P1L - Algorithmic Thinking and Representation (ATR) Pre-Lab

For most programming pre-labs, there is some reading and a walkthrough programming activity to illustrate some of the points you read about. For this pre-lab, it's just a reading. The reading is specifically the *Programming for Engineering* chapter from the *Programming for Engineering* course pack. You are being provided a different copy of it since I've added some questions here to track progress.

Programming for Engineering

Computers and microcontrollers are becoming more and more prevalent in engineering design. There are few remaining corners of our profession where programming, or at least a familiarity with the algorithmic thinking involved in programming, is not utilized and more often is outright necessary. All engineers should have some experience with programming and the elements involved. They should be capable of interpreting algorithms and understanding how they are supposed to function.

This first chapter in your programming reference text, **Programming for Engineering**, serves as a resource for your use of programming in your cornerstone course and its representation. It is not an instructional text but rather a review and reference for common aspects of programming that are independent of any programming language. This includes Proper Programming Practices, Representing Programs/Algorithms, and Understanding basic Programming Elements. For each of the programming elements, their pseudocode and flowchart representation are included. Some best practices tips are also covered. These are meant to be used as a reminder/reference and not as a robust lesson on their use.

The other two chapters serve as command and usage references for the programming languages used in Cornerstone, **C++** and **MATLAB**. These two are also not instructional texts but rather reference texts, to be used as resources for your reference. They cover the basic description and guidance to reinforce a basic level of proficiency of these engineering tools gained through your in-class instruction.

P1L: Experience Levels

Grade Responses

This is an anonymous discussion so you can see the range of experiences in our classroom. Our program expects that some students have 0 programming experience while others might have been programming for years. There's benefits and challenges to both in this curriculum.
Please **BRIEFLY** describe your level of programming and algorithmic thinking experience. If None, it's ok to post "None."

Responses

Reply

Ordered by Newest Responses ▼

Albert Lin 4 months ago

Little bit of python, c++, more than a little bit of java

Grading 1/1

Comments 0 👍 1

Ansh Vats 4 months ago

Little but of python and java

Grading 1/1

Comments 0 👍 1

Foster Ward 4 months ago

none

Grading 1/1

Comments 0 👍 1

Jonah Wang 4 months ago

I've done a little bit of programming in java

Grading 1/1

Comments 0 👍 1

Juan-Mario Vuksan 4 months ago

i have done a little of programming in the past but its been a long time.

Grading 1/1

Comments 0 👍 1

Charles Wamester 4 months ago

I have programmed in javascript and python before but it has been a long time since.

Grading 1/1

Comments 0 👍 1

Roman Ton

4 months ago

None

Grading 1/1

Comments 0 👍 1

Phil Warren

4 months ago

Python and a little arduino. Don't recall a lot

Grading 1/1

Comments 0 👍 1

Kyra Pallod

4 months ago

Python in high school but I dont remember much

Grading 1/1

Comments 0 👍 1

Harrison Zabar

4 months ago

None

Grading 1/1

Comments 0 👍 1

Aditya Chowdhury

4 months ago

none

Grading 1/1

Comments 0 👍 1

Korel Unluer

4 months ago

None

Grading 1/1

Comments 0 👍 1

Owen Weiss

4 months ago

i have experience from doing arduino in highschool as well as taking CS9 and APCSA but im not very good...

Grading 1/1

Comments 0 👍 1

Liam Yin 4 months ago

none

Grading 1/1

Comments 0 👍 1

David Hamman 4 months ago

Some experience in python.

Grading 1/1

Comments 0 👍 1

Joshua Wolf 4 months ago

I have experience in python, java, and swift, but most of my knowledge has slipped my mind.

Grading 1/1

Comments 0 👍 1

Oscar Peng 4 months ago

intermediate to advanced level in java and beginner in python

Grading 1/1

Comments 0 👍 1

John Wolfe 4 months ago

None.

Grading 1/1

Comments 0 👍 1

Jacob Hathcock 4 months ago

I have experience with Python and Java, ive frequently used both through schoolwork the past 3 years.

Grading 1/1

Comments 0 👍 1

Hana Wright 4 months ago

None

Grading 1/1

Comments 0 👍 1

Benjamin Anderson 4 months ago

mostly minimal and basic programming

Grading 1/1

Comments 0 👍 1

Katherine Woodbury 4 months ago

I have done some basic Python and more advanced Java programming.

