1.1 What is Computer Science?

Here are some important questions that you should be able to answer by the end of this module.

1. What is computer science (CS)?
2. What is a computer?
3. What is an algorithm?
4. What is programming?
5. What is a program?
6. What is a programming language?
7. What is Java?
8. What is a Java program?
9. How is CS and programming different from other endeavors?
10. What are the expectations for this course?

Figure 1.1.1: Shall We Play a Game? (2:58)

2:59

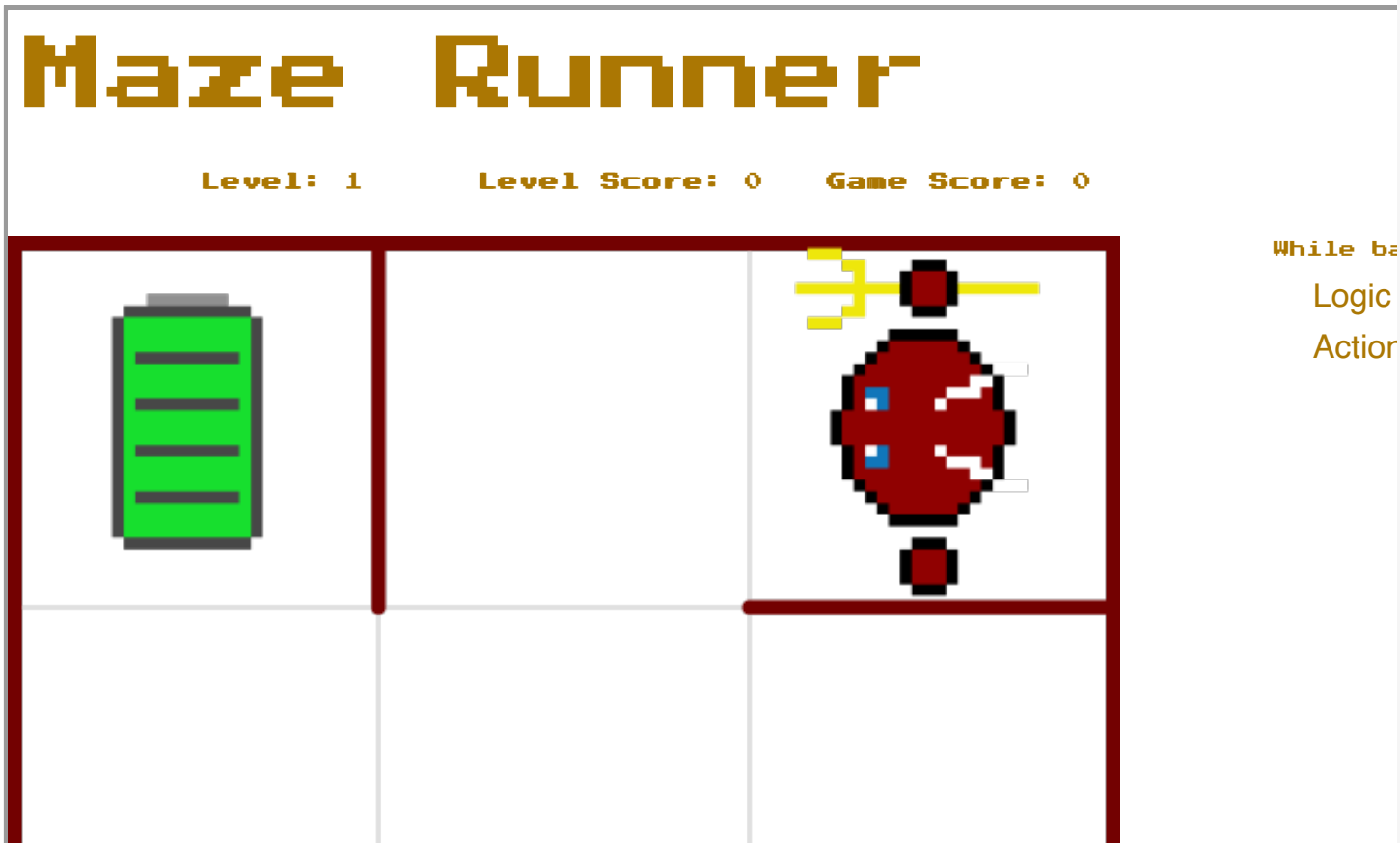
Maze Runner

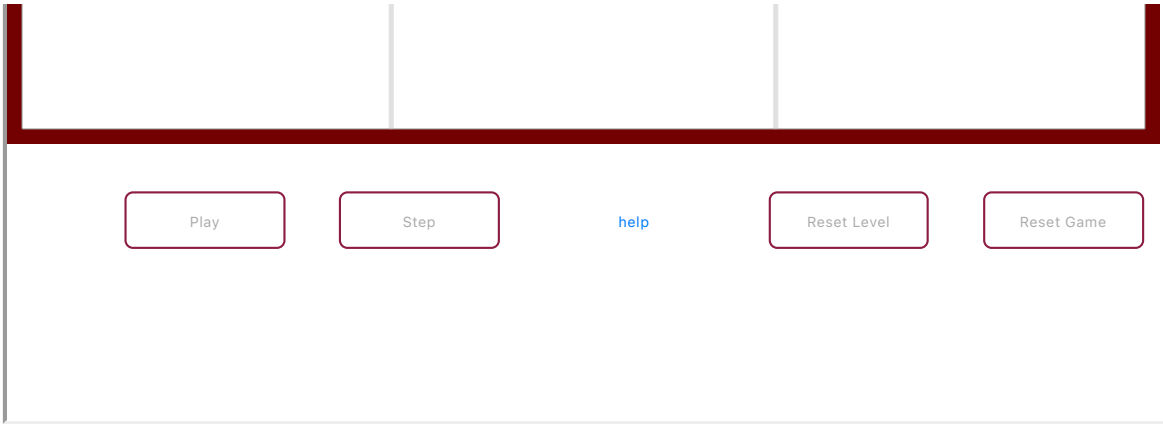
Context

Sparky the robot has a low battery and must be given instructions on how to find the charging station (green battery icon), before the robot's battery drains completely.

Be sure to spend some time playing with the interactive puzzle below before moving on to the sections below. Just dive right in and explore. After you have completed several levels, then move on to the content below.

Figure 1.1.2: Maze Runner



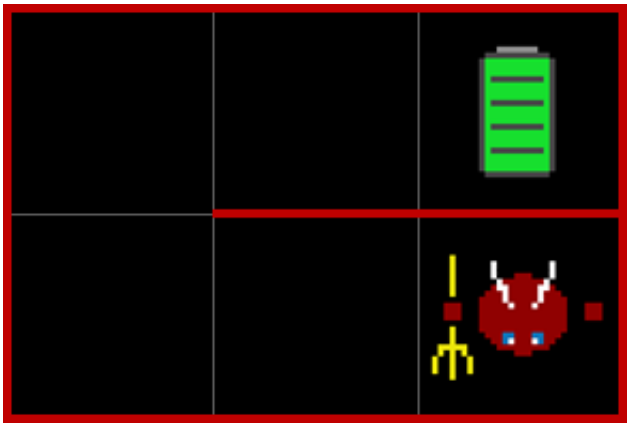


Considerations

Now that you have spent some time exploring the interactive Maze Runner puzzle, here are some things to consider.

Notice that, for any given maze, it is fairly easy to compose a specific list of commands that the robot can execute to reach the charging station. For example, given a maze like this one:

Figure 1.1.3: Maze Runner Maze 01



You can solve this maze by giving the robot a list of commands like this:

Turn Right
Go Forward
Go Forward
Turn Right
Go Forward

Turn Right
Go Forward
Go Forward

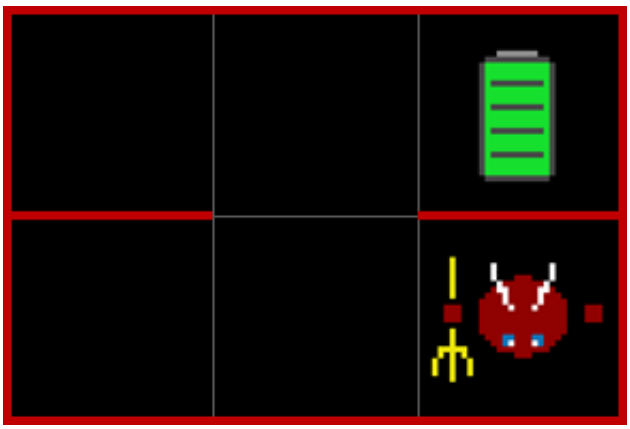
Sequence

You can solve all mazes in this manner - by listing the specific sequence of instructions required for the robot to directly maneuver to the charging station. In Computer Science we call any such list of instructions an **algorithm**.

Algorithm
A list of instructions that, when followed, solves a problem

Now consider a slightly different maze like this one:

Figure 1.1.4: Maze Runner Maze 02



This new maze cannot be solved by the same list of commands that we used above to solve the previous maze. The reason for this is the fact that our previous algorithm is specialized to solve just that one maze. As we mentioned above, we can solve this new maze by composing a new algorithm that is specialized to solve this particular maze. For example, following the sequence of commands below will get Sparky to the charging station in this new maze:

Turn Right
Go Forward
Turn Right

Turn Right
Go Forward
Turn Right
Go Forward

However, as you no doubt discovered as you played with the interactive puzzle, there are many possible mazes, and composing a new specialized algorithm to solve each new maze is time consuming and error prone. This may lead us to ask a question. Is it possible to design and construct an algorithm that will solve more than one maze - perhaps to solve all mazes?

Did you ask this question as you were playing with the Maze Runner puzzle? Did you discover such an algorithm? The game will reward you for such a discovery by doubling your score. If you did discover an algorithm which was able to solve more than one maze, then what things do you notice about the differences between such a generalized algorithm and the specialized algorithms that can solve only one maze?

If you did not discover such a generalized algorithm (one that can solve more than on maze), then I encourage you to go back to the puzzle and explore some more. It may be helpful to think about the problem from the robot's perspective. It is easy for us to create a specialized algorithm to solve a specific maze because we can see the whole maze at once. The robot can only see its immediate surroundings - Is there a wall in front of me? Is there an opening to my right? , etc. Go back to the puzzle and see if you can design and construct an algorithm that can be used to solve more than just one specific maze.

After you have made another attempt to design a generalized algorithm that solves more than one specific maze, and you have put some time and effort into thinking about the differences between a generalized algorithm and a specialized algorithm, imagine how you would explain the difference to a friend or family member.

Decision

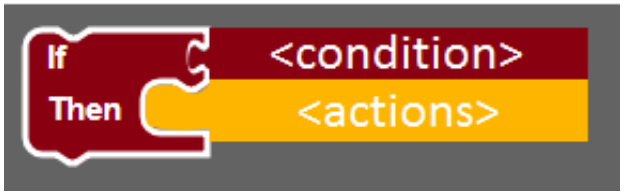
As you most likely discovered, Just following a simple sequence of commands will not do. The key to generalized algorithms for solving mazes is to get the robot to make some decisions. The ability to make decisions is the source of most of the power and flexibility of modern computers.

It is important to realize that, while computers (or robots) can make decisions, they do so without any comprehension or understanding. No matter what list of instructions we give to Sparky the robot, it will never comprehend or understand that it is navigating a path through a maze. It will never even know what a maze is. It will simply and unquestioningly follow the instructions that it is given. It is we - the programmers - who must understand and design and compose an algorithm to solve the problem.

We can use the **If-Then** block to make decisions in our maze solving algorithms for the interactive puzzle. The **If-Then** block has two components - a condition and a set of one or more actions

puzzle. The **If-Then** block has two components - a condition and a set of one or more actions.

Figure 1.1.5: Maze Runner Decision Block 01



The condition is always something that is either **True** or **False**. For example, the condition "**Right is Open**" is **True** when there is an opening (not a wall) on the right hand side of Sparky. If there is a wall on Sparky's right hand side, then the condition "**Right is Open**" is **False**.

Figure 1.1.6: Maze Runner Decision Block 02

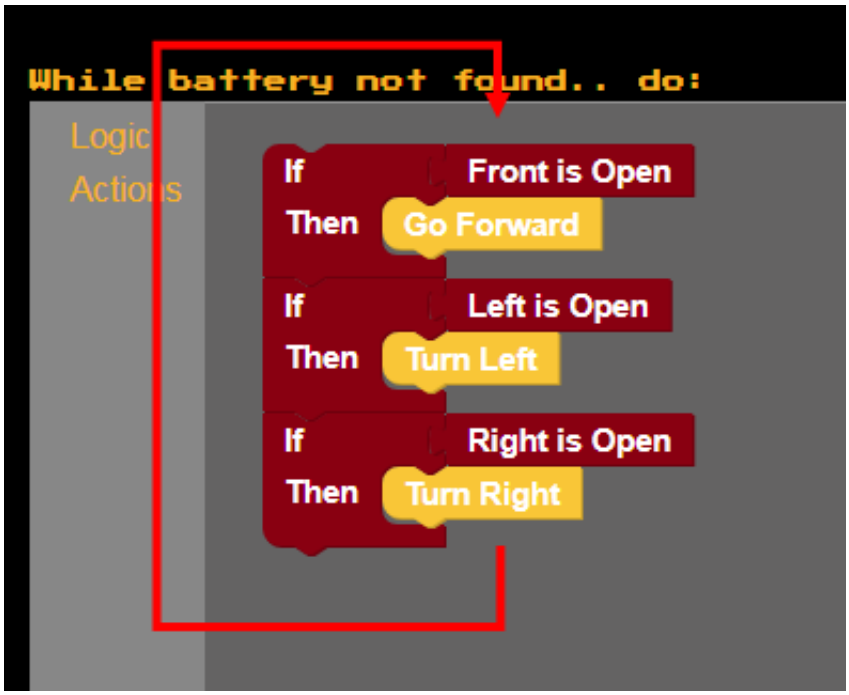
If the condition is **True**, then the actions of the **If-Then** block will be executed. In the case of the example above - if it is **True** that the "**Right is Open**", then the command "**Turn Right**" will be executed by the robot. And, if the condition is **False**, then the actions of the **If-Then** block will be skipped.

Repetition

One last thing to notice about the Maze Runner interactive puzzle is the fact that, after the last instruction in the list is executed, it goes back up to the first instruction and begins to execute the sequence again. In other words, it repeats the sequence of commands and decisions. This ability to repeat some instructions is another powerful mechanism for designing and composing algorithms to solve problems.

Figure 1.1.7: Maze Runner Repetition Block 01

Figure 1.1.7: Maze Runner Repetition Block 01



Structure

In fact, it may surprise you to know that all algorithms - **anything that any computer can be programmed to do - can be composed out of nothing more than sequences, decisions, and repetitions**. We will spend considerable time in this course exploring these three logic structures and ways in which we can combine them to design and construct algorithms to solve problems.

