. (e.g. Python's random, numpy libraries)

The use of libraries simplifies the task of creating complex programs (abstraction).

Application program interfaces (APIs) are specifications for how the procedures in a library behave and can be used.

For example, Twitter's API allow programmers to access and analyze tweets.

Documentation for an API/library is necessary in understanding the behaviors provided by the API/library and how to use them. Twitter has documentation that allows programmers to learn how to use their API.

References

Reichelson, Seth. AP Computer Science Principles Premium with 6 Practice Tests (Barron's Test Prep) (p. 92). Barrons Educational Series.