Grading 1/1

Comments 0 👍 1

Dante Varano 4 months ago

Scratch

Grading 1/1

Comments 0 👍 1

Bella Wong 4 months ago

None

Grading 1/1

Comments 0 👍 1

James Wheeler 4 months ago

Done a little bit, but years ago, and don't really remember anything

Grading 1/1

Comments 0 👍 1

Foster Ward 4 months ago

none

Grading 1/1

Comments 0 👍 1

Jonah Wolk 4 months ago

none

Grading 1/1

Comments 0 👍 1

Taylor Luks 4 months ago

None

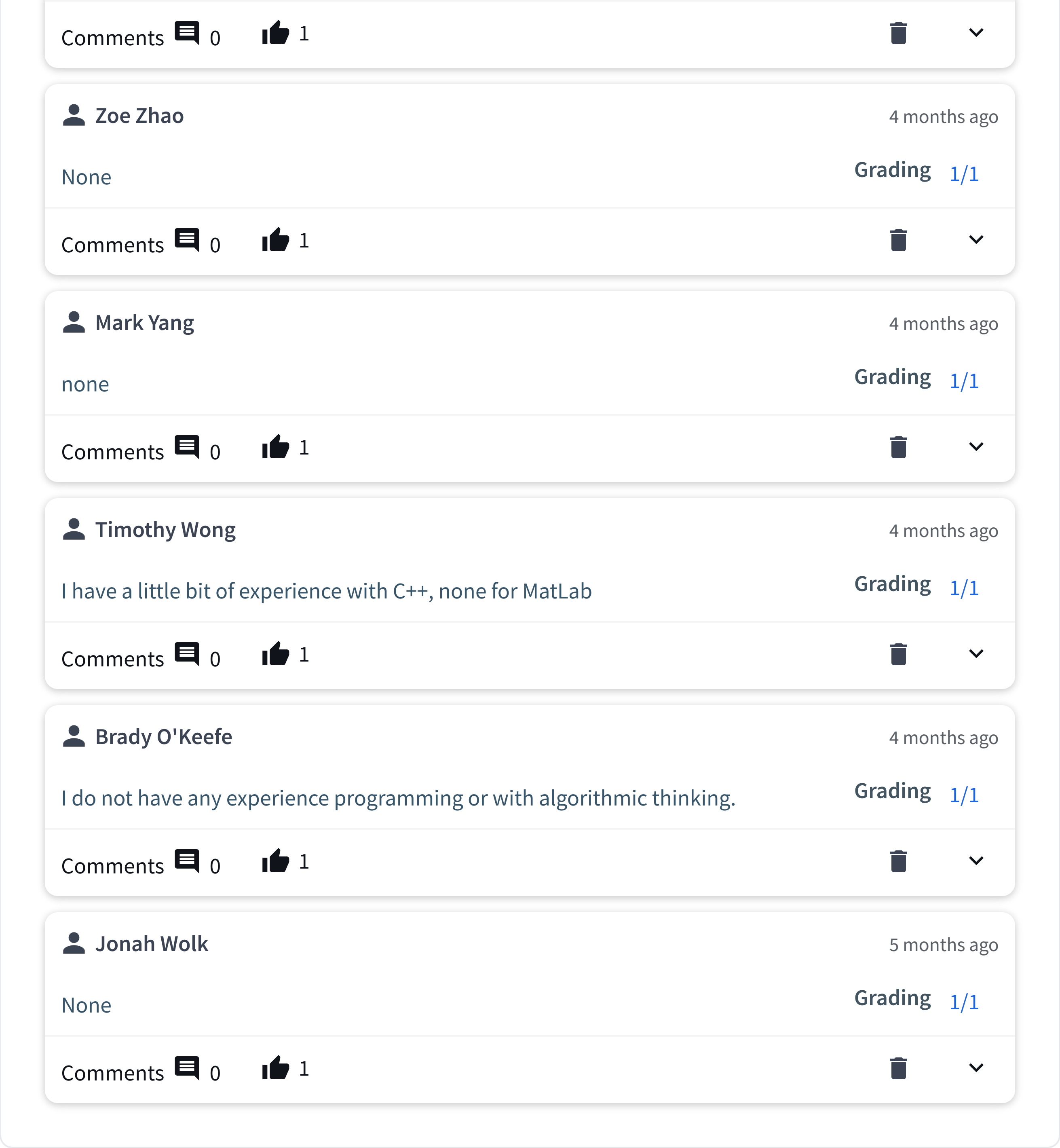
Grading 1/1

Comments 0 👍 1

Nathan Yost 4 months ago

I have done some basic Arduino and vex robotics coding

Grading 1/1



Comments 0 👍 1

Zoe Zhao 4 months ago

None

Grading 1/1

Comments 0 👍 1

Mark Yang 4 months ago

none

Grading 1/1

Comments 0 👍 1

Timothy Wong 4 months ago

I have a little bit of experience with C++, none for MatLab

Grading 1/1

Comments 0 👍 1

Brady O'Keefe 4 months ago

I do not have any experience programming or with algorithmic thinking.

Grading 1/1

Comments 0 👍 1

Jonah Wolk 5 months ago

None

Grading 1/1

Comments 0 👍 1

Proper Programming Practices

Human Readable Elements

A good program should be self-documenting, meaning that it should not only include the necessary code to achieve its purpose when run, it should also include enough additional information that a non-programmer should understand what the code is supposed to do when read. This is achieved by including the human readable elements throughout your code, specifically a **header** and **comments**.

Headers

Every code should contain a header block at its top. The format for commenting out a block of text is language dependent but your header should contain:

- File Name
- Author(s) name
- Date
- Assignment (If Applicable)
- Description of the code
- Usage instruction for the code (If Applicable)

Comments

Comments are required to explain the purpose and actions behind your code.

- Include enough detail that another novice programmer could understand what your code is attempting to do.
  - Pseudocode (readable code-like statements done before programming) is a good starting point for this.
- Avoid using 1 comment per line of code. There is such a thing as over-commenting.
  - This will take time and experience to get a good sense of how much is too much or too little but be closer to over-commenting in the beginning.

Indentation/White Space

Proper indentation/white space should be included throughout your code. It makes your code easier to follow and understand. Dependencies should be indented one more from the previous line. These include **while**, **do**, **for**, **if**, and **switch** statements as well as in **function** blocks. Let the development environment help you out with this.

P1L: Content check 1 ✓ 100% Correct 31/31

Why is it important to use indentation and white space in your code?

?

A	It allows the program to run faster

B It ensures that the code is secure

C It makes your code easier to follow and understand

D It reduces the size of the files

HINT

EXPLANATION

Show Responses Show Answer

Developing algorithms

Things to remember when developing your algorithms:

- Computers are Limited. They do not easily work along parallel paths. They work step-by-step.
- They can’t just take in everything at once and sort. They handle data bit by bit, one item at a time.
- Sequence matters. They do everything in the order they're given.
- Start Small. If you're told to program something to alphabetize an unknown number of words, start with considering how you'd place 2 words in order. Then how would you do it if you had 4, then 10, then an unknown number of words.
- Experience Helps.
  - How have you solved similar problems in the past?
  - How have others solved similar problems?
- Ask for help. New eyes or someone else's experience can reveal the logical steps necessary.

Representing Programs

Programs and other logical constructs are represented primarily through **pseudocode** and **flowcharts**. These represent the steps in a program in a human readable format. There is no limitation of syntax to cause errors in *'Running'* the program since, in this form, they are meant to lay out the logical steps for another human being to interpret. They are the planning or outline of the program before filling it in with the commands and syntax of any formal programming language. Programmers use these to help develop the logic of their algorithms before investing time in hammering out the details of full implementation and specific syntax. These should represent the program to a sufficient level of detail that it supports the formal programming efforts.

P1L: Pseudocode Analogs Show Responses

No correct answers: No correct answer has been set for this question

Can you think of other instances where complex concepts or actions are represented in simplified shared terms and/or representations? Please name one or two if you can. A negative response is also accepted if you can't think of anything (And required so Top Hat will give you participation credit).

Pseudocode

Pseudocode is a **plain language** version of your program. It is written out, line-by-line, with each line being a step in the program. These steps may represent a single line of code or possibly several. Pseudocode tends to be a preferred development tool because it easily converts to comments in your formal code.

Standard formats for pseudocode do exist. The guidelines provided here represent a simplified version of some of the standard formats and you should check with your instructor or other course materials for the level of detail required of you. In the future, you employer or other associated institutions may have a specific standard format they require or a style guide they expect you to use.

Basic Pseudocode Rules

Use semi-colons to end lines

For young programmers, this is simply a habit building requirement. Many programming languages use semi-colons to indicate a command line break. Getting into the habit will help eliminate this common compiler error when you start with formal languages.

Example:

```
<Do Something>;
```

`IMPORTANT INFORMATION:` I use `< >` as a placeholder in many examples and in the starter codes I provide you. When you see these, you're expected to replace the entire placeholder, so the space indicated by and including the angle brackets, with something based on the description between the them. For instance, let's say you're told to use the following as a template to complete your assignment/activity :

`My favorite dessert is <Your favorite food>.`

If *peanut butter pie* is your favorite food (As it is mine), you would then submit your version as:

`My favorite dessert is peanut butter pie.`

This is used throughout the class and other assignments. Let's try it.

P1L - Using placeholder brackets

Grade Responses (0)

Write your version of the following prompt:

`My favorite dessert is <Your favorite food>.`

Students will write their responses here...

Specify start and ends of elements

Programs are broken up into functional blocks so your pseudocode should represent that same organization of the logic. In general, you want to make clear where the beginning and end of those blocks are. In the specific programming element sections below, examples for how to do so for each are provided.

Example:

```
PROGRAM <ProgramName>:
< Activity 1>;
IF <conditional>
  < Activity 2>;
ELSE
  < Activity 3>;
ENDIF
Call <FunctionName>;
< Activity 4>;
END.
FUNCTION <FunctionName>:
< Function Activity >;
END.
```

No data declaration

You do not need to declare data and functions at the start of your pseudocode.