Adventure

Now you have already started your journey into the world of computer science and programming. Many challenges and adventures await you in the days and weeks to come. You will not likely make it through this without some struggles - the struggles are an essential part of learning the powerful concepts and skills that this course covers. Many students find this class to be one of the most challenging. Many students find it to be one of the most rewarding. Our hope is that you find this course to be both :)

1.2 Computers and programs (general)

Figure 1.2.1: Looking under the hood of a car.

Source: zyBooks

Just as knowing how a car works "under the hood" has benefits to a car owner, knowing how a computer works under the hood has benefits to a programmer. This section provides a very brief introduction.

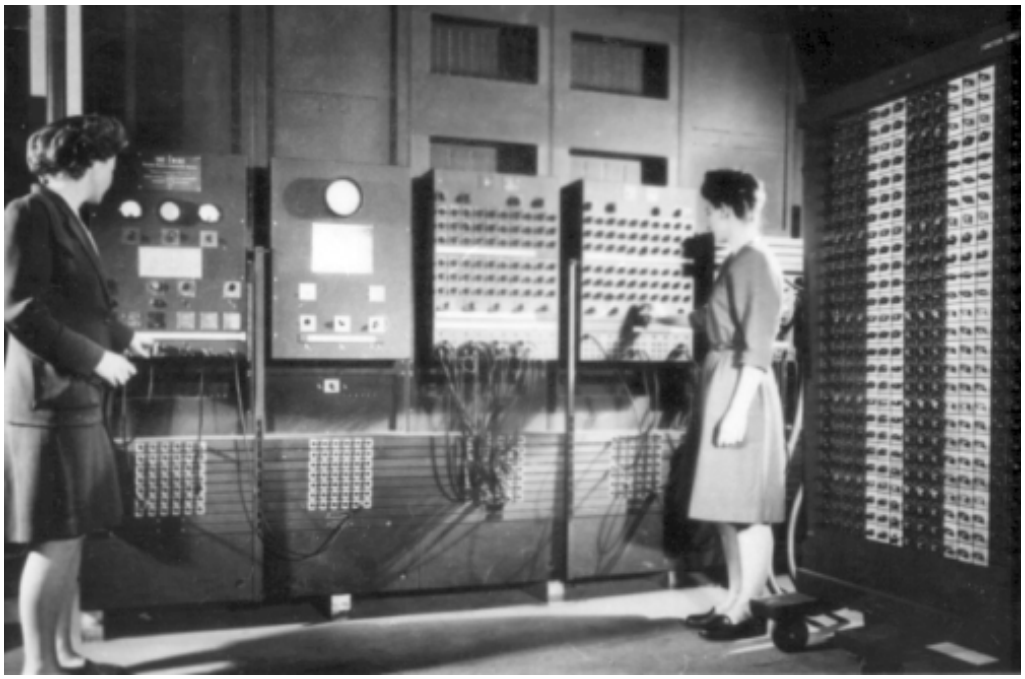
Switches

When people in the 1800s began using electricity for lights and machines, they created switches to turn objects on and off. A *switch* controls whether or not electricity flows through a wire. In the early 1900s, people created special switches that could be controlled electronically, rather than by a person moving the switch up or down. In an electronically controlled switch, a positive voltage at the control input allows electricity to flow, while a zero voltage prevents the flow. Such switches were useful, for example, in routing telephone calls. Engineers soon realized they could use electronically controlled switches to perform simple calculations. The engineers treated a positive voltage as a "1" and a zero voltage as a "0". 0s and 1s are known as ***bits*** (*b*inary dig*its*). They built connections of switches, known as *circuits*, to perform calculations such as multiplying two numbers.

PARTICIPATION | 1.2.1: A bit is either 1 or 0, like a light switch is either on or off (click the

ACTIVITY switch).

Figure 1.2.2: Early computer made from thousands of switches.



Source: ENIAC computer (U. S. Army Photo / Public domain)

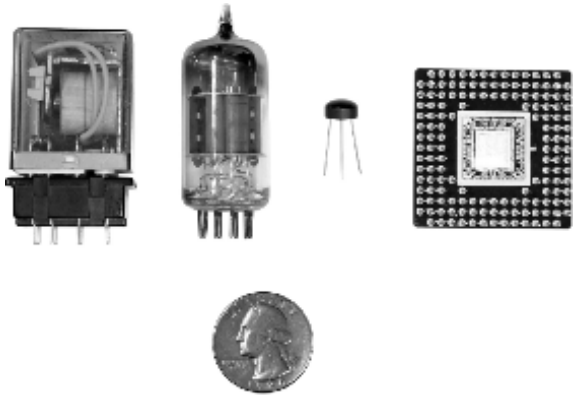
These circuits became increasingly complex, leading to the first electronic computers in the 1930s and 1940s, consisting of about ten thousand electronic switches and typically occupying entire rooms as in the above figure. Early computers performed thousands of calculations per second, such as calculating tables of ballistic trajectories.

Processors and memory

To support different calculations, circuits called ***processors*** were created to process (aka *execute*) a list of desired calculations, with each calculation called an ***instruction***. The instructions were

a list of desired calculations, with each calculation called an ***instruction***. The instructions were specified by configuring external switches, as in the figure above. Processors used to take up entire rooms but today fit on a chip about the size of a postage stamp, containing millions or even billions of switches.

Figure 1.2.3: As switches shrunk, so did computers. The computer processor chip on the right has millions of switches.



Source: zyBooks

Instructions are stored in a memory. A ***memory*** is a circuit that can store 0s and 1s in each of a series of thousands of addressed locations, like a series of addressed mailboxes that each can store an envelope (the 0s and 1s). Instructions operate on data, which is also stored in memory locations as 0s and 1s.

Figure 1.2.4: Memory.

Thus, a computer is basically a processor interacting with a memory, as depicted in the following example. In the example, a computer's processor executes program instructions stored in memory, also using the memory to store temporary results. The example program converts an hourly wage ($20/hr) into an annual salary by multiplying by 40 (hours/week) and then by 50 (weeks/year), outputting the final result to the screen.

PARTICIPATION ACTIVITY 1.2.2: Computer processor and memory.

Animation captions:

1. The processor computes data, while the memory stores data (and instructions).
2. Previously computed data can be read from memory.
3. Data can be output to the screen.

The arrangement is akin to a chef (processor) who executes instructions of a recipe (program), each instruction modifying ingredients (data), with the recipe and ingredients kept on a nearby counter (memory).

Instructions

Below are some sample types of instructions that a processor might be able to execute, where *X*, *Y*, *Z*, and *num* are each an integer.

Table 1.2.1: Sample processor instructions.

Add X, #num, Y	Adds data in memory location *X* to the number *num*, storing result in location *Y*.
Sub X, #num, Y	Subtracts *num* from data in location *X*, storing result in location *Y*.
Mul X, #num, Y	Multiplies data in location *X* by *num*, storing result in location *Y*.
Div X, #num, Y	Divides data in location *X* by *num*, storing result in location *Y*.
Jmp Z	Tells the processor that the next instruction to execute is in memory location *Z*.

For example, the instruction "Mul 97, #9, 98" would multiply the data in memory location 97 by the number 9, storing the result into memory location 98. So if the data in location 97 were 20, then the instruction would multiply 20 by 9, storing the result 180 into location 98. That instruction would actually be stored in memory as 0s and 1s, such as "011 1100001 001001 1100010", where 011 specifies a multiply instruction and 1100001, 001001, and 1100010 represent 97, 9, and 98 (as described previously). The following animation illustrates the storage of instructions and data in memory for a program that computes F = (9*C)/5 + 32, where C is memory location 97 and F is memory location 99.

PARTICIPATION ACTIVITY 1.2.3: Memory stores instructions and data as 0s and 1s.

Animation captions:

1. Memory stores instructions and data as 0s and 1s.
2. The material will commonly draw the memory with the corresponding instructions and data to improve readability.

The programmer-created sequence of instructions is called a ***program***, ***application***, or just ***app***.

When powered on, the processor starts by executing the instruction at location 0, then location 1, then location 2, etc. The above program performs the calculation over and over again. If location 97 is connected to external switches and location 99 to external lights, then the computer programmer (like the women operating the ENIAC computer in the earlier picture) could set the switches to represent a particular Celsius number, and the computer would automatically output the Fahrenheit number using the lights.

PARTICIPATION ACTIVITY 1.2.4: Processor executing instructions.

Animation captions:

1. The processor starts by executing the instruction at location 0.
2. The processor next executes the instruction at location 1, then location 2. 'Next' keeps track of the location of the next instruction.
3. The Jmp instruction indicates that the next instruction to be executed is at location 0, so 0 is assigned to 'Next'.
4. The processor executes the instruction at location 0, performing the same sequence of

4. The processor executes the instruction at location 0, performing the same sequence of instructions over and over again.

PARTICIPATION ACTIVITY 1.2.5: Computer basics.

1) A bit can only have the value of 0 or 1.
   - True
   - False

2) Switches have gotten larger over the years.
   - True
   - False

3) A memory stores bits.
   - True
   - False

4) The computer inside a modern smartphone would have been huge in the 1980s.
   - True
   - False

5) A processor executes instructions like Add 200, #9, 201, represented as 0s and 1s.
   - True
   - False

Writing computer programs

In the 1940s, programmers originally wrote each instruction using 0s and 1s, such as "001 1100001 001001 1100010". Instructions represented as 0s and 1s are known as ***machine instructions***, and a sequence of machine instructions together form an ***executable program***

(sometimes just called an *executable*). Because 0s and 1s are hard to comprehend, programmers soon created programs called *assemblers* to automatically translate human readable instructions, such as "Mul 97, #9, 98", known as ***assembly*** language instructions, into machine instructions. The assembler program thus helped programmers write more complex programs.

In the 1960s and 1970s, programmers created ***high-level languages*** to support programming using formulas or algorithms, so a programmer could write a formula like: `F = (9 /5 ) * C + 32`. Early high-level languages included *FORTRAN* (for "Formula Translator") or *ALGOL* (for "Algorithmic Language"), which were more closely related to how humans thought than were machine or assembly instructions.

To support high-level languages, programmers created ***compilers***, which are programs that automatically translate high-level language programs into executable programs.

PARTICIPATION ACTIVITY 1.2.6: Program compilation and execution.

Animation captions:

1. A programmer writes a high-level program.
2. The programmer runs a compiler, which converts the high-level program into an executable program.
3. Users can then run the executable.

Using the above approach, an executable can only run on a particular processor type (like an x86 processor); to run a program on multiple processor types, the programmer must have the compiler generate multiple executables. Some newer high-level languages like Java use an approach that allows the same executable to run on different processor types. The approach involves having the compiler generate an executable using machine instructions of a "virtual" processor, such an executable is sometimes called ***bytecode***. Then, the real processor runs a program, sometimes called a ***virtual machine***, that executes the instructions in the bytecode. Such an approach may yield slower program execution, but has the advantage of portable executables.

PARTICIPATION ACTIVITY 1.2.7: Programs.

Assembly language **Application** **Machine instruction** **Compiler**

	Translates a high-level language program into low-level machine instructions.
	Another word for program.
	A series of 0s and 1s, stored in memory, that tells a processor to carry out a particular operation like a multiplication.
	Human-readable processor instructions.

Reset

Note (mostly for instructors): Why introduce machine-level instructions in a high-level language book? Because a basic understanding of how a computer executes programs can help students master high-level language programming. The concept of sequential execution (one instruction at a time) can be clearly made with machine instructions. Even more importantly, the concept of each instruction operating on data in memory can be clearly demonstrated. Knowing these concepts can help students understand the idea of assignment (x = x + 1) as distinct from equality, why x = y; y = x does not perform a swap, what a pointer or variable address is, and much more.

1.3 Computer tour

The term *computer* has changed meaning over the years. The term originally referred to a person that performed computations by hand, akin to an accountant ("We need to hire a computer.") In the 1940s/1950s, the term began to refer to large machines like in the earlier photo. In the 1970s/1980s, the term expanded to also refer to smaller home/office computers known as personal computers or PCs ("personal" because the computer wasn't shared among multiple users like the large ones) and to portable/laptop computers. In the 2000s/2010s, the term may also cover other computing devices like pads, book readers, and smart phones. The term computer even refers to computing devices embedded inside other electronic devices such as medical equipment, automobiles, aircraft, consumer electronics, military systems, etc.

In the early days of computing, the physical equipment was prone to failures. As equipment

became more stable and as programs became larger, the term *software* became popular to distinguish a computer's programs from the *hardware* on which they ran.

A computer typically consists of several components (see animation below):

- ***Input/output devices***: A ***screen*** (or monitor) displays items to a user. The above examples displayed textual items, but today's computers display graphical items, too. A ***keyboard*** allows a user to provide input to the computer, typically accompanied by a *mouse* for graphical displays. Keyboards and mice are increasingly being replaced by *touchscreens*. Other devices provide additional input and output means, such as microphones, speakers, printers, and USB interfaces. I/O devices are commonly called *peripherals*.
- ***Storage***: A ***disk*** (aka *hard drive*) stores files and other data, such as program files, song/movie files, or office documents. Disks are *non-volatile*, meaning they maintain their contents even when powered off. They do so by orienting magnetic particles in a 0 or 1 position. The disk spins under a head that pulses electricity at just the right times to orient specific particles (you can sometimes hear the disk spin and the head clicking as the head moves). New *flash* storage devices store 0s and 1s in a non-volatile memory, rather than disk by tunneling electrons into special circuits on the memory's chip and removing them with a "flash" of electricity that draws the electrons back out.
- ***Memory***: ***RAM*** (random-access memory) temporarily holds data read from storage and is designed such that any address can be accessed much faster than disk, in just a few clock ticks (see below) rather than hundreds of ticks. The "random access" term comes from being able to access any memory location quickly and in arbitrary order, without having to spin a disk to get a proper location under a head. RAM is costlier per bit than disk, due to RAM's higher speed. RAM chips typically appear on a printed-circuit board along with a processor chip. RAM is volatile, losing its contents when powered off. Memory size is typically listed in bits or in bytes, where a ***byte*** is 8 bits. Common sizes involve megabytes (million bytes), gigabytes (billion bytes), or terabytes (trillion bytes).
- ***Processor***: The ***processor*** runs the computer's programs, reading and executing instructions from memory, performing operations, and reading/writing data from/to memory. When powered on, the processor starts executing the program whose first instruction is (typically) at memory location 0. That program is commonly called the *BIOS (basic input/output system)*, which sets up the computer's basic peripherals. The processor then begins executing a program called an *operating system (OS)*. The ***operating system*** allows a user to run other programs and interfaces with the many other peripherals. Processors are also called *CPUs* (central processing units) or *microprocessors* (a term introduced when processors began fitting on a single chip, the "micro-" suggesting something small). Because speed is so important, a processor may contain a small amount of RAM on its own chip, called ***cache*** memory, accessible in one clock tick rather than several, for maintaining a copy of the most-used instructions/data.

- ***Clock***: A processor's instructions execute at a rate governed by the processor's ***clock***, which ticks at a specific frequency. Processors have clocks that tick at rates such as 1 MHz (1 million ticks/second) for an inexpensive processor ($1) like those found in a microwave oven or washing machine, to 1 GHz (1 billion ticks/second) for costlier ($10-$100) processors like those found in mobile phones and desktop computers. Executing about 1 instruction per clock tick, processors thus execute millions or billions of instructions per second.

Computers typically run multiple programs simultaneously, such as a web browser, an office application, a photo editing program, etc. The operating system actually runs a little of program A, then a little of program B, etc., switching between programs thousands of times a second.

PARTICIPATION ACTIVITY 1.3.1: Some computer components.

Animation captions:

1. A disk is able to store terabytes of data and may contain various programs such as ProgA, ProgB, Doc1, Doc2, and OS. The memory is able to store Gigabytes of data. User runs ProgA. The disk spins and the head loads ProgA from the disk, storing the contents into memory.
2. The OS runs ProgB. The disk spins and the head loads ProgB from the disk, storing the contents into memory.
3. The OS lets ProgA run again. ProgA is already in memory, so there is no need to read ProgA from the disk.

After computers were invented and occupied entire rooms, engineers created smaller switches called ***transistors***, which in 1958 were integrated onto a single chip called an ***integrated circuit***, or IC. Engineers continued to make transistors smaller, leading to ***Moore's Law***: the doubling of IC capacity roughly every 18 months, which continued for several decades.

Note: Moore actually said every 2 years. And the actual trend has varied from 18 months. The key is that doubling occurred roughly every two years, causing much improvement over time. Intel: Moore's Law.

By 1971, Intel produced the first single-IC processor named the 4004, called a *microprocessor* (*micro-* suggesting something small), having 2,300 transistors. New, more powerful microprocessors appeared every few years, and by 2012, a single IC had several *billion* transistors containing multiple processors (each called a *core*).

PARTICIPATION

ACTIVITY 1.3.2: Programs.

Disk | Clock | Moore's Law | RAM | Operating system | Cache

	Manages programs and interfaces with peripherals.
	Nonvolatile storage with slower access.
	Volatile storage with faster access usually located off processor chip.
	Relatively small volatile storage with fastest access, which is located on the processor chip.
	Rate at which a processor executes instructions.
	The doubling of IC capacity roughly every 18 months.

Reset

A side note: A common way to make a PC faster is to add more RAM. A processor spends much of its time moving instructions/data between memory and storage, because not all of a program's instructions/data may fit in memory—akin to a chef who spends most of his/her time walking back and forth between a stove and pantry. Just as adding a larger table next to the stove allows more ingredients to be kept close by, a larger memory allows more instructions/data to be kept close to the processor. Moore's Law results in RAM being cheaper a few years after buying a PC, so adding RAM to a several-year-old PC can yield good speedups for little cost.

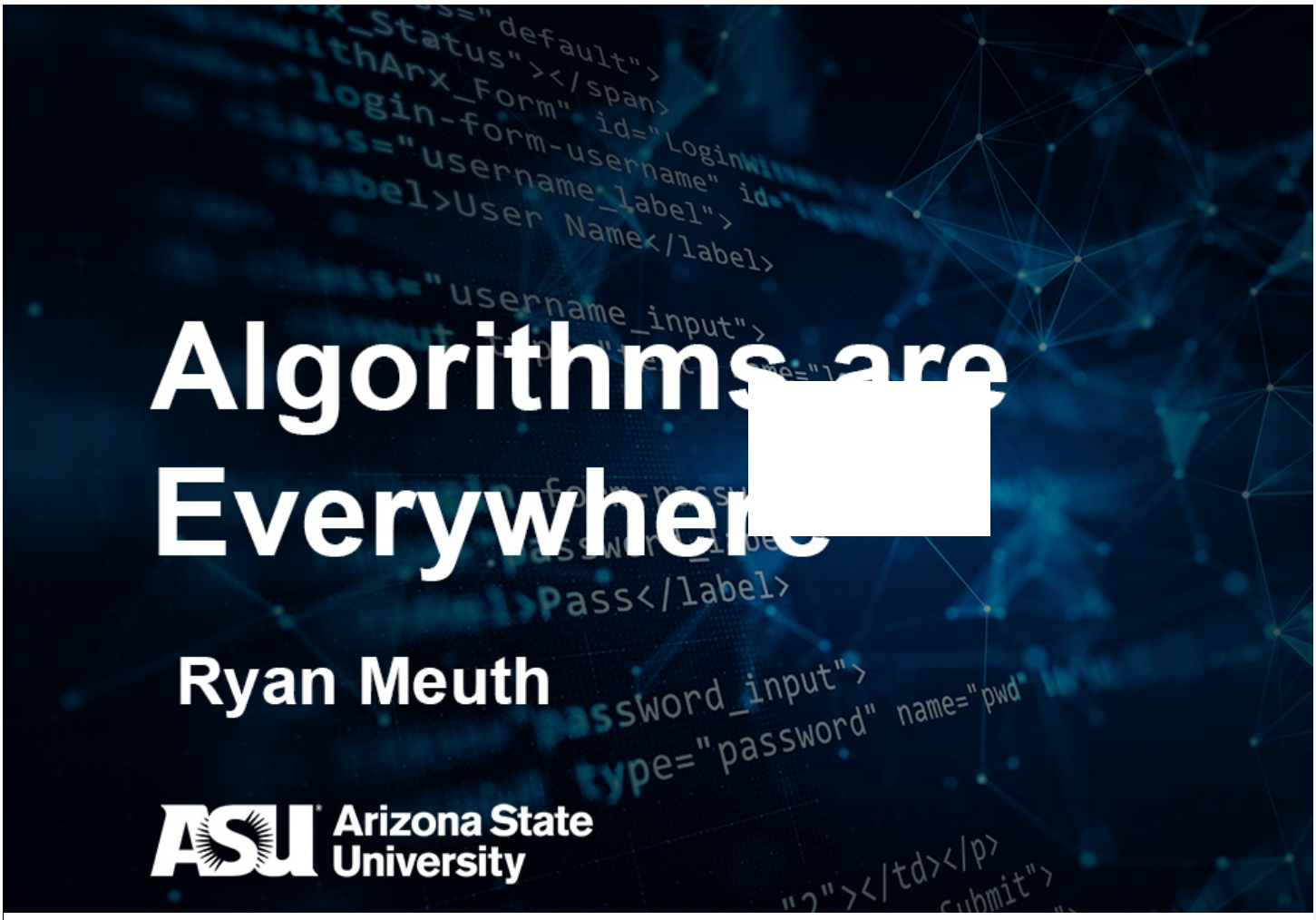
Exploring further:

- Video: Where's the disk/memory/processor in a desktop computer (20 sec).

- Link: What's inside a computer (HowStuffWorks.com)
- Video: How memory works (1:49)
- Link: How Microprocessors Work (HowStuffWorks.com)

1.4 What is an Algorithm?

Figure 1.4.1: Algorithms are Everywhere (1:10)



1:11

An **Algorithm** is:
A list of instructions that, when followed, solves a problem

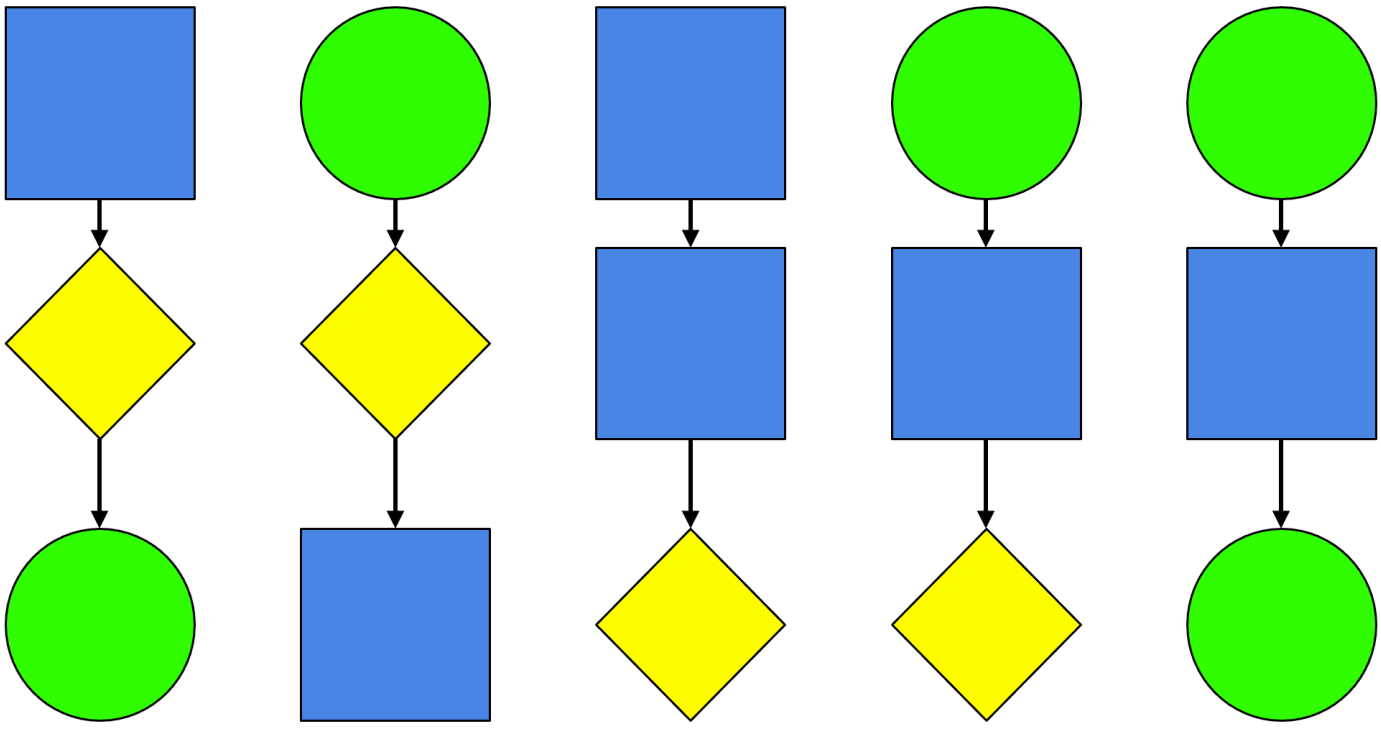
The basic elements of a structured algorithm are:

- **Sequence**: Instructions are followed in a specific order, top to bottom.
- **Decision**: If a condition is met, different subsets of instructions are followed.
- **Iteration**: Instructions are repeated while a condition is met. These are often referred to as 'loops'.

We can compose complex algorithms by:

- **Stacking**: Listing algorithm structures (instructions, decisions, and loop) in sequential order.

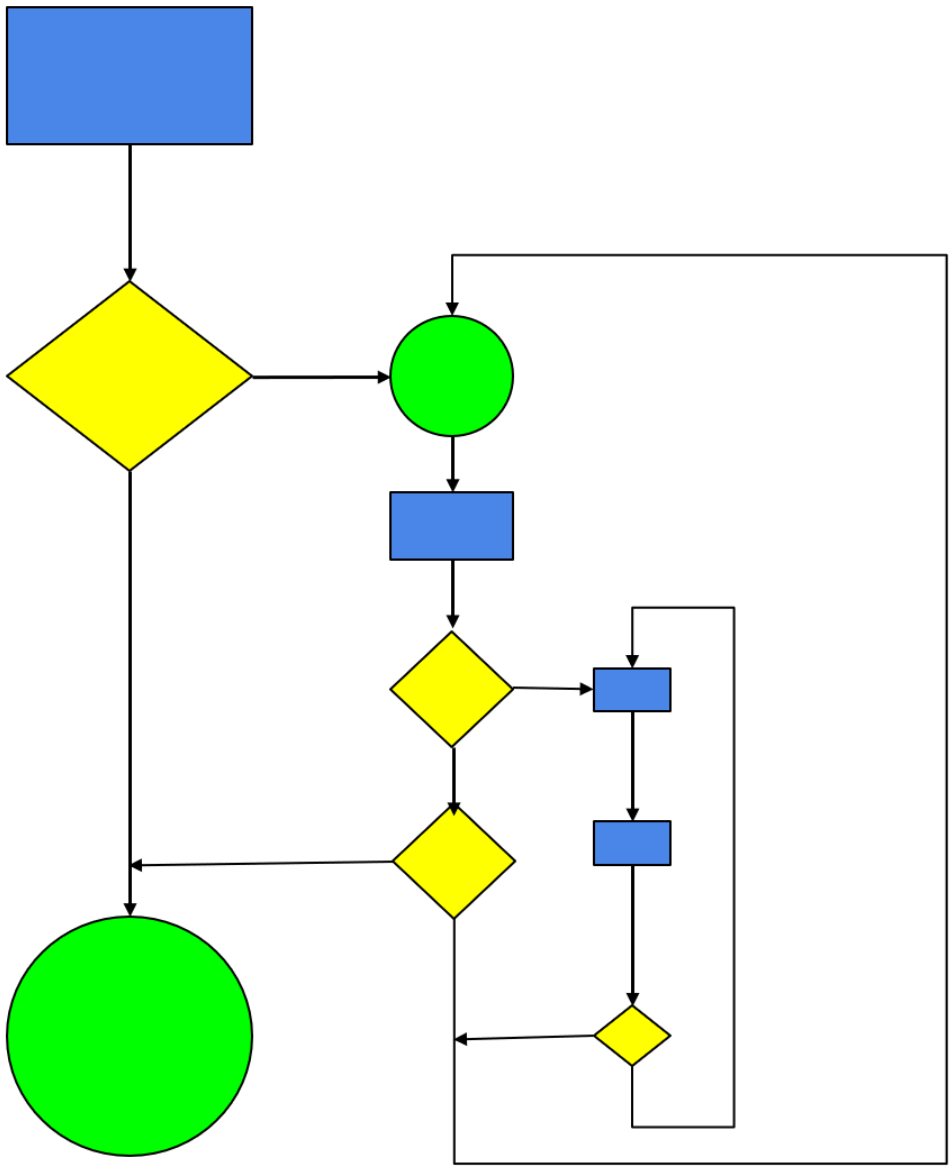
Figure 1.4.2: Stacking Structural Elements



- **Nesting**: Instructions, decisions, and loops can be listed as the target of decisions and loops.

This allows for exponential complexity.

Figure 1.4.3: Nesting Structural Elements



Pseudocode

One of the challenges that we face as computer scientists and as programmers is the need to specify our algorithms in a form that computers can follow. As we will learn, despite the apparent power of modern computers, these computers are fundamentally only capable of doing a few very

simple things - like addition and subtraction, and copying and comparing values. They possess no inherent intelligence. They do not know or understand anything.