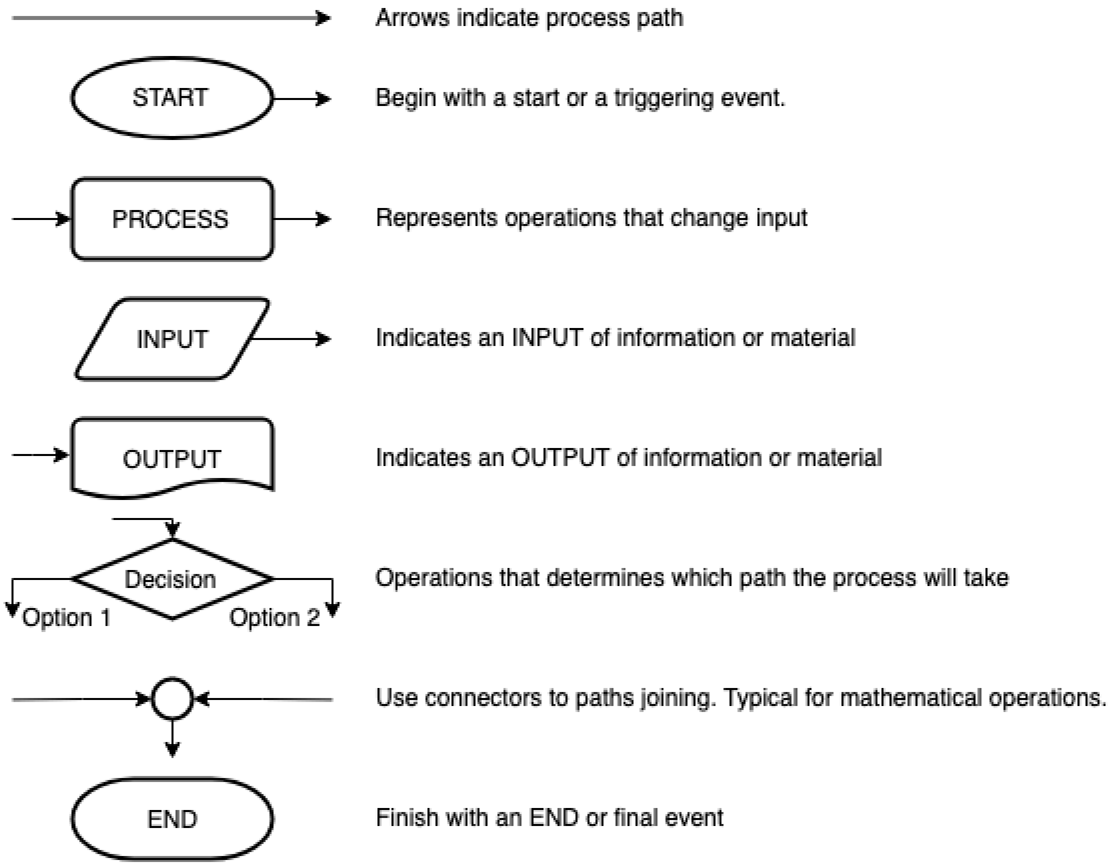
ATR Pre-Lab: Pseudocode Styles Show Correct Answer Show Responses

Name a standard style of pseudocode (Hint: Click the link earlier in this section)

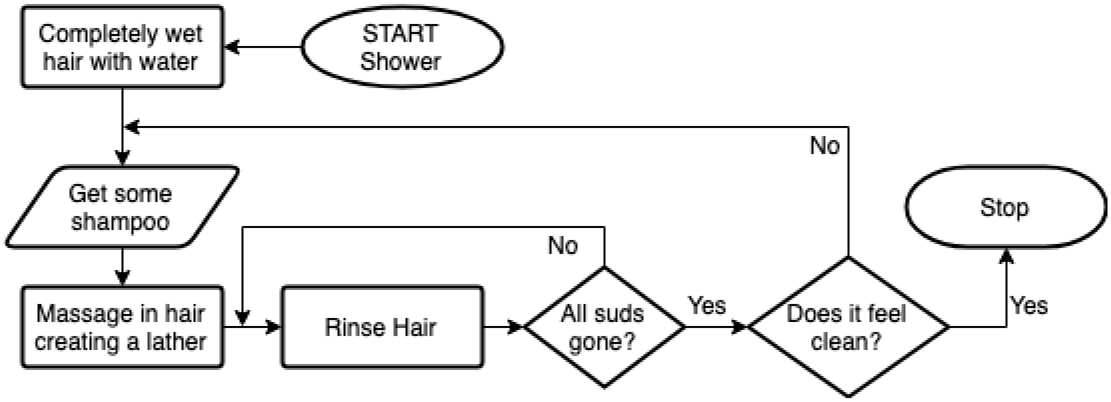
Flowcharts

Flowcharts communicate an algorithm through a graphical representation. It uses graphics to make clear the logical path the program takes. This method is typically preferred for communicating an algorithm in a presentation or report. They tend to be easier to quickly be read by an attendee or reader. This is due to the use of a set of standardized symbols with descriptions and flowlines to visually organize and communicate your algorithm.

Those standardized symbols are listed below:


Arrows indicate process path
START
Begin with a start or a triggering event.
PROCESS
Represents operations that change input
INPUT
Indicates an INPUT of information or material
OUTPUT
Indicates an OUTPUT of information or material
Decision
Option 1
Option 2
Operations that determines which path the process will take
Use connectors to paths joining. Typical for mathematical operations.
END
Finish with an END or final event

Standard symbols for Flowcharts and their Meanings



Flowchart Example for Washing your Hair

Submitting Flowcharts

For any submission requirements and guidelines, please refer to any requirements laid out by your instructor. The following are just general guidelines.

If your writing is legible, hand drawn flowcharts are typically acceptable. If you have to ask if your writing is legible enough, it isn’t. Take the time to represent it in a cleaner format using a computer aid.