Because of this limitation, we must express algorithms for computers in terms that are very detailed, complete, and absolutely unambiguous. Programming languages (like Python and Java) are designed to support and strictly enforce these requirements in terms of grammar rules and syntax. This means that it is very easy to make mistakes when writing in a programming language, and any mistakes means that the program will not run or will run incorrectly. This can be very frustrating when you are just learning to program or just learning a new programming language. It is often even frustrating for experienced programmers. Fortunately we have some techniques to help us in this process.

Learning to read and write algorithms written in a programming language like Python or Java is a lot like learning to read and write works written in human languages like English or Arabic. When you are learning a new language (perhaps you are learning Japanese or maybe Swahili) it is usually easier to compose your thoughts in the language that you are more familiar with; and then to translate these into the language that you are less familiar with. The same is true when learning a programming language like Python or Java.

Instead of trying to write an algorithm in a language that we are less familiar with (like Python or Java) we could write the algorithm in a language that we are more familiar with - like English. We are less likely to make mistakes in a language we are more familiar with. Also, unlike programming languages, which have very strict rules for grammar and spelling and syntax, human languages like English are much more forgiving. We can make spelling and grammar mistakes and would often still be able to read and understand the intended message. For example, consider the following English sentence:

go too the grocery store and by some potato

There are several errors in this sentence. The first letter is not capitalized. The words "too" and "by" should be "to" and "buy". The word "potato" should probably be plural, or the word "some" should be "a". There is no period at the end. However, we can still read it and understand it.

So, we can use English as the base language for describing our algorithms. We can even be flexible in terms of the actual grammatical rules of the language. Then, when we have worked out our algorithm, we can start to translate it into our programming language.

This idea of using a simplified version of English (or any other human language) to work out and express your algorithm is called **pseudocode** (pronounced soo-doe-code). The foremost rules of pseudocode are to make it as easy as possible to read and to understand, and to keep it simple. Pseudocode is not quite English and not quite a programming language. It is somewhere between a programming language and human language. The purpose of pseudocode is to specify an algorithm in enough detail that you could give it to any programmer using any programming

algorithm in enough detail that you could give it to any programmer, using any programming language, and they would be able to implement that algorithm in the programming language.

Pseudocode
An informal, high-level; description of an algorithm, intended for maximum human readability.

With pseudocode, there are no strict rules for syntax, and it is meant for simplicity and optimal readability by humans, not a computer.

Consider an example we have previously seen.

```
Repeat Until all cards are sorted in correct order
  Pick two cards at random

  If card on the left is greater than card on the right then
    Swap the positions of these two cards

End
```

This is a pretty good description of an algorithm for sorting a set of cards. It is easy to read and to understand, and it is pretty simple. No computer will be able to follow these instructions, because these instructions are not written in any programming language. However, a good programmer with knowledge of a programming language, like Python or Java, would be able to translate this into that programming language without much trouble.

The harder part of computer science and programming is designing and expressing the algorithm. The easier part is translating that algorithm into a programming language. The technique of using pseudocode allows us to separate these two things - designing and expressing the algorithm, and then translating (or implementing) that algorithm in a programming language - and therefore focus on one problem at a time. This is an example of what I believe is the most powerful ability in the computer scientists skill set - the ability to break complicated things down into two or more simpler things and then to better use our limited resources to solve each of these smaller simpler problems one at a time - in other words, to divide and conquer.

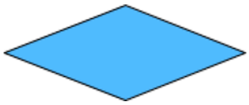
Flowcharts

Another useful tool for understanding algorithms is a flowchart. A flowchart makes communicating and documenting a process (an algorithm) quick and clear, so that the process will more likely be understood correctly. There are also benefits to the process of creating a flowchart itself. As you build a flowchart step by step, you'll be able to focus on the detail of each individual step and part

build a flowchart step by step, you'll be able to focus on the detail of each individual step and part of the process, without being overwhelmed by the complete algorithm - you can better zoom in to see the pieces, and then zoom out again to see the bigger picture.

Flowcharts are graphical or pictorial representations of algorithms. We use a set of geometric shapes and arrows as the visual elements. Here is a list of those visual elements and a description of their meaning.

Figure 1.4.4: Flowchart Symbols

Symbol	Meaning
	An oval (or circle or rounded rectangle) represents the start point or the end point of an algorithm
	A rectangle represents a step in the process or algorithm
	A parallelogram represents an input or an output
	A diamond represents a decision
	A line with an arrow is a connector that represents the flow through the steps of the process or algorithm.

To see just how effective a flowchart can be, consider Euclid's algorithm for computing the greatest common divisor of two numbers (the greatest common divisor is the largest whole number that will divide both of the two numbers evenly).

Here is a representation of Euclid's algorithm in pseudocode.

Input a

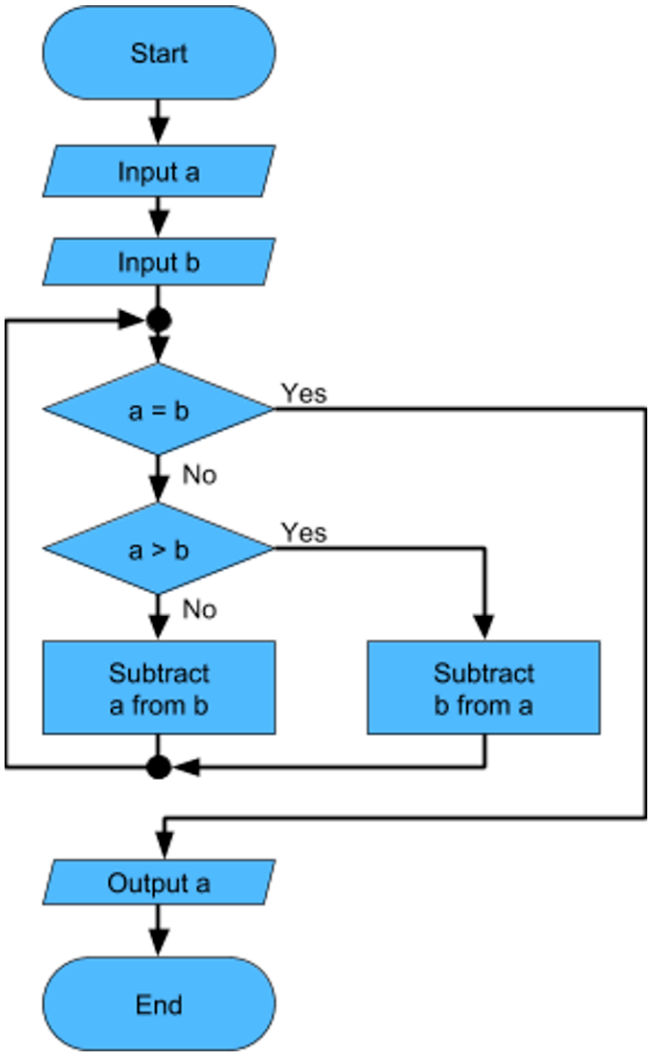
Input b

Repeat Until a is equal to b
  If a is greater than b then
    subtract b from a
  Otherwise
    subtract a from b

Output a

Here is the same algorithm represented as a flowchart.

Figure 1.4.5: Flowchart for Euclid's Algorithm



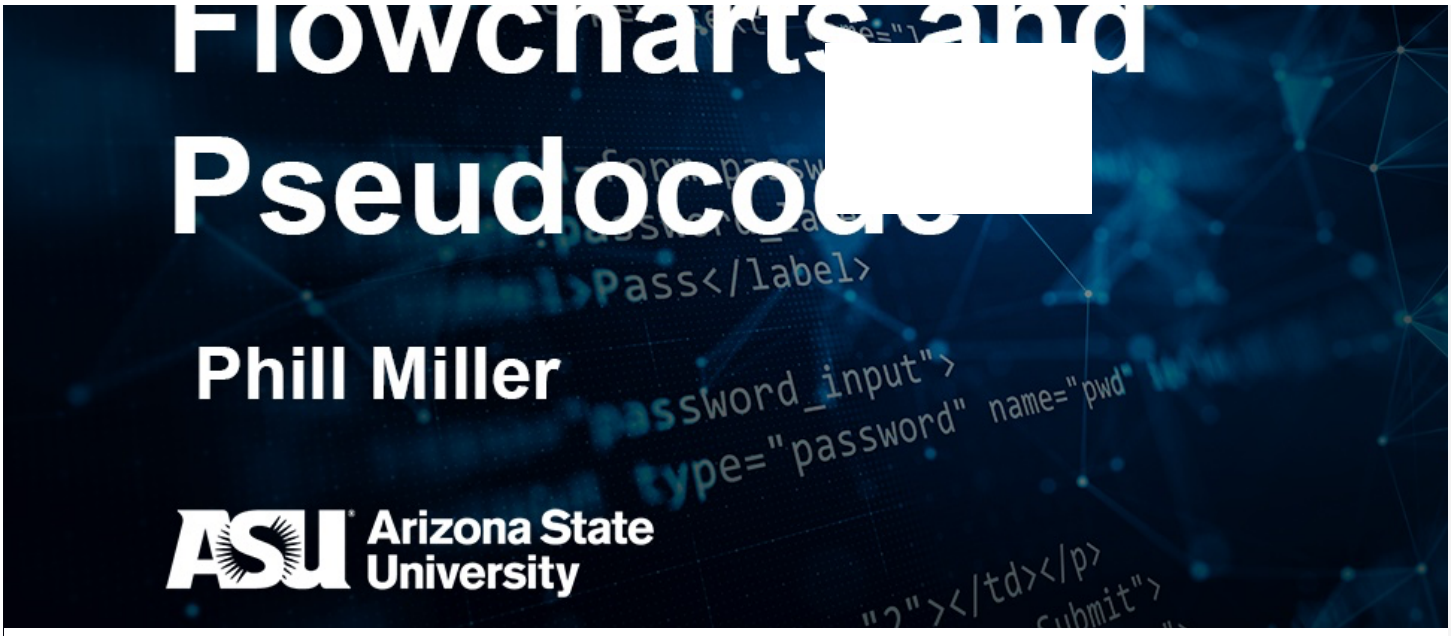
While it may seem as if the pseudocode representation is simpler and easier to read and understand, this is likely an illusion due to the fact that the textual pseudocode representation is more compact.

Notice that other than the **Start** and **End** elements of the flowchart, each line of text in the pseudocode algorithm is represented by one geometric symbol in the flowchart. Additionally the flowchart has the advantage of directly representing the flow (the sequence of steps) through the algorithm because the arrows show explicitly which instruction(s) follow other instructions. These features of flowcharts make it much easier for humans to follow the algorithm - it is much easier to keep track of where you are in the algorithm and what step comes next. This makes flowcharts an ideal tool for algorithm design and analysis - for reasoning about algorithms.

In contrast, pseudocode has the advantage of being more like code in a programming language. It is easier to take an algorithm that is represented in pseudocode and translate that algorithm into some programming language. Flowcharts and pseudocode are two powerful tools that computer scientists and programmers use to help them solve problems. Each of these tools has strengths and weaknesses. Each of these tools meets some of our needs when we are solving problems. Flowcharts are great when we are designing or trying to understand some algorithm because their visual nature makes them easier to reason about. Pseudocode is great for sketching out our algorithms in an informal way in order to make it much easier for us to then implement that algorithm in some programming language.

Flowchart
A graphical representation of an algorithm
that uses few words and simple symbols to make a process easier to understand.

Figure 1.4.6: Flowcharts and Pseudocode (11:08)



1.5 Programming (general)

Computer program basics

Computer programs are abundant in many people's lives today, carrying out applications on smartphones, tablets, and laptops, powering businesses like Amazon and Netflix, helping cars drive and planes fly, and much more.

A computer ***program*** consists of instructions executing one at a time. Basic instruction types are:

- ***Input***: A program gets data, perhaps from a file, keyboard, touchscreen, network, etc.
- ***Process***: A program performs computations on that data, such as adding two values like x + y.
- ***Output***: A program puts that data somewhere, such as to a file, screen, network, etc.

Programs use ***variables*** to refer to data, like x, y, and z below. The name is due to a variable's value varying as a program assigns a variable like x with new values.

PARTICIPATION ACTIVITY 1.5.1: A basic computer program.

Animation captions:

1. A basic computer program's instructions get input, process, and put output. This program first assigns x with what is typed on the keyboard input, in this case 2.
2. The program's next instruction gets the next input, in this case 5.
3. The program then does some processing, in this case assigning z with x + y (so 2 + 5 yields z of 7).
4. Finally, the program puts z (7) to output, in this case to a screen.

PARTICIPATION ACTIVITY 1.5.2: A basic computer program.

Consider the example above.

1) The program has a total number of _____ instructions.

Check Show answer

2) Suppose a new instruction was inserted as follows:

```
...
z = x + y
Add 1 more to z (new
instruction)
Put z to output
```

What would the last instruction then output to the screen?

Check Show answer

3) Consider the instruction: z = x +

y. If x is 10 and y is 20, then z is assigned with _____.

Check Show answer

A program is like a recipe

Some people think of a program as being like a cooking recipe. A recipe consists of *instructions* that a chef executes, like adding eggs or stirring ingredients. Likewise, a computer program consists of instructions that a computer executes, like multiplying numbers or outputting a number to a screen.

Bake chocolate chip cookies:

- Mix 1 stick of butter and 1 cup of sugar.
- Add egg and mix until combined.
- Stir in flour and chocolate.
- Bake at 350F for 8 minutes.

A first programming activity

Below is a simple tool that allows a user to rearrange some prewritten instructions (in no particular programming language). The tool illustrates how a computer executes each instruction one at a time, assigning variable m with new values throughout and outputting ("printing") values to the screen.

PARTICIPATION ACTIVITY 1.5.3: A first programming activity.

Execute the program by clicking the "Run program" button and observe the output. Click and drag the instructions to change the order of the instructions, and execute the program again. Not required (points are awarded just for interacting), but can you make the program output a value greater than 500? How about greater than 1,000?

program output a value greater than 500? How about greater than 1,000?

Run program

```
m = 5

put m

m = m * 2
put m

m = m * m
put m

m = m + 15
put m
```

m:

PARTICIPATION ACTIVITY 1.5.4: Instructions.

1) Which instruction completes the program to compute a triangle's area?
base = Get next input
height = Get next input
Assign x with base * height

____

Put x to output

- Multiply x by 2
- Add 2 to x
- Multiply x by 1/2

2) Which instruction completes the program to compute the average of three numbers?
x = Get next input
y = Get next input
z = Get next input

____

Put a to output

Put a to output

- a = (x + y + z) / 3
- a = (x + y + z) / 2
- a = x + y + z

Computational thinking

Mathematical thinking became increasingly important throughout the industrial age to enable people to successfully live and work. In the information age, many people believe ***computational thinking***, or creating a sequence of instructions to solve a problem, will become increasingly important for work and everyday life. A sequence of instructions that solves a problem is called an ***algorithm***.

PARTICIPATION ACTIVITY | 1.5.5: Computational thinking: Creating algorithms to draw shapes using turtle graphics.

A common way to become familiar with algorithms is called turtle graphics: You instruct a robotic turtle to walk a certain path, via instructions like "Turn left", "Walk forward 10 steps", or "Pen down" (to draw a line while walking).

The 6-instruction algorithm shown below ("Pen down", "Forward 100", etc.) draws a triangle.

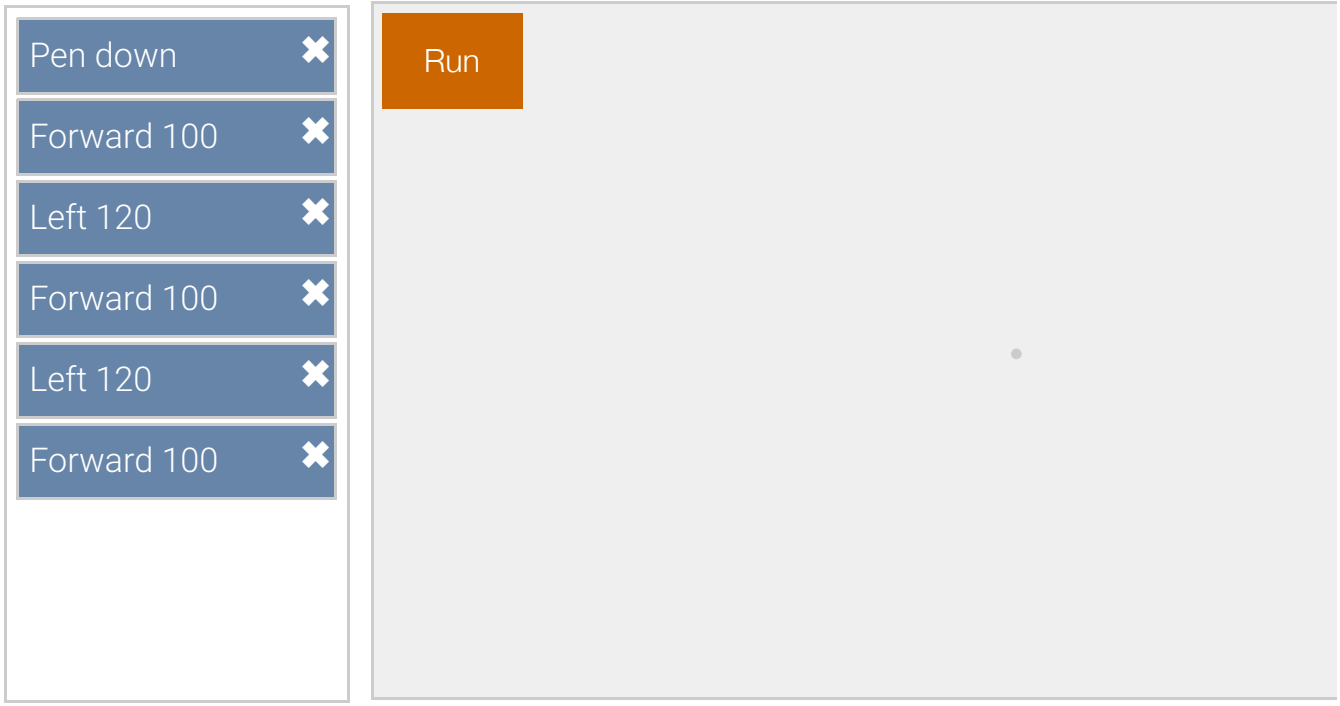
1. Press "Run" to see the instructions execute from top to bottom, yielding a triangle.
2. Can you modify the instructions to draw a square? Hint: "Pen down", "Forward 100", "Left 90", "Forward 100", "Left 90"—keep going!
3. Experiment to see what else you can draw.

Note: The values after a Left or RIght turn are angles in degrees.

How to:

- Add an instruction: Click an orange button ("Pen up", "Pen down", "Forward", "Turn left").
- Delete an instruction: Click its "x".
- Move an instruction: Drag it up or down.

Pen up | Pen down | Forward | Turn left | Clear



1.6 Programming basics

A first program

A simple Java program appears below.

- A ***program*** starts in main(), executing the statements within main's braces { }, one at a time.
- Each statement typically appears alone on a line and ends with a ***semicolon***, as English sentences end with a period.
- The `int wage` statement creates an integer variable named wage. The `wage = 20` statement assigns wage with 20.
- The print and println statements output various values.

PARTICIPATION ACTIVITY | 1.6.1: Program execution begins with main, then proceeds one statement at a time.

Animation content:

```
undefined
```

Animation captions:

1. A program begins executing statements in main(). 'int wage' declares an integer variable. 'wage = 20' assigns wage with 20.
2. The System.out.print statement outputs 'Salary is ' to the screen at the cursor's present location.
3. This System.out.println statement outputs the result of wage * 40 * 52, so 20 * 40 * 52 or 41600, and then moves cursor to next line.

PARTICIPATION ACTIVITY 1.6.2: A first program.

Consider the program above.

1) Program execution begins at main() and executes statements surrounded by which symbols?
   - ( )
   - { }
   - " "

2) The statement `int wage;` creates a variable named wage that is used to _____ the value 20.
   - input
   - output
   - hold

3) Would the following order of statements work the same as above?

```
wage = 20;
int wage;
```

   - No
   - Yes

4) Each statement ends with what

symbol?

- Semicolon ;
- Period .
- Colon :

5) The expression wage * 40 * 52 resulted in what value?

- 20
- 41600
- 20 * 40 * 52,

6) Each System.out.print() and System.out.println() statement outputs items to _____.

- a file named output.txt
- the keyboard
- the screen

zyDE 1.6.1: A first program.

Below is the zyBooks Development Environment (zyDE), a web-based programming environment. Click run to compile and execute the program, then observe the output. 20 to a different number like 35 and click run again to see the different output.

Load default template...

Run

```
1  public class Salary {
2     public static void main (String
3        int wage;
4
5        wage = 20;
6
7        System.out.print("Salary is "
8        System.out.println(wage * 40
9     }
10 }
11
```

Basic input

Programs commonly get input values, perform some processing on that input, and put output values to a screen or elsewhere. Input is commonly gotten from a keyboard, a file, fields on a web form or app, etc.

The following code at the top of a file enables the program to get input:

```
import java.util.Scanner;
```

A ***Scanner*** is a text parser that can get numbers, words, or phrases from an input source such as the keyboard. Getting input is achieved by first creating a Scanner object via the statement: `Scanner scnr = new Scanner(System.in);`. System.in corresponds to keyboard input. Then, given Scanner object scnr, the following statement gets an input value and assigns x with that value: x = ***scnr.nextInt()*** ;

PARTICIPATION ACTIVITY 1.6.3: A program can get an input value from the keyboard.

Animation content:

undefined

Animation captions:

1. A scanner object is setup to scan input from System.in. The scnr.nextInt() statement gets an input value from the keyboard (or file, etc.) and puts that value into the wage variable.
2. wage's value can then be used in subsequent processing and outputs.

PARTICIPATION ACTIVITY 1.6.4: Basic input.

1) Assuming scnr already exists, which statement gets an input number into variable numCars?

- `scnr.nextInt(numCars);`
- `numCars = scnr.nextInt;`
- `numCars = scnr.nextInt();`

PARTICIPATION ACTIVITY 1.6.5: Basic input.

1) Type a statement that gets an input value into variable numUsers. Assume scnr already exists and numUsers has been declared.

Check Show answer

zyDE 1.6.2: Basic input.

Run the program and observe the output. Change the input box value from 3 to anoth number, and then run again. Note: Handling program input in a web-based developm environment is surprisingly difficult. *Preentering* the input is a workaround in zyDE. Fo dynamic output and input interaction, use a traditional development environment.

Load default template...

```
1  import java.util.Scanner;
2
3  public class DogYears {
4     public static void main(String [
5        Scanner scnr = new Scanner(Sy
6        int dogYears;
7        int humanYears;
8
9        dogYears = scnr.nextInt();
10       humanYears = 7 * dogYears;
11
```

3

Run

```
12      System.out.print("A ");
13      System.out.print(dogYears);
14      System.out.print(" year old d
15      System.out.print(humanYears);
```

Basic output: Text

The ***System.out.print*** construct supports output. Outputting text is achieved via: `System.out.print("desired text");`. Text in double quotes " " is known as a ***string literal***. Multiple output statements continue printing on the same output line.

System.out.println (note the ln at the end, short for "line"), starts a new output line after the outputted values, called a ***newline***. A common error is to type the number "1" or a capital `I`, as in "in", instead of a lower case `l` as in "print line".

Figure 1.6.1: Outputting text and new lines.

```
public class KeepCalm {
   public static void main (String []
args) {

      System.out.print("Keep calm");
      System.out.print("and");
      System.out.print("carry on");
   }
}
```

```
Keep calmandcarry
on
```

```
public class KeepCalm {
   public static void main (String []
args) {

      System.out.println("Keep calm");
      System.out.println("and");
      System.out.println("carry on");
   }
}
```

```
Keep calm
and
carry on
```

Outputting a blank line is achieved by: System.out.println().

PARTICIPATION ACTIVITY 1.6.6: Basic text output.

1) Which statement outputs: Welcome!

- `System.out.print(Welcome!);`
- `System.out.print "Welcome!";`
- `System.out.print("Welcome!");`

2) Which statement outputs Hey followed by a new line?

- `System.out.print("Hey"ln);`
- `System.out.println(Hey);`
- `System.out.println("Hey");`

PARTICIPATION ACTIVITY 1.6.7: Basic text output.

End each statement with a semicolon. Do not output a new line unless instructed.

1) Type a statement that outputs: Hello

Check Show answer

2) Type a statement that outputs Hello and then starts a new output line.

Check Show answer

Outputting a variable's value

Outputting a variable's value is achieved via: System.out.print(x); Note that no quotes surround x. println() could also be used.

Figure 1.6.2: Outputting a variable's value.

```
public class Salary {
  public static void main (String []
args) {
      int wage;

      wage = 20;

      System.out.print("Wage is: ");
      System.out.println(wage);
      System.out.println("Goodbye.");
  }
}
```

```
Wage is:
20
Goodbye.
```

Note that the programmer intentionally did *not* start a new output line after outputting "Wage is: " so that the wage variable's value would appear on that same line.

PARTICIPATION ACTIVITY 1.6.8: Basic variable output.

1) Given variable numCars = 9, which statement outputs 9?

- `System.out.print("numCars");`
- `System.out.print numCars;`
- `System.out.print(numCars);`

PARTICIPATION ACTIVITY 1.6.9: Basic variable output.

1) Type a statement that outputs the value of numUsers (a variable). End statement with a semicolon.

Check Show answer

Outputting multiple items with one statement

Programmers commonly use a single output statement for each line of output by combining the outputting of text, variable values, and a new line. The programmer simply separates the items with a + symbol. Such combining can improve program readability because the program's code corresponds more closely to the program's output.

Figure 1.6.3: Outputting multiple items using one output statement.

```
public class Salary {
   public static void main (String [] args)
{
      int wage;

      wage = 20;

      System.out.println("Wage is: " +
wage);
      System.out.println("Goodbye.");
   }
}
```

```
Wage is:
20
Goodbye.
```

zyDE 1.6.3: Single output statement.

Modify the program to use only two output statements, one for each output sentence

```
In 2014, the driving age is 18.
10 states have exceptions.
```

Do not type numbers directly in the output statements; use the variables. ADVICE: Ma
incremental changes—Change one code line, run and check, change another code lin
check, repeat. Don't try to change everything at once.

Load default template...

Run

```
1 public class DrivingAge {
2    public static void main (String
3       int drivingYear;
4       int drivingAge;
5       int numStates;
```

```
6
7       drivingYear = 2014;
8       drivingAge = 18;
9       numStates = 10;
10
11      System.out.print("In ");
12      System.out.print(drivingYear)
13      System.out.print(", the drivi
14      System.out.print(drivingAge);
15      System.out.println(".");
```

PARTICIPATION ACTIVITY 1.6.10: Basic output.

Indicate the actual output of each statement. Assume userAge is 22.

1)
```
System.out.print("You're " +
userAge + " years.");
```

- You're 22 years.
- You're userAge years.
- No output; an error exists.

2)
```
System.out.print(userAge + "years
is good.");
```

- 22 years is good.
- 22years is good.
- No output; an error exists.

PARTICIPATION ACTIVITY 1.6.11: Output simulator.

The following variable has already been declared and assigned: `countryPopulation = 1344130000;`. Using that variable (do not type the large number) along with text, finish the output statement to output the following:

```
China's population was 1344130000 in 2011.
```

Then, try some variations, like:

```
1344130000 is the population. 1344130000 is a lot.
```

```
System.out.print( "Change this string!"
);
```

```
Change this string!
```

CHALLENGE ACTIVITY 1.6.1: Enter the output.

426488.2706032.qx3zqy7

Start

Type the program's output

```
public class GeneralOutput {
   public static void main (String [] args) {
      System.out.print("Joe is good.");
   }
}
```

```
Joe is good
```

1 2 3 4

Check Next

CHALLENGE ACTIVITY 1.6.2: Output basics.

For activities with output like below, your output's whitespace (newlines or spaces) must match exactly. See this note.

426488.2706032.qx3zqy7

Start

Write code that outputs the following. End with a newline. Remember to use println instead of print to output a newline.

```
This meal was good.
```

```
1 public class OutputTest {
2    public static void main (String [] args) {
3
4       /* Your code goes here */
5
6    }
7 }
```

1 2 3 4 5 6

Check Next Show solution

CHALLENGE ACTIVITY 1.6.3: Read multiple user inputs.

Write two ***scnr.nextInt*** statements to get input values into birthMonth and birthYear. Then write a statement to output the month, a slash, and the year. End with newline.

The program will be tested with inputs 1 2000 and then with inputs 5 1950. Ex: If the input is 1 2000, the output is:
`1/2000`

Note: The input values come from user input, so be sure to use scnr.nextInt statements, as in `birthMonth = scnr.nextInt();`, to get those input values (and don't assign values directly, as in `birthMonth = 1`).

426488.2706032.qx3zqy7

```
1 import java.util.Scanner;
2
```

```
3  public class InputExample {
4     public static void main(String [] args) {
5        Scanner scnr = new Scanner(System.in);
6        int birthMonth;
7        int birthYear;
8
9        /* Your solution goes here  */
10
11    }
12 }
```

Run

View your last submission

*A new output line can also be produced by inserting \n, known as a **newline character**, within a string literal. Ex: Outputting "1\n2\n3" outputs each number on its own output line. \n consists of two characters, \ and n, but together are considered as one newline character. Good practice is to use println to output a newline when possible, as println has some technical advantages not mentioned here.*

1.7 What is a Programming Language?

There are many natural languages - thousands. Ethnologue's Language of the World lists over 7,000 natural languages.