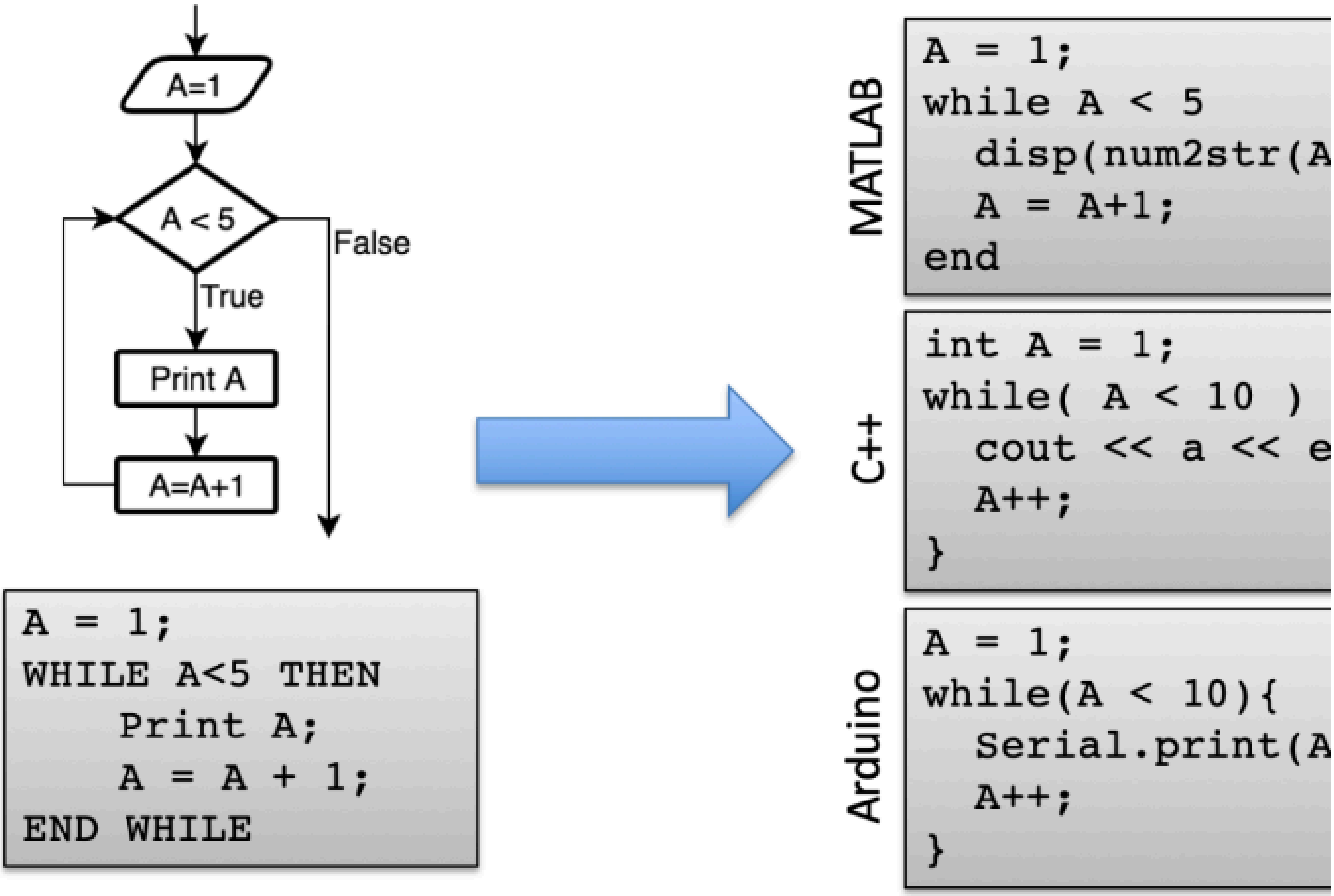
For presentation and documentation though, hand drawn flowcharts **are never** acceptable. You should be using a drawing program to make a clear and crisp flowchart like those above.

There are several options available for this:

- Diagrams.net (Formerly Draw.io)
  - This is what was used to make the flowcharts here
- Microsoft Word
  - The symbols are available under 'Shapes'
- DIA Diagram Editor
  - A dedicated open-source flowchart creation tool
  - Well-recommended with available tutorials
- Adobe Illustrator
  - A powerful and popular graphic tool

Translate to Code

Proper use of these representations can make translating your logic into actual programming language much easier. The formats and standardized forms line up with how programming languages are already set up. It becomes a matter of converting the syntax and filling in the blanks.



```
A = 1;
WHILE A<5 THEN
    Print A;
    A = A + 1;
END WHILE
```

```
A = 1;
while A < 5
  disp(num2str(A
  A = A+1;
end
```

```
int A = 1;
while( A < 10 )
  cout << a << e
  A++;
}
```

```
A = 1;
while(A < 10){
  Serial.print(A
  A++;
}
```

Example of how snippets of a pseudocode and flowchart translate to programming languages (Missing required coc link and definition sections of the formal code)

This is a *very* simple example, essentially a 1 to 1 conversion. There will be times when a single line of pseudocode or a symbol on a flowchart will actually represent multiple lines of code. Even in this example, the formal code is missing the definition sections of the code where some of the variables would be declared and the requires libraries loaded and set up. For instance, you may say ‘Record data to file’ which would require an additional header in your code, opening the file, printing the information to the file, and then closing the file. It is fine to represent all of that concisely in your pseudocode but understand the translation is more complex than the above example.

Common Programming Elements

The following are common programming elements. These are the items and operations that, in combination with one another, make up the algorithms which drive your programs. They are independent of any programming language but serve the same function. These elements are used to implement programs, the specific syntax will change by language.

Storing Data

Value

A representation of information of a specific data type. Values can be altered by the program but only within the allowable range of that data type.

Variable

The storage location and associated symbolic name set aside for a value or value(s).

Typically the data type and size of the variable can not be changed once set. MATLAB is an exception to this constraint.

Variables can come in different sizes based on the number and arrangement of values contained:

- **Scalars -** stores a single element
- **Arrays -** elements of the same data type collected along a single dimension
- **Matrices -** elements of the same data type collected along multiple dimensions, typically 2

Data Types

Selecting a data type determines the kind of information that can be stored in a variable. The available data types change between programming languages but the following are commonly used.

For the numerical types, integers and floating points, they have limits in the allowable range of values due to the amount of memory set aside for the variable location. For instance, an int type integer is allocated 16 bits so could have a possible 2^16 values.

Integers

Numbers without fractional content, also known as whole numbers.

Signed is the default integer type. This means the value can either be positive or negative numbers.

Unsigned integers are always positive. They don't allow for a sign, therefore unsigned, in front of the number, i.e. the - symbol is not allowed to be used, forcing it always positive.

The specific data types may have slightly different names in different languages.

Data Type	Bits	Range
short	16	−32,768 to 32,767
long	32	−2,147,483,648 to 2,147,483,647
long long	64	−2^63 to 2^63-1
unsigned shot	16	0 to 65,535
unsigned long	32	0 to 4,294,967,295
unsigned long long	64	0 to 2^64-1

The availability of some data types may be system dependent. For instance, a 32-bit system would not be able to provide 64 bit data types.

Floating Points

Numbers that have decimal representations. They are interpreted in scientific notation: M x 10^n where the size is limited by the amount of memory allocated for the significand(M) and the exponent(n).

Data Type	Bits	Range	Significant Digits
float	32	+/- 3.40282 x 10 ^ +/- 38	7
double	64	+/- 1.79769 x 10 ^ +/- 308	15

Ranges written using more formal mathematic representation (Top Hat doesn't yet allow them in Tables)

- **float** - $\pm 3.40282^{10 \pm 38}$
- **double** - $\pm 1.79769^{10 \pm 308}$

The availability of some data types may be system dependent. For instance, a 32-bit system would not be able to provide 64 bit data types.

Boolean

Represents a binary choice of **True** or **False** by assigning 1 bit to **0** or **1**. Acceptable text representations vary between programming languages but most accept the integer values of 0 or 1.

Characters

A single letter of the alphabet, a digit, a punctuation mark, etc. Encoded as a single byte of information that is then cross referenced against a character set. ASCII is the most common standard character set.

Strings

An array of characters, although not always treated specifically as an array variable. Typically used to store words and text.

Other

Non-native data types are available but their availability is dependent on the programming language and whether if external class or header files are available and provided (Additional code to preload to include definition of the non-native data type). Some IDEs load and make available non-native data types by default.

P1L: Data Types Concept Check

✓ 100% Correct 31/31

When dealing with data types in programming, which of the following statements is accurate?

?

A Unsigned integers can be both positive and negative.

B A 32-bit system can provide 64 bit data types.

C One bit is assigned to represent a binary choice of True or False.

D Non-native data types' availability is not dependent on the programming language.

HINT

EXPLANATION

Show Responses Show Answer

Representation of Storing Data

Pseudocode

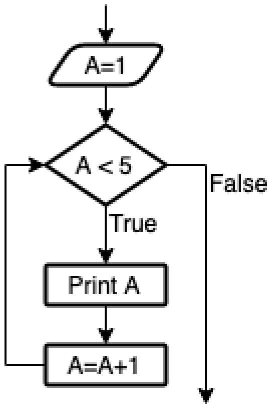
Creating a variable to store data in pseudocode is as simple as referring to it as it comes up. See the example below.

```
1. PROGRAM <ProgramName>:
2. Initialize A = 5;
3. Set A = 5;
4. A = 5;
5. A = A + 3;
6. Increase A by 3;
7. Use A for <Some action>;
8. END.
```

Lines 2-4 are sufficient for initially creating any variable. Lines 5-7 are acceptable for using the variable. You do not need to declare data types in pseudocode unless you need to for the logic to make sense. There is a lot of freedom in how you introduce and handle variables in pseudocode.

Flowchart

There is a similar amount of freedom in Flowcharts. Once data is introduced and given a name, that's the variable. This can take place in any of the symbols by stating the variable and doing something with it.



For instance, see the example shown to the right. A is used in multiple places and in different symbol types. As long as it is clear that A is a placeholder for some data and can be followed, its use is not constrained in any meaningful or highly limiting way.

Conditionals

This is the core to how computers make decisions. Through the use of operators to define the conditions, they check to see if a conditional is TRUE or FALSE. This comparison then leads to different possible alternatives based on the conditional's result.

Boolean operators (AND, OR, NOT)

Used to determine a TRUE or FALSE based on whether the combination of multiple variables or operations meet the operator criteria

Operator	Example	Description
AND	A AND B	TRUE if A and B are both TRUE
OR	A OR B	TRUE if either A or B are TRUE
NOT	A NOT B	TRUE only if A is TRUE and B is FALSE

Comparative operators (GREATER THAN, LESS THAN, etc)

Used to determine a TRUE or FALSE based on whether a size comparison between two values

Operator	Common Symbol	Example	Description
Greater Than	>	A > B	TRUE if A is Greater Than B
Less Than	<	A < B	TRUE if A is Less Than B
Greater than or equal to	>=	A >= B	TRUE if A is Greater Than B or Equal to B
Less than or equal to	<=	A <= B	TRUE if A is Less Than B or Equal to B
Equal to	==	A == B	TRUE if A is Equal to B
Not Equal to	!=	A != B	TRUE if A is Not Equal to B

Representation of Conditionals

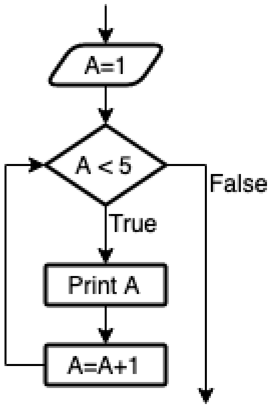
Pseudocode

In the tables above, using language similar to the 'Description' columns is typical. Lines 2 and 3 below use this style. Use of the 'Common Symbol' can be used but it limits the extent of understanding. Someone not familiar with programming languages may not recognize != as meaning 'not equal to'. Conditionals typically take place within other programming elements though. We have not gotten to 'If this then that' but you can see how the conditional is integrated in line 6 below. You can see more examples of representing conditionals in the Representation section of programming elements that depend on them.

```
1. PROGRAM <ProgramName>:
2. Result1 = A is less than B;
3. Result2 = A equals B;
4. IF A is less than B THEN
5.     <Do something in that case>;
6. ENDIF
7. END
```

Flowchart

Conditionals are inherently a decision and there is a specific symbol for that. The diamond shape, shown in the example as 'A<5', is a conditional statement. The two available options, True and False, branch off from the diamond, showing the potential paths depending on the answer.



Branching

Branches in programming are the instructions that lead to the execution of different segments of code, essentially the rules that define the decisions it makes. Branching is primarily driven by conditionals to guide which path to take. This is typically understood by the most common method of branching, "If This Then That."

IFTTT

If This Then That (IFTTT) asks a conditional statement then determines whether to take an action or to skip past it based on the conditional's output. There are 3 different variations on IFTTT that programs utilize:

IF

An IF statement is the most basic. It takes the form:

```
1. <Code up to IF statement>
2. IF <Conditional> THEN
3.     <Code in the IF statement>
4. <Code after the IFTTT Block>
```

When the code gets to the IF statement, if the Conditional is True (Line 2), the program performs the code in the IF statement (Line 3). If that Conditional is False, the program skips to the Code after the IFTTT Block (Line 4).

IF/ELSE

An IF/ELSE statement guarantees two possible paths, one if the conditional is False and one if the conditional is True. It takes the form:

```
1. <Code up to IF statement>
2. IF <Conditional> THEN
3.     <Code Segment 1>
4. ELSE
5.     <Code Segment 2>
6. <Code after the IFTTT Block>
```

When the code gets to the IF statement, if the Conditional is True (Line 2), the program performs the Code segment 1 (Line 3). If that Conditional is False, the program follows the ELSE option (Line 4) and performs the Code Segment 2 (Line 5). It will not perform both Code Segments. Once it has branched to either path, the program will then continue onto to the Code after the IFTTT Block (Line 6).

IF/ELSE IF/ELSE

An IF/ELSE IF/ELSE statement allows for multiple paths to occur. They contain multiple IF statements with associated segments of code for each. The program will run the code segment for the first conditional statement that is True and skip the other statements. It takes the form:

```
1. <Code up to IF statement>
2. IF <Conditional 1> THEN
3.     <Code Segment 1>
4. ELSE IF <Conditional 2> THEN
5.     <Code Segment 2>
6. ELSE
7.     <Code Segment 3>
8. <Code after the IFTTT Block>
```

When the code gets to the IF statement, it will check the first IF statement (Line 2). If conditional 1 is True, it runs Code Segment 1 (Line 3) and then skips to the Code after the IFTTT block (Line 6). If it False, the Conditional 2 (Line 4) is True, runs the associated code (Code Segment 2 - Line 5) and then skips to the Code after the IFTTT block. If Conditional 2 is False, it moves onto the next one and then the next, until there are no more ELSE IF statements involved. If none of the conditionals are True, then Code Segment 3 (Line 7) is run. The last ELSE statement is usually for the default or if none are true, and is not required.

Representation of Branching

Pseudocode

IF statements in Pseudocode are represented similar to the examples above. Like with Programs and Functions, its best to indicate when they end to make them easier to follow and it helps in translating to actual code later. See the Examples Below.

```
1. PROGRAM Grader
2. Check Exam Grade;
3. IF Grade is above 90 THEN
4.     Assign student an A;
5. ENDIF
6. Update Grade book;
7. END.
```

Note that in line 4, an indentation is used. Again, this helps with organizing the code, making it visually clear that the code is part of the IF statement. The ENDIF caps off the IFTTT block, also making clear what code is part of the IF statement and which is not.