Natural languages, like Mandarin Chinese, English, Spanish, Hindi, Arabic, French, Portuguese, Russian, German, Japanese, and Korean - are evolved languages that developed over time through use, repetition, and geographic and cultural change. They have evolved to support the many communication needs of writers and speakers in a given culture.

Some of these languages are fairly similar to one another, like Hindi and Urdu, or Danish and

Some of these languages are fairly similar to one another, like Hindi and Urdu, or Danish and Norwegian, or Italian and Spanish. Others are very different from one another, like Japanese and English, or Arabic and Russian, or Hindi and German.

There are also many programming languages—hundreds. Wikipedia's List of programming languages page includes more than 500 programming languages.

Programming languages are constructed languages. They are designed and created for specific purposes. Unlike natural languages (which are generally large, complex, and suited for a wide range of communication needs) programming languages are generally small, simple, and suited for only one purpose - that is to communicate an unambiguous list of instructions for a machine to follow to accomplish some task.

Some programming languages are very similar to one another, like C++ (pronounced see-plus-plus), and Java, and C# (pronounced see-sharp). Others are very different from one another, like Scheme, Prolog, and Python.

Here is a list of a just few of the many programming languages:

Table 1.7.1: Programming Languages

ActionScript	Eiffel	OCaml
Ada	Erlang	Pascal
ALGOL	F#	Perl
Alice	FLOW-MATIC	PHP
Amiga E	Forth	PL/I
APL	Fortran	Plankalkül
AppleScript	Go	Prolog
Assembly language	Groovy	Python
AWK	Haskell	R
B	Icon	Racket

Babbage	Java	Ruby
BASIC	JavaScript	Rust
BCPL	Julia	Scala
Blockly	Legoscript	Scheme
Boo	Lisp	Scratch
B	Logo	Simula
C	Lua	Smalltalk
C++	Maple	SNOBOL
C#	Mathematica	SQL
Caml	MATLAB	Squeak
Clojure	Miranda	Swift
COBOL	Modula	Typescript
CoffeeScript	NetLogo	Verilog
D	Nim	VHDL
Dart	Oberon	Visual Basic
Delphi	Objective-C	Wolfram Language

Two important concepts in the study of languages are the concepts of **syntax** and **semantics**.

Syntax is the set of rules that define the structure of how sentences or statements are formed. In a natural language like English, syntax defines the ways that nouns and verbs and other parts of the language can be combined in a sequence to create a grammatically correct sentence.

For example, "Through raced green the car red the light." is not a syntactically correct sentence in English. It is still interesting to notice that you are probably able to infer the meaning of this text, even though it is not grammatically correct. On the other hand, "The green car raced through the red light." is a syntactically correct sentence in English.

Semantics is the meaning of a sentence or statement. For example the meaning of the English sentence "My dad woke up." is something to the effect that a biological process characterized by unconsciousness ceased in a person, who is the writer's male parent.

Natural languages are complex and open to interpretation, with loose rules for grammar and syntax. Words and phrases can have multiple meanings.

Here are a couple of examples of natural phrases with multiple meanings:

- Time flies like an arrow; fruit flies like a banana.
- He fed her cat food.
- One morning I shot an elephant in my pajamas

Programming languages are simple and unambiguous:

- With strict rules for grammar and syntax
- Instructions (statements) can have only one meaning

Natural languages are context sensitive. The same thing said or written in one context or setting, can have a different meaning in another context.

Programming languages are context free:

- The same text always has the same meaning, regardless of context.
- The meaning of any statement does not depend on the context or surrounding statements.
- A line of code always has a single meaning.

Human brains are very different from computers, and can easily infer meaning from the context, and from what was previously or subsequently said or written.

Computers, which are machines, must be able to unambiguously interpret every instruction in a program.

The fact that computers can only interpret programs written in simple languages and cannot infer meaning based on context or nuance, is actually the source of their power and usefulness.

Computers are able to repeatedly execute simple instructions extremely fast and without error because there is no ambiguity. It is necessary that a programming language be simple, unambiguous, and free from the nuances of context so that we can specify the instructions for the computer with the precision required for a machine (that can only do exactly what it is told) to perform.

What is Java

Java is a general purpose programming language that is designed to support writing programs that are platform independent (Can be executed on Windows, Mac, Mobile, etc.).

Java is Class-based and Object Oriented, which means that it is designed for Object Oriented Programming. We will learn more about what this means later in the semester.

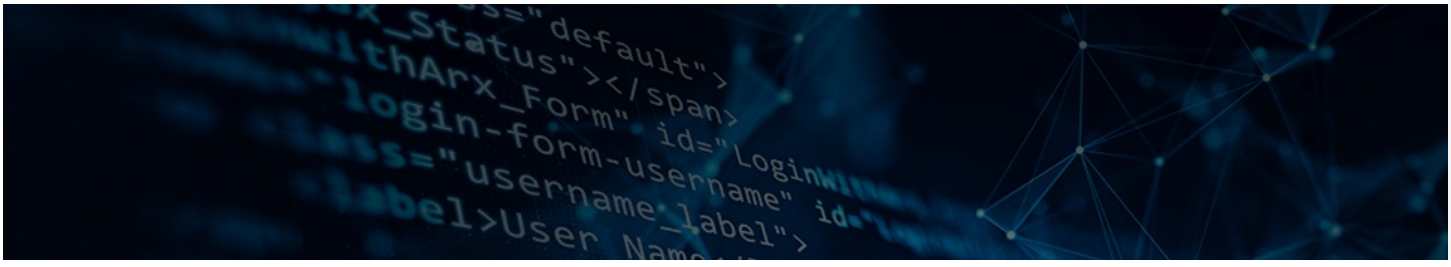
Java is strongly typed - meaning that we must declare the types of variables that we use, and they must remain the same type throughout execution of a program.

Java is a compiled language, but Java bytecode runs on the Java Virtual Machine (JVM). This means that the Java source code that we write is translated into bytecode by the Java compiler. This bytecode is not executed directly by the computer, but by an interpreter program called the Java Virtual Machine. This means that bytecode is portable across operating systems and computer types, as long as someone has written a version of the Java Virtual machine for that platform. The JVM then only needs to be written once for a platform, and most programs written in Java can be executed on any machine without modifying the source code.

How is Java Used?

Java is used in a huge variety of industries, ranging from enterprise software to medical applications and devices, server applications, games, mobile devices and more! The Android Operating System is heavily based on Java related technologies.

Figure 1.7.1: How do Programming Languages Work? (5:46)

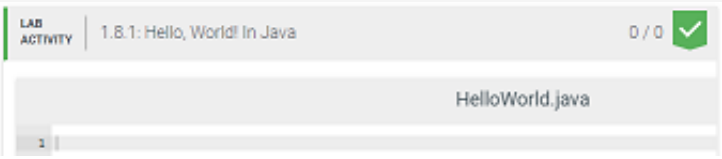


1.8 Hello, World! In Java

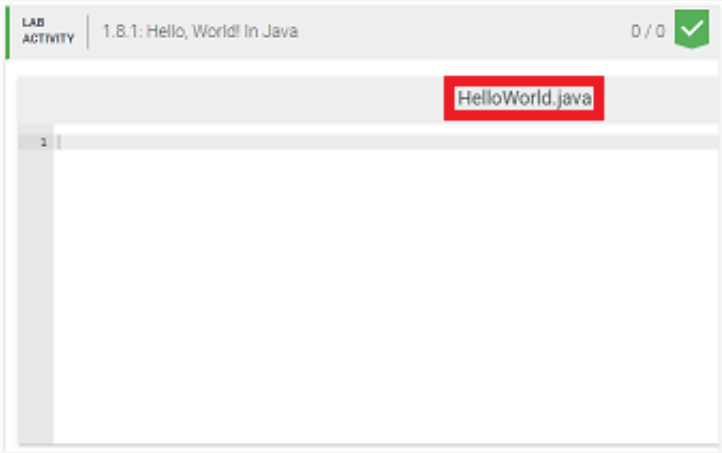
In this section we will write, compile, run, test, edit, and submit our first complete Java program. This will be a very simple Java program that does nothing more than display a message in the **terminal** when we run it. **You may wish to read all the way through these instructions once before you begin to follow the instructions and write the code.**

Write

We will write all of our code in the **LAB ACTIVITY** code editor window below. The **LAB ACTIVITY** code editor window looks like this:



We will start by writing a classic "Hello, World!" program. First, notice that in the LAB ACTIVITY code editor window below, we will be writing our Java code in a file named **`HelloWorld.java`**. You can see the file name at the top of the code editor window (highlighted in red in the image below).



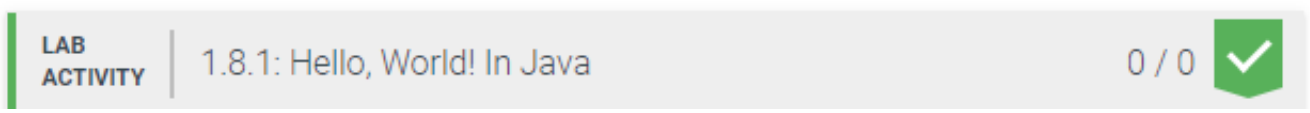
A Java program is composed of one or more Java class declarations (you will learn more about Java classes and Java class declarations later in this course). In general, each Java class declaration is contained in one **`.java`** file. It is important to remember that Java requires that the name of your class must exactly match (including spelling and capitalization) the name of the file that the class is in. For example a Java class named **`HelloWorld`** should be in a file named **`HelloWorld.java`**.

So to begin our first Java program, we will start by writing a Java class declaration for a class named **`HelloWorld`**. Here is the code you should write first:

```
public class HelloWorld {

}
```

When you have written your **`HelloWorld`** class declaration, your code editor should look like this:



HelloWorld.java

```
1 public class HelloWorld {
2
3 }
4
```

Do not worry too much about all the details right now. Things like **`public`** and the curly braces **`{ }`** will be explained as we move through the course. For now, you just need to know that all of your Java programs will look like this, with a **`.java`** file that contains a Java class declaration that declares a class with the same name as the file. In this example we have a class named **`HelloWorld`** in a file named **`HelloWorld.java`**.

Next we must write a method. A method is just a named sequence of instructions that we want the computer to execute. Again, we will learn much more about methods later in this course. All methods must be inside the Java class curly braces. We will be writing a special method named **`main`**. Write the following **`main`** method inside of the **`HelloWorld`** class curly braces:

```
public static void main(String[] args) {


}
```

When you have written your **`main`** method, your code editor should look like this:

LAB ACTIVITY | 1.8.1: Hello, World! In Java | 0 / 0

HelloWorld.java

```
1 public class HelloWorld {
2    public static void main(String[] args) {
3
4    }
5 }
6
```

Note that we indent the **`main`** method. This is to make it easier to see that the **`main`** method is inside the **`HelloWorld`** class.

The **`main`** method is special in a Java program, because this is where the computer will look to find the sequence of instructions that you want it to execute. Again, do not worry too much about the

details like `public static void` or `(String[] args)` - these will be explained in time. For now, you just need to know that your Java programs will consist of a `.java` file that contains a Java class with the same name, and in this class will be a `main` method.

In the `main` method, we can write the instructions that we want the computer to execute when we run the program. For this very simple Java program, we just want to display a message in the terminal. One easy way to do this in Java is with a `System.out.println` statement. Write the following statement inside the curly braces for your `main` method:

```
System.out.println("Hello, World!");
```

When you have written your `System.out.println` statement, your code editor should look like this:

Note that we indent the statements. This is to make it easier to see that the statements are inside the method.

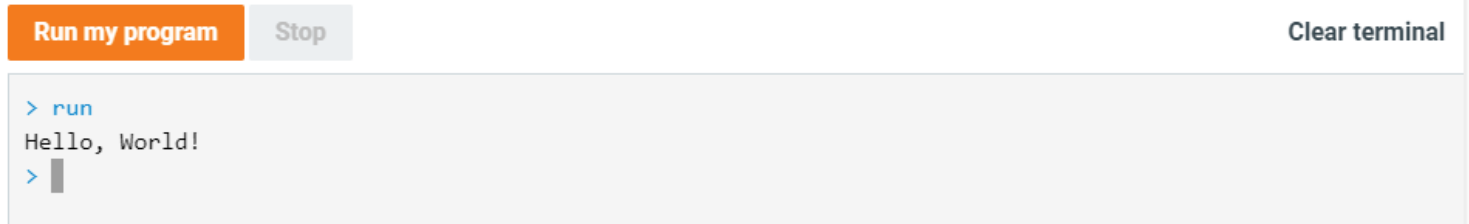
Compile, Run, and Test

Now that we have written our first Java program, we should attempt to run it. We can attempt to run our Java program by clicking on the **Run my program** button (Run my program) below the code editor window.

Before the computer can run our program, the Java compiler must translate our Java code into instructions that the computer can execute (machine instructions). Every time we run our Java program, the Java compiler will first compile (translate into machine instructions) our code. If the Java compiler succeeds, then our program will run. If the Java compiler does not succeed, the compiler will output some compiler error messages to help us find and fix the errors in our code.

If your code looks like the example above, then your code should compile and run without errors. Go ahead and click the **Run my program** button.

If there are no errors in your code, you should see the following output in the **terminal** window:



If there are errors, then make sure that your code looks just like this (below), and then run it again.

Edit

Now, let's explore a bit. We will make a series of changes to the code, and each time we will compile and run the code again to see what affect the changes have.

Let's start by changing what the program prints out to the **terminal**. Instead of printing out `Hello, World!`, let's make it print out `Hello, Java!`. Edit the code to print out `Hello, Java!` instead of `Hello, World!`, and then click the **Run my program** button. Be sure that you are only changing the text inside the double quote marks.

Notice that, with a `System.out.println` statement, the computer will print to the **terminal** whatever you have placed between the double quote marks.

So, now your code should look like this:

And your **terminal** output should look like this:

```
> run
Hello, Java!
>
```

Errors and Compiler Error Messages

Now, let's introduce an error in the code just to see what happens. One common error is to forget to place a semicolon (`;`) at the end of a statement. So, delete the semicolon at the end of the `System.out.println` statement. Then attempt to run the program again.

When you click the **Run my program** button, the Java compiler will attempt to compile your Java code, but it will fail because there is an error in your code. Instead of your program running and printing out the message you expected, the compiler will print out a compiler error message that looks like this:

```
> run
HelloWorld.java:3: error: ';' expected
      System.out.println("Hello, Java!")
                                        ^
1 error
>
```

These compiler error messages can be very helpful. They give you a lot of information about what might be wrong with your code and where. For example, let's look at the parts of the compiler error message above.

It starts with `HelloWorld.Java:3`. This is telling us that there is an error in the file named `HelloWorld.java`, and that error is on or just before line #`3`. You will notice that line #`3` in your program is the `System.out.println` statement. So, this is where we should start looking for the error.

The next part of the compiler error message is `Error: ';' expected`. This is telling us that the compiler expected to find a semicolon (`;`), but did not find one.

The next part of the compiler error message is:

```
      System.out.println("Hello, Java!")
```

```
                                                                    ^
```

This is a copy of the line that the compiler believes that the error is on, and the caret (^) is pointing to the place on that line where the compiler believes the semicolon should go. In other words, this compiler error message says that the compiler believes that there should probably be a semicolon at the end of line #3 in your code - and this is exactly correct. Compiler error messages will not always be this helpful, but they will always be helpful if you read them and try to understand what they are telling you. Learning to read and understand compiler error messages takes time and practice, so never miss an opportunity to do so - always read all compiler error messages.

Correct this error by adding the semicolon back to the end of the `System.out.println` statement on line #3 in your code. Then run the program again to confirm that it is working as expected now.

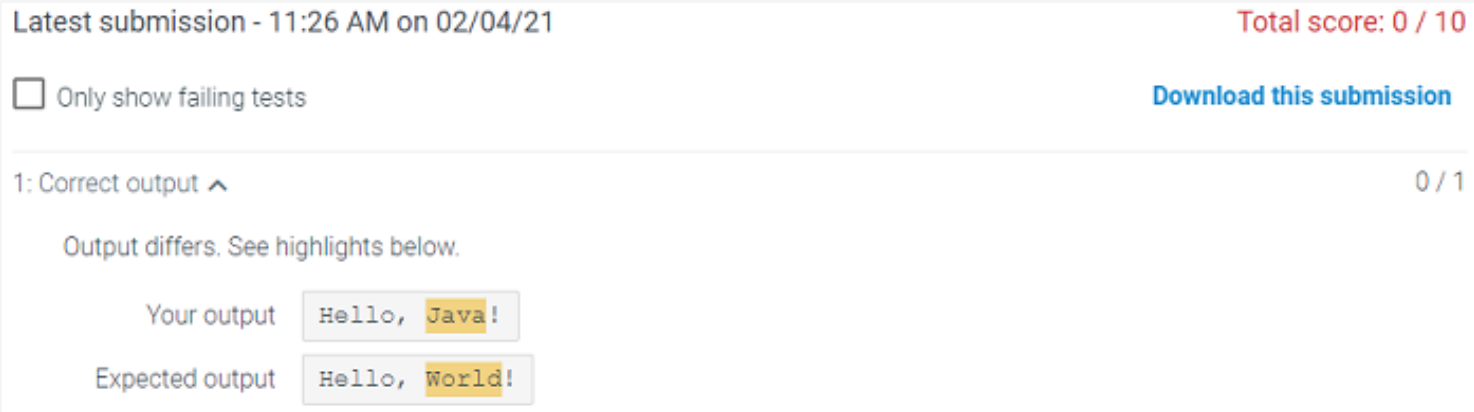
Submit

Now that we have corrected the compiler errors, and our program seems to be working correctly, we can submit our code for grading.

To submit our code for grading we must first click on the **Submit mode** button (Submit mode), below the **terminal** window. Then we click on the **Submit for grading** button (Submit for grading).

Do this now.

After a moment you should see the feedback from the auto-grader, that looks like this:

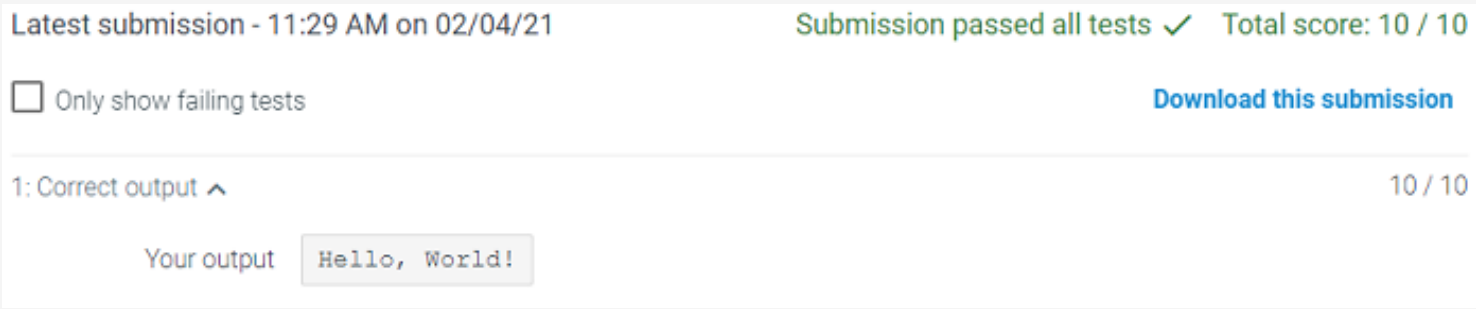


You can see that our code submission is currently failing the test (named **Correct output**), which is worth 10 points. Because of this we are currently getting 0 / 10 points for this assignment. If we look closely at the details provided in the failed test message, we can see that **Your output** was `Hello, Java!`, but the **Expected output** is `Hello, World!`. In this case the auto-grader has even tried to identify and highlight the specific parts of the outputs that do not match.

From this failed test message we can clearly see that we need to change our code so that it outputs `Hello, world!`.

Do this now, then switch back to **Develop mode**, (Develop mode) and run the program to confirm that it is producing the correct output - `Hello, world!`. Then switch to **Submit mode** again, and re-submit your code. For these coding assignments in this course, you may continue this process of writing, compiling, running, testing, correcting, submitting, and re-submitting as many times as you like until the due date has passed.

Once you have corrected your code to pass all the auto-grader tests, you should see auto-grader feedback that looks something like this:



You're Finished!

Congratulations! You have completed your first coding assignment.

426488.2706032.qx3zqy7

LAB ACTIVITY | 1.8.1: Hello, World! In Java | 10 / 10

HelloWorld.java

```
1 public class HelloWorld {
2     public static void main (String[] args) {
3         System.out.println("Hello, World!");
4     }
5 }
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Predefine program input (optional)

If you'd like to predefine your inputs, provide them here.

Run my program | Stop | Clear terminal

```
>
```

Coding trail of your work What is this?

```
8/22 M---10 min:3
```

1.9 Comments and whitespace

Comments

A ***comment*** is text a programmer adds to code, to be read by humans to better understand the code but ignored by the compiler. Two common kinds of comments exist:

- A ***single-line comment*** starts with // and includes all the following text on that line. Single-line comments commonly appear after a statement on the same line.
- A ***multi-line comment*** starts with /* and ends with */, where all text between /* and */ is part of the comment. A multi-line comment is also known as a ***block comment***.

Figure 1.9.1: Comments example.

```
import java.util.Scanner;

/*
   This program calculates the amount of pasta to cook, given the
   number of people eating.

   Author: Andrea Giada
   Date:   May 30, 2017
*/

public class PastaCalculator {
   public static void main (String [] args) {
      Scanner scnr = new Scanner(System.in);
      int numPeople;         // Number of people that will be eating
      int totalOuncesPasta;  // Total ounces of pasta to serve
numPeople

      // Get number of people
      System.out.println("Enter number of people: ");
      numPeople = scnr.nextInt();

      // Calculate and print total ounces of pasta
      totalOuncesPasta = numPeople * 3;  // Typical ounces per person
      System.out.println("Cook " + totalOuncesPasta + " ounces of
pasta.");
   }
}
```

PARTICIPATION ACTIVITY 1.9.1: Comments.

Indicate which are valid code.

1)
```
// Get user input
```

- Valid
- Invalid

2)
```
/* Get user input */
```

- Valid
- Invalid

3)
```
/* Determine width and height,
   calculate volume,
   and return volume squared.
*/
```

- Valid
- Invalid

4)
```
// Print "Hello" to the screen //
```

- Valid
- Invalid

5)
```
// Print "Hello"
   Then print "Goodbye"
   And finally return.
//
```

- Valid
- Invalid

6)
```
/*
 * Author: Michelangelo
 * Date: 2014
 * Address: 111 Main St, Pacific
Ocean
 */
```

- Valid
- Invalid

7)
```
// numKids = 2;  // Typical number
```

- Valid

Valid

Invalid

8)
```
/*
   numKids = 2;  // Typical number
   numCars = 5;
*/
```

Valid

Invalid

9)
```
/*
   numKids = 2;  /* Typical number
*/
   numCars = 5;
*/
```

Valid

Invalid

JavaDoc comments

Java supports a third type of comment, known as a JavaDoc comment (discussed elsewhere), which is a specially formatted multi-line comment that can be converted to program documentation in HTML.

Whitespace

Whitespace refers to blank spaces (space and tab characters) between items within a statement and blank lines between statements (called newlines). A compiler ignores most whitespace.

Good practice is to deliberately and consistently use whitespace to make a program more readable. Programmers usually follow conventions defined by their company, team, instructor, etc., such as:

- Use blank lines to separate conceptually distinct statements.
- Indent lines the same amount.
- Align items to reduce visual clutter.
- Use a single space before and after any operators like =, +, *, or / to make statements more readable.

Figure 1.9.2: Good use of whitespace.

```
import java.util.Scanner;

public class WhitespaceEx {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int myFirstVar;    // Aligned comments yield less
      int yetAnotherVar; // visual clutter
      int thirdVar;

      // Above blank line separates variable declarations from the rest
      System.out.print("Enter a number: ");
      myFirstVar = scnr.nextInt();

      // Above blank line separates user input statements from the rest
      yetAnotherVar = myFirstVar;        // Aligned = operators
      thirdVar      = yetAnotherVar + 1;
      // Also notice the single-space on left and right of + and =
      // (except when aligning the second = with the first =)

      System.out.println("Final value is " + thirdVar); // Single-space on each side
of +
   }
}
```

Figure 1.9.3: Bad use of whitespace.

```
import java.util.Scanner;
public class PastaCalculator {
public static void main (String [] args) {
Scanner scnr = new Scanner(System.in);int numPeople;int
totalOuncesPasta;
System.out.println("Enter number of people:");
numPeople = scnr.nextInt(); totalOuncesPasta = numPeople * 3;
System.out.println("Cook "+totalOuncesPasta+" ounces of pasta.");}}
```

PARTICIPATION ACTIVITY | 1.9.2: Whitespace.

Are the specified lines of code good or bad uses of whitespace?

```
import java.util.Scanner;
```

```
public class WhitespaceGoodAndBad {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int userAge;
      int currentDecade;
      int    nextDecade;
      int nextMilestone;

      System.out.println("Enter your age: ");
      userAge = scnr.nextInt();

      currentDecade=userAge/10;
      nextDecade = currentDecade + 1;
          nextMilestone = nextDecade * 10;
      System.out.println("Next big birthday is at " + nextMilestone);
   }
}
```

1) `int       nextDecade;`

- Good
- Bad

2) `currentDecade=userAge/10;`

- Good
- Bad

3) `nextDecade = currentDecade + 1;`

- Good
- Bad

4)

`nextMilestone = nextDecade * 10;`

- Good
- Bad

Compiling code with comments and whitespace

The animation below provides a (simplified) demonstration of how a compiler processes code from left-to-right and line by line, finding each statement (and generating machine code using 0s and 1s) and ignoring whitespace and comments.

PARTICIPATION ACTIVITY 1.9.3: A compiler scans code line-by-line, left-to-right; whitespace is mostly irrelevant.

Animation captions:

1. The compiler converts a high level program into an executable program using Java bytecode.
2. Comments do not generate machine code.
3. The compiler recognizes end of statement by semicolon ";"

PARTICIPATION ACTIVITY 1.9.4: Compiling code with whitespace and comments.

1) Spaces are always ignored by the compiler.

   - True
   - False

2) How many spaces will the compiler ignore in the code below?

```
numToBuy = numNeeded - numInStock +
2;
```

   - 3
   - 6
   - 7

3) How many lines will the compiler ignore in the code below?

```
int userAge;
int currentDecade;
int nextDecade;
int nextMilestone;

// FIXME: Get user age
userAge = 29; // Testing with 29

currentDecade = userAge / 10;
nextDecade = currentDecade + 1;
nextMilestone = nextDecade * 10;
```

- 1
- 2
- 3

1.10 Why whitespace matters

Whitespace and precise formatting

For program output, ***whitespace*** is any blank space or newline. Most coding activities strictly require a student program's output to exactly match the expected output, including whitespace. Students learning programming often complain:

"My program is correct, but the system is complaining about output whitespace. "

However, correctness often includes output being formatted correctly.

PARTICIPATION ACTIVITY 1.10.1: Precisely formatting a meeting invite.

Animation content:

```
undefined
```

Animation captions:

1. This program for online meetings not only does computations like scheduling and creating a unique meeting ID, but also outputs text formatted neatly for a calendar event.
2. A calendar program may append more text after the meeting invitation text.
3. The programmer of the invitation on the right wasn't careful with whitespace. "Join meeting" is buried, the link is hard to see, and the "Phone" text is dangling at a line's end.
4. The programmer also didn't end with a newline, causing subsequent text to appear at the end of a line, and even wrap to the next line. This output looks unprofessional.

PARTICIPATION 1.10.2: Program correctness includes correctly formatted output

ACTIVITY 1.10.2: Program correctness includes correctly-formatted output.

Consider the example above.

1) The programmer on the left intentionally inserted a newline in the first sentence, namely "Kia Smith ... video meeting". Why?
   - Probably a mistake
   - So the text appears less jagged
   - To provide some randomness to the output

2) The programmer on the right did not end the first sentence with a newline. What effect did that omission have?
   - "Join meeting" appears on the same line
   - No effect

3) The programmer on the left neatly formatted the link, the "Phone:" text, and phone numbers. What did the programmer on the right do?
   - Also neatly formatted those items
   - Output those items without neatly formatting

4) On the right, why did the "Reminder..." text appear on the same line as the separator text "------"?
   - Because programs behave erratically
   - Because the programmer didn't end the output with a newline

5) Whitespace _____ important in

program output.

- ○ is
- ○ is not

Programming is all about precision

Programming is all about *precision*. Programs must be created precisely to run correctly. Ex:

- = and == have different meanings.
- Using i where j was meant can yield a hard-to-find bug.
- Declaring a variable as int when char was needed can cause confusing errors.
- Not considering that n could be 0 in sum/n can cause a program to fail entirely in rare but not insignificant cases.
- The difference between typing x/2 vs. x/2.0 can have huge impacts.
- Counting from i being 0 to i < 10 vs. i <= 10 can mean the difference between correct output and a program outputting garbage.

In programming, every little detail counts. Programmers must get in a mindset of paying extreme *attention to detail*.

Thus, another reason for caring about whitespace in program output is to help new programmers get into a "precision" mindset when programming. Paying careful attention to details like whitespace instructions, carefully examining feedback regarding whitespace differences, and then modifying a program to exactly match expected whitespace is an exercise in strengthening attention to detail. Such attention can lead programmers to make fewer mistakes when creating programs, thus spending less time debugging, and instead creating programs that work correctly.

PARTICIPATION ACTIVITY 1.10.3: Thinking precisely, and attention to detail.

Programmers benefit from having a mindset of thinking precisely and paying attention to details. The following questions emphasize attention to detail. See if you can get all of the questions correct on the first try.

1) How many times is the letter F (any case) in the following?
If Fred is from a part of France, then of course Fred's French is good.

Check Show answer

2) How many differences are in these two lines?
Printing A linE is done using printIn
Printing A linE is done using print1n

Check Show answer

3) How many typos are in the following common phrase?
Keep calmn and cary on.

Check Show answer

4) If I and E are adjacent, I should come before E, except after C (where E should come before I). How many violations are in the following?
BEIL CEIL ZIEL YIEIK TREIL

Check Show answer

5) A password must start with a letter, be at least 6 characters long, include a number, and include a special symbol. How many of the following

passwords are valid?
hello goodbye Maker1 dog!three
Oops_again 1augh#3

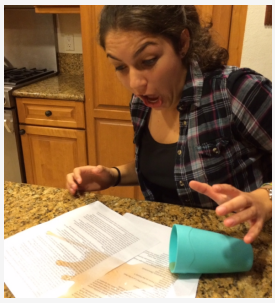
Check Show answer

Programmer attention to details

The focus needed to answer the above correctly on the first try is the kind of focus needed to write correct programs. Due to this fact, some employers give "attention to detail" tests to people applying for programming positions. See for example this test, or this article discussing the issue. Or, just web search for "programmer attention to details" for more such tests and articles.

1.11 Errors and warnings

Syntax errors



People make mistakes. Programmers thus make mistakes—lots of them. One kind of mistake, known as a ***syntax error***, is to violate a programming language's rules on how symbols can be combined to create a program. An example is forgetting to end a statement with a semicolon.

A compiler generates a message when encountering a syntax error. The following program is missing a semicolon after the first output statement.

Figure 1.11.1: Compiler reporting a syntax error.