Example for IF/ELSE.

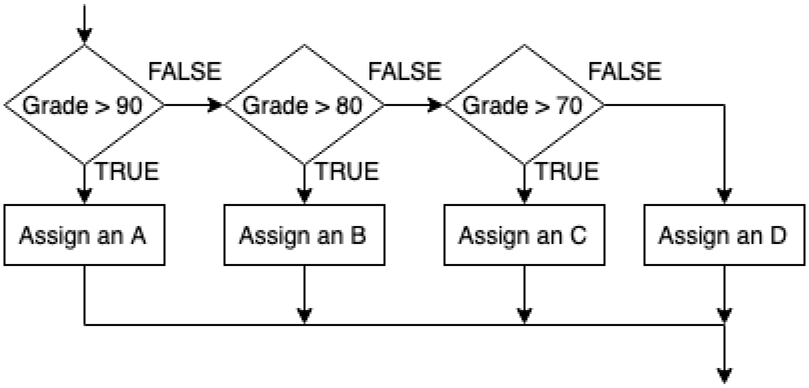
```
PROGRAM Grader
Check Exam Grade;
IF Grade is above 90 THEN
    Assign student an A;
ELSE
    Assign student a B;
ENDIF
Update Grade book;
END.
```

Example for IF/ELSE IF/ELSE.

```
PROGRAM Grader
Check Exam Grade;
IF Grade is above 90 THEN
    Assign student an A;
ELSE IF Grade is above 80 THEN
    Assign student a B;
ELSE IF Grade is above 70 THEN
    Assign student a C;
ELSE
    Assign student a D;
ENDIF
Update Grade book;
END.
```

Flowchart

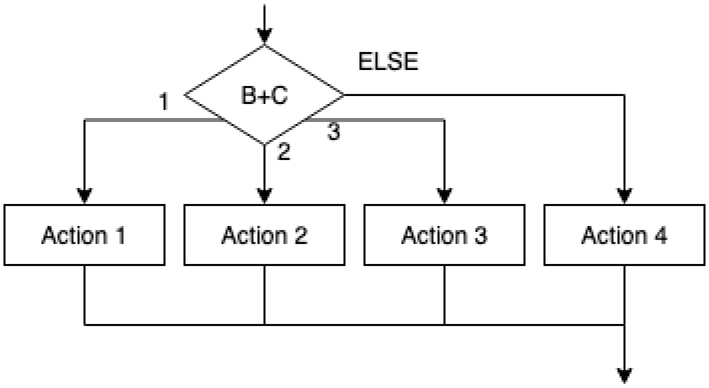
For IFTTT statements, flowcharts use decision symbols, the diamond. Below is an example using the same IF/ELSE IF/ELSE example from the Pseudocode. Note that each choice goes to another block or connection, be very careful of "dead ends", in programming, the sequential nature requires that each step have a next location or end.



The TRUE and FALSE do not need to be the only available answers though. Take the case of the following pseudocode snippet:

```
...
A = B + C
IF A == 1 THEN
    Action 1;
ELSE IF A == 2 THEN
    ACTION 2;
ELSE IF A == 3 THEN
    ACTION 3;
ELSE
    ACTION 4;
END IF
...
```

In this case, the series of conditionals can be collapsed into a single decision with multiple possible answers:



As usual, with representing these logical elements in either pseudocode or as a flowchart, as long as the logical processes are clear to the average reader, you have a lot of leeway. Understanding how to translate this into actual code is easier for young programmers if you maintain a simple, standard representation. For the above, you have to realize that this needs to be a series of IF/ELSE IF statements while the first example is more obviously so.

P1L: Tough concept Show Correct Answer Show Responses

A tough concept with IFTTT is that with an "IF this ELSE IF this ELSE IF this ... and so on" situation, the first If statement that's true is what will be executed and all of the following will be skipped. In the following situation, which will be executed:

```
...
B = 5;
C = -3;
A = B + C;
IF A >= 1 THEN
  Sit Down;
ELSE IF A == 2 THEN
  Stand Up;
ELSE IF A < 4 THEN
  DO A Jumping Jacks;
ELSE
  Do B Burpies;
END IF
...
```

A Jumping Jacks

B Burpies

C Sit Down

D Stand Up

E Does every action

Looping

This refers to methods that cause a snippet of code to repeat, or loop, based on some criteria.

While

Will loop until a specified conditional calculates **True**. If the condition is met initially, the loop will not begin. The subject of the Conditional should alter in some way during the loop otherwise the loop will continue forever. Sometimes an infinite loop is the intent.

```
...
WHILE <Conditional> THEN
    <Code in the WHILE Loop>
...
```

do/while

In this case, the first run of the loop is guaranteed since it does not ask the question WHILE <Conditional> until after the loop runs.

```
...
DO
    <Code in the WHILE Loop>
WHILE <Conditional>
...
```

For

These types specify a number of iterations, limiting the total number of loops initially. These are generally for loops that run a known number of times.

```
...
FOR each item in a list or array
    <Code in the WHILE Loop>
...
```

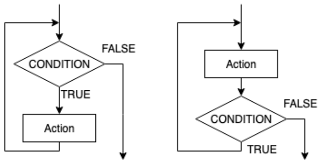
Representation of Looping

Pseudocode

Like with IFTTT statements, the pseudocode is represented similar to their general descriptions.

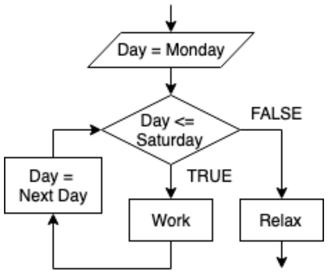
```
...
WHILE <Conditional> THEN
    <Code in the WHILE Loop>;
ENDWHILE
...
DO
    <Code in the WHILE Loop>;
WHILE <Conditional>
ENDWHILE
...
FOR each item in a list or array
    <Code in the WHILE Loop>;
ENDFOR
...
```

Flowchart



In flow charts, looping is indicated by having the logical path flow to a previous point in the flowchart's path. In the case of a WHILE loop, the loop should begin with the condition. For a DO WHILE loop, the condition should come after the action.

When representing a FOR loop, the iterating variable with its initial state should be passed into the loop and then iterated upon while in the loop.



P1L: Loops

Show Correct Answer

Show Responses

Try and match the loop type to the appropriate situation

Premise			Response	
1	While	→	A	Will run a pre-determined number of times
2	For	→	B	Will run an undetermined number of times
3	Do While	→	C	Will never runs
			D	Will run at least once

Functions

A function is a block of code written as a block or module that is crucial to organizing efficient code. The function will ONLY run when called. As an example, you use functions on your calculators to find the sum of numbers, or the mean, or the sine of an angle but only when you call it by pressing the appropriate buttons and give it values to use. All integrated development environments have some built-in functions that you will use quite often. You will write your own functions as well.

The format and handling is slightly different for each language but the basics are the same. You must assign it a name, declare what inputs should be expected, and what information and the type of information it will return. Custom functions are typically intended for tasks you have to repeat with some regularity. For instance, if you were writing a program for a game, you might have a function to check for a winning move or for checking character health that you run after each player move. That would take many lines of code for those tasks in many places in your program, so instead of just copy/pasting that snippet of code in each place, you can set it aside in a function and call that function each time you need it.

The following example may help illustrate how these work. It is written in a generic programming language format but is generally how the process works. There are variations depending on which programming language you are using. The function is called in Line 2 within the main program or another function and the function's code is defined in lines 4-9.

```
1.  ...
2.  AMOUNT_TO_PAY = TIP_CALCULATOR(X,Y);
3.  ...
4.  int TIP_CALCULATOR(float bill, int tip)
5.  {
6.      int Total_Paid;
7.      Total_Paid = ROUND(bill+(bill*(tip/100.0)));
8.      return Total_Paid;
9.  }
10. ...
```

Name

Functions must have a unique name so it can be recognized when called. In the example, we us `TIP_CALCULATOR()`. This is a bit lengthy but it is clearly unique. It also helps to describe the functions purpose, another useful best practice. Be careful to avoid basic names that may already have some underlying meaning in the code. Note that in line 7 there is a call for the function `ROUND()`. This is a common name for a function that rounds the number to the nearest integer. Watch out for common simple names, they've likely already been used. For instance, if you want a function to ***display*** a specific message, don't use the name `DISPLAY()`. That is actually a built-in function in most programming languages. This is your function so be more specific, like `DISPLAY_MY_MESSAGE()` or just `DISPMESSAGE()`.

Inputs

Functions may accept input parameters. Some do not and in those cases then the parentheses suffix will be left empty when the function is defined or called. In the example above, `TIP_CALCULATOR` is called with `X` and `Y` as parameters. Their information is then passed to the function as it runs. In the setup of the function, `int TIP_CALCULATOR(float bill, int tip)`, it identifies 2 parameters in the parenthesis. This is saying that the functions expects a float that it will refer to as bill and an int that it will refer to as tip. The function call will transfer the value of X into bill and the value of Y into tip, then proceed running the code inside the defined function.

Return

The final element of most functions is the return. This is the information that is outputted by the function. Line 2 actually calls the function while setting another variable equal to it. That variable will take on the value of the function's output. Notice in line 4 that when the function is defined, int is included at the beginning. This is stating that the output of this function will be an integer. If the function will have no output, then the type void is used. In the function, the command return is used to indicate which information should be returned or outputted by the function. The number of outputs allowed as well as how these are defined and handled changes between programming languages but the basic concepts are similar.

Representation of Functions

If you represent individual functions in pseudocode or a flowchart, make it clear that this is separate from the main program and identify what is input and output.

Pseudocode

Breaking out functions is not always necessary, especially if small, but this can help the planning and organization. They typically come out of necessity when writing the actual program when you start seeing patterns or similarities across multiple snippets of code and decide that you may be able to use a single snippet of code in multiple places with little alteration. You could have the line "`Calculate roots using quadratic formula`" in a few places in your pseudocode and that would be sufficient. We would have learned the process for this in our HS algebra courses. In programming though, that process would end up being quite a few lines of code, something straightforward but lengthy. You might decide during implementation that this deserves to be a function during this planning phase.

If you do predetermine the need for a function, make clear that it is separate from the program and, in your program, make clear that you are calling that function. Everything inside of it you treat the same as other programming elements. In the main program portion of your pseudocode, make clear you are calling that function and what information you want it to use.

In the function, make clear what information its expecting to have passed in and what information it will be passing out. You don't necessarily need either, functions can be called to perform some action without passing along data or taking data from them.

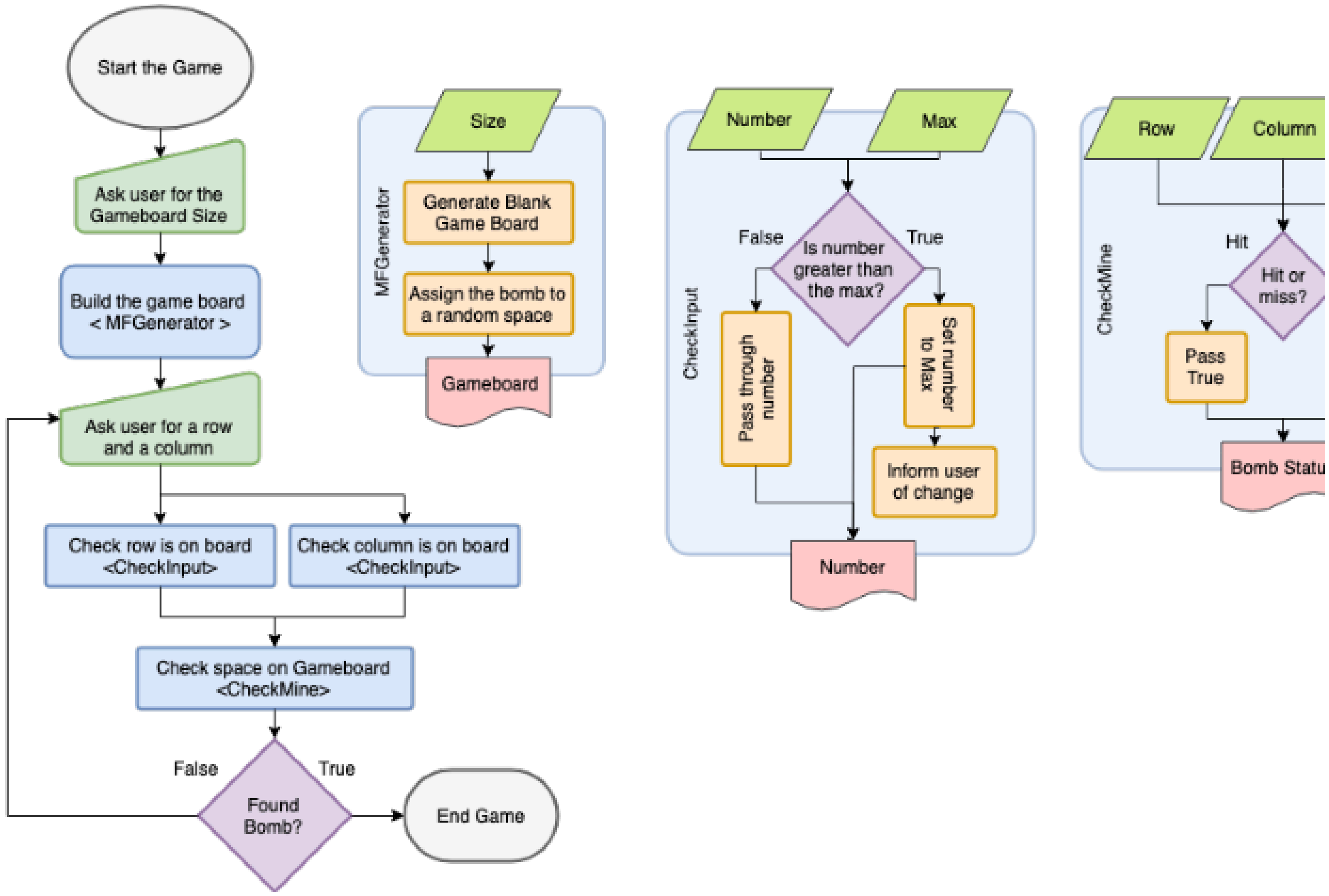
Example:

```
PROGRAM <ProgramName>:
...
Call <FunctionName> WITH <DATA>;
...
END.
FUNCTION <FunctionName>:
PASS IN: <DATA>
```

8. ...
9. PASS OUT: <NEW DATA>
10. END.

Flowcharts

Similar to pseudocode, including functions in flowcharts is not always required. Also similar to pseudocode, make sure to be clear that you are including functions. This is particularly important here because things are typically connected through flowlines but that wouldn't be the case with a separate function. See how this is achieved in the example below for a simple mine finding game.



Start the Game
Ask user for the Gameboard Size
Build the game board < MFGenerator >
Ask user for a row and a column
Check row is on board <CheckInput>
Check column is on board <CheckInput>
Check space on Gameboard <CheckMine>
False
True
Found Bomb?
End Game
MFGenerator
Size
Generate Blank Game Board
Assign the bomb to a random space
Gameboard
CheckInput
Number
Max
False
True
Is number greater than the max?
Pass through number
Set number to Max
Inform user of change
Number
CheckMine
Row
Column
Hit
Hit or miss?
Pass True
Bomb Statu

The symbols that call the function include <'Name of Function'> to reference the functions. Each function flowchart has been labelled as well as made clear what its inputs and outputs are. Color is even used to help make the logic and flow clear. This is also an example of the freedom allowed with this, if color will help communicate the ideas, go ahead and use it.

Conclusion and Review

Topics covered in this chapter:

- Proper Programming Practice
- Representing and Planning Programs
- Common Programming Elements

By reading through this chapter, you should be prepared for writing programs and understanding the logic of programs. Key to take away are the tools for planning programs, good planning and thinking ahead of writing any code will create well-written, efficient and effective programs. Knowledge of the building blocks of programs from this chapter gives you a working structure for programming in any language. You can quickly navigate to any topic by using Top Hat's Index feature while viewing it in Fullscreen mode. Happy programming!

Question ✓ 100% Correct 31/31

In an IF/ELSE IF/ELSE programming statement, what happens if none of the conditionals are true?

?

A	The program executes all the code segments provided.
B	The ELSE statement code segment is run.
C	The program stops running.
D	The program runs only the first code segment.

HINT

EXPLANATION

Show Responses Show Answer

P1L: Concept Check 3 ✓ 100% Correct 31/31

What does the round() function, commonly used in programming, typically do?

?

A	Displays a specific message
B	Calculates the roots using the quadratic formula
C	Rounds the number to the nearest integer
D	No specific operation, the name is too common

HINT

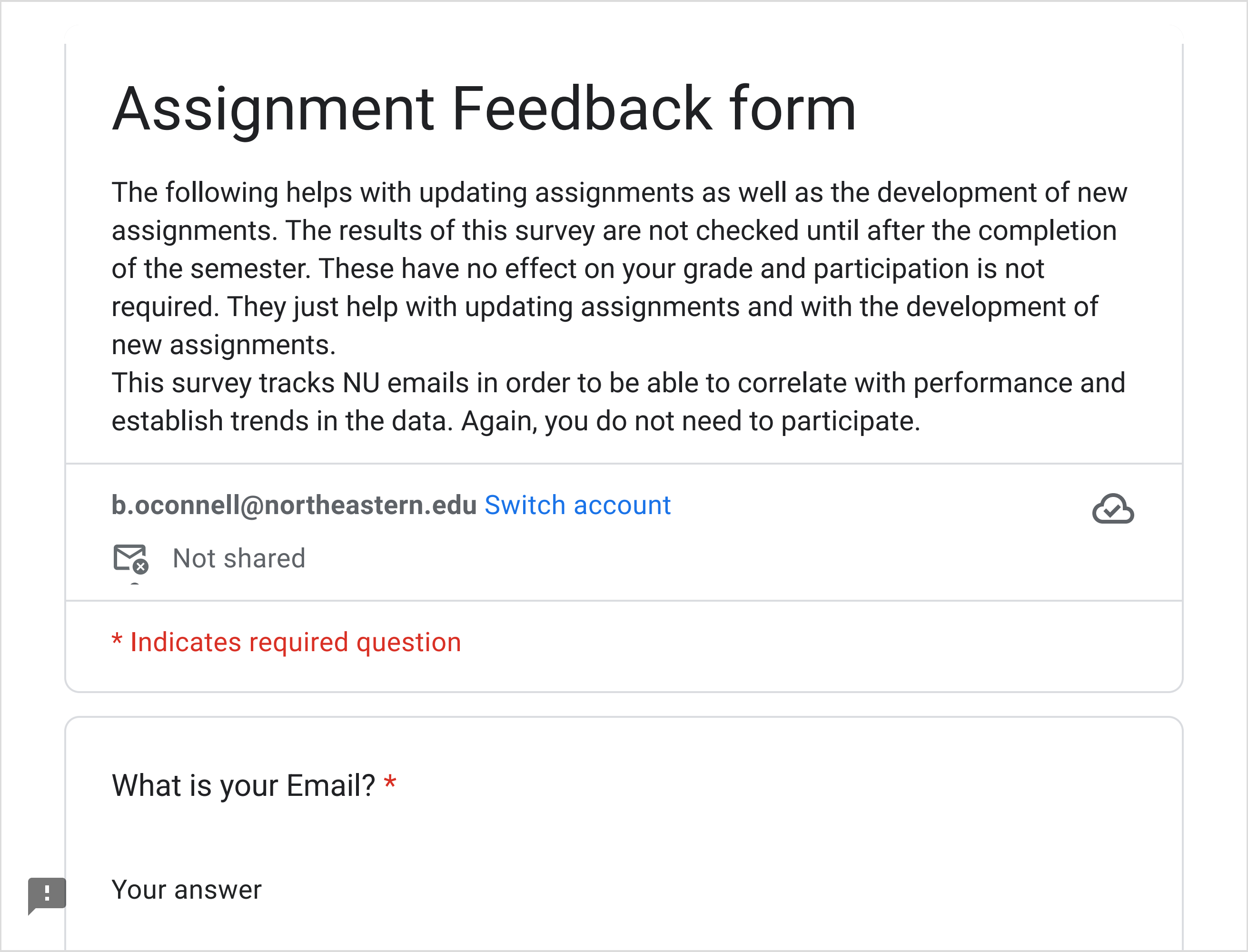
EXPLANATION

Show Responses Show Answer

Image Credits

All images provided by the authors via draw.io

The following survey is for my reference to help improve future assignments. The results are not checked until after the semester is completed and participation is not required.



Assignment Feedback form

The following helps with updating assignments as well as the development of new assignments. The results of this survey are not checked until after the completion of the semester. These have no effect on your grade and participation is not required. They just help with updating assignments and with the development of new assignments.
This survey tracks NU emails in order to be able to correlate with performance and establish trends in the data. Again, you do not need to participate.

b.oconnell@northeastern.edu Switch account

Not shared

* Indicates required question

What is your Email? *

Your answer

Exported for Brian O'Connell on Tue, 27 May 2025 19:21:35 GMT