```
1:
2:
3:
4:
5:
6:
7:
```

```
public class Traffic {
   public static void main(String []
args) {

      System.out.print("Traffic
today")
      System.out.println(" is very
light.");
   }
}
```

```
Traffic.java:4: ';' expected
      System.out.print("Traffic
today")

^
1 error
```

Above, the 4 refers to the 4th line in the code.

zyDE 1.11.1: Syntax errors often exist before the reported line.

The program below has a syntax error on line 3. The = at the end of the statement `int playerScore=` should be a semicolon.

Before correcting the error, press Run, and notice that the error message is for a line comes AFTER that statement, because that later line is where the compiler got confu

Replace the = with a semicolon, and press Run again. Your program should work cor now.

Load default template...

Run

```
1  public class DisplayPlayer {
2     public static void main(String [
3        int playerScore=
4        int playerNum;
5
6        playerNum = 1;
7        playerScore = 100;
8        System.out.println(playerNum)
9        System.out.println(playerScor
10    }
11 }
12
```

PARTICIPATION ACTIVITY 1.11.1: Syntax errors.

Find the syntax errors. Assume variable numDogs has been declared.

1) `System.out.print(numDogs).`

- Error
- No error

2) `System.out.print("Dogs: " numDogs);`

- Error
- No error

3) `system.out.print("Everyone wins.");`

- Error
- No error

4) `System.out.print("Hello friends!);`

- Error
- No error

5) `System.out.print("Amy // Michael");`

- Error
- No error

6) `System.out.print(NumDogs);`

- Error
- No error

7)
```
int numCats
numCats = 3;
System.out.print(numCats);
```

- Error
- No error

8) `System.print(numDogs);`

- Error
- No error

Unclear error messages

Compiler error messages are often unclear or even misleading. The message is like the compiler's "best guess" of what is really wrong.

Figure 1.11.2: Misleading compiler error message.

```
1: public class Traffic {
2:    public static void main(String [] args) {
3:
4:       System.out.print "Traffic today ";
5:       System.out.println("is very light.");
6:    }
7: }
```

```
Traffic.java:4: error: not a statement
        System.out.print "Traffic today ";
                         ^
Traffic.java:4: error: ';' expected
        System.out.print "Traffic today ";
                                         ^
```

The compiler indicates a missing semicolon ';'. But the real error is the missing parentheses.

Sometimes the compiler error message refers to a line that is actually many lines past where the error actually occurred. Not finding an error at the specified line, the programmer should look to *previous* lines.

PARTICIPATION ACTIVITY 1.11.2: The compiler error message's line may be past the line with the actual error.

Animation captions:

1. The compiler hasn't yet detected the error.
2. Now the compiler is confused, so it generates a message. But the reported line number is past the actual syntax error.
3. Upon not finding an error at line 5, the programmer should look at earlier lines.

PARTICIPATION ACTIVITY 1.11.3: Unclear error messages.

1) When a compiler says that an error exists on line 5, that line must have an error.
   - True
   - False

2) If a compiler says that an error exists on line 90, the actual error may be on line 91, 92, etc.
   - True
   - False

3) If a compiler generates a specific message like "missing semicolon", then a semicolon must be missing somewhere, though maybe from an earlier line.
   - True
   - False

Fixing the first error

Some errors create an upsettingly long list of error messages. Good practice is to focus on fixing just the first error reported by the compiler and then recompiling. The remaining error messages may be real but are more commonly due to the compiler's confusion caused by the first error and are thus irrelevant.

Figure 1.11.3: Good practice for fixing errors reported by the compiler.

1. Focus on FIRST error message, ignoring the rest.
2. Look at reported line of first error message. If error found, fix. Else, look at previous few lines

few lines.
3. Compile, repeat.

zyDE 1.11.2: Fixing syntax errors.

Click run to compile, and note the long error list. Fix only the first error, then recompil
that process (fix first error, recompile) until the program compiles and runs. *Expect* tc
misleading error messages as well as errors that occur before the reported line numl

Load default template...

Run

```
1  public class BeansInJars {
2     public static void main (String
3        int numBeans
4        int numJars;
5        inl totalBeans;
6
7        numBeans = 500;
8        numJars = 3;
9
10       System.out.print(numBeans + "
11       System.out.print(numJar + " j
12       totalBeans = numBeans * numJa
13       Systems.out.println(totalBean
14    }
15 }
```

PARTICIPATION ACTIVITY 1.11.4: Fixing the first error.

A compiler generates the following error messages:
Line 7: Missing semicolon
Line 9: numItems not defined
Line 10: Expected '('

1) The programmer should start by examining line ____.

- 7

- 9
- 10

2) If the programmer corrects an error on line 7, the programmer should _____.

- check earlier lines too
- compile
- check line 9

3) If the programmer does NOT find an error on line 7, the programmer should check line _____.

- 6
- 8
- 9

CHALLENGE ACTIVITY 1.11.1: Basic syntax errors.

Type the statements. Then, correct the one syntax error in each statement. Hints: Statements end in semicolons, and string literals use double quotes.

```
System.out.printl("Predictions are hard.");
System.out.print("Especially ');
System.out.println("about the future.").
System.out.println("Num is: " - userNum);
```

426488.2706032.qx3zqy7

```
1 import java.util.Scanner;
2
3 public class Errors {
4    public static void main(String [] args) {
5       int userNum;
6
```

```
      userNum = 5;

      /* Your solution goes here  */

   }
}
```

Run

View your last submission

CHALLENGE ACTIVITY 1.11.2: More syntax errors.

Retype the statements, correcting the syntax errors.

```
System.out.println("Num: " + songnum);
System.out.println(int songNum);
System.out.println(songNum " songs");
```

Note: These activities may test code with different test values. This activity will perform two tests: the first with songNum = 5, the second with songNum = 9. See How to Use zyBooks.

426488.2706032.qx3zqy7

```
import java.util.Scanner;

public class Errors {
   public static void main (String [] args) {
      int songNum;

      songNum = 5;

      /* Your solution goes here  */

   }
}
```

Run

View your last submission

Logic errors

Because a syntax error is detected by the compiler, a syntax error is known as a type of ***compile-time error***.

New programmers commonly complain: "The program compiled perfectly but isn't working." Successfully compiling means the program doesn't have compile-time errors, but the program may have other kinds of errors. A ***logic error***, also called a ***bug***, is an error that occurs while a program runs. For example, a programmer might mean to type `numBeans * numJars` but accidentally types `numBeans + numJars` (+ instead of *). The program would compile but would not run as intended.

Figure 1.11.4: Logic errors.

```
public class BeansInJars {
   public static void main (String [] args) {
      int numBeans;
      int numJars;
      int totalBeans;

      numBeans = 500;
      numJars = 3;

      System.out.print(numBeans + " beans in ");
      System.out.print(numJars + " jars yields ");
      totalBeans = numBeans + numJars; // Oops, used + instead
of *
      System.out.println(totalBeans + " total");
   }
}
```

zyDE 1.11.3: Fix the bug.

Click run to compile and execute and then note the incorrect program output. Fix the the program.

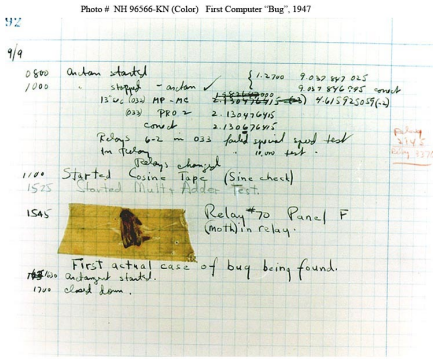
Load default template...

Run

```
1  // This program has a bug that caus
2  // Can you find the bug?
3  public class BeansInJars {
4     public static void main (String
5        int numBeans;
6        int numJars;
7        int totalBeans;
8
9        numBeans = 500;
10       numJars = 3;
11
12       System.out.print(numBeans + "
13       System.out.print(numJars + "
14       totalBeans = numBeans * numJa
15       System.out.println("totalBean
```

Bugs

The term *bug* to describe a runtime error was popularized when in 1947 engineers discovered their program on a Harvard University Mark II computer was not working because a moth was stuck in one of the relays (a type of mechanical switch). They taped the bug into their engineering log book, still preserved today (The moth).



Compiling frequently

Good practice, especially for new programmers, is to compile after writing only a few lines of code, rather than writing tens of lines and then compiling. New programmers commonly write tens of lines before compiling, which may result in an overwhelming number of compilation errors and warnings and logic errors that are hard to detect and correct.

PARTICIPATION ACTIVITY 1.11.5: Compile and run after writing just a few statements.

Animation captions:

1. Writing many lines of code without compiling and running is bad practice.
2. New programmers should compile and run programs after every few lines. Even experienced programmers compile and run frequently.

PARTICIPATION ACTIVITY 1.11.6: Compiling and running frequently.

1) A new programmer writes 5 lines of code, compiles and runs, writes 5 more lines, and compiles and runs again. The programmer is _____ .
   - wasting time
   - following good practice

2) An experienced programmer writes 80 lines of code and then compiles and runs. The programmer is probably _____ .
   - programming dangerously
   - following good practice

Compiler warnings

A compiler will sometimes report a ***warning***, which doesn't stop the compiler from creating an

executable program but indicates a possible logic error. Ex: Some compilers will report a warning like "Warning, dividing by 0 is not defined" if encountering code like: `totalItems = numItems / 0` (running that program does result in a runtime error). Even though the compiler may create an executable program, good practice is to write programs that compile without warnings.

By default, Java compilers don't print every possible warning. Ex: A programmer's Java code may use something old that has a newer, better alternative. The compiler recognizes this, but since the code may still run fine, no warning is given. So, many programmers recommend the good practice of configuring compilers to be more picky with warnings than the default. Ex: javac can be run as `javac -Xlint yourfile.java` to enable all recommended warnings. Programmers often feel more confident about code that compiles with no warnings when -Xlint is used.

PARTICIPATION ACTIVITY 1.11.7: Compiler warnings.

1) A compiler warning by default will prevent a program from being created.
   - True
   - False

2) Generally, a programmer should not ignore warnings.
   - True
   - False

3) A compiler's default settings cause most warnings to be reported during compilation.
   - True
   - False

1.12 Problem solving

Programming languages vs. problem solving

A chef may write a new recipe in English, but creating a new recipe involves more than just knowing English. Similarly, creating a new program involves more than just knowing a programming language. Programming is largely about ***problem solving***: creating a methodical solution to a given task.

The following are real-life problem-solving situations encountered by one of this material's authors.

Example 1.12.1: Solving a (nonprogramming) problem: Matching socks.

A person stated a dislike for matching socks after doing laundry, indicating there were three kinds of socks. A friend suggested just putting the socks in a drawer and finding a matching pair each morning. The person said that finding a matching pair could take forever: Pulling out a first sock and then pulling out a second, placing them back and repeating until the second sock matches the first could go on many times (5, 10, or more).

The friend provided a better solution approach: Pull out a first sock, then pull out a second, and repeat (without placing back) until a pair matches. In the worst case, if three kinds of socks exist, then the fourth sock will match one of the first three.

PARTICIPATION ACTIVITY 1.12.1: Matching socks solution approach.

Exactly three sock types A, B, and C exist in a drawer.

1) If sock type A is pulled first, sock type B second, and sock type C third, the fourth sock type must match one of A, B, or C.
   - ○ True
   - ○ False

2) If socks are pulled one at a time and kept until a match is found, at least four pulls are necessary.

- ○ True
- ○ False

3) If socks are pulled two at a time and put back if not matching, and the process is repeated until the two pulled socks match, the maximum number of pulls is 4.

- ○ True
- ○ False

Example: Greeting people

PARTICIPATION ACTIVITY 1.12.2: Greeting people problem.

An organizer of a 64-person meeting wants to start by having every person individually greet each other person for 30 seconds. Indicate whether the proposed solution achieves the goal without using excessive time. Before answering, think of a possible solution approach for this seemingly simple problem.

1) Form an inner circle of 32 and an outer circle of 32, with people matched up. Every 30 seconds, have the outer circle shift left one position.

- ○ Yes
- ○ No

2) Pair everyone randomly. Every 30 seconds, tell everyone to find someone new to greet. Do this 63 times.

- ○ Yes
- ○ No

- No

3) Have everyone form a line. Then have everyone greet the person behind them.

- Yes
- No

4) Have everyone form a line. Have the first person greet the other 63 people for 30 seconds each. Then have the second person greet each other person for 30 seconds each (skipping anyone already met). And so on.

- Yes
- No

5) Form two lines of 32 each, with attendees matched up. Every 30 seconds, have one line shift left one position (with the person on the left end wrapping to right). Once the person that started on the left is back on the left, then have each line split into two matched lines, and repeat until each line has just 1 person.

- Yes
- No

Example: Sorting name tags

Example 1.12.2: Example: Sorting name tags.

1,000 name tags were printed and sorted by first name into a stack. A person wishes to instead sort the tags by last name. Two approaches to solving the problem are:

- Solution approach 1: For each tag, insert that tag into the proper location in a new

last-name-sorted stack.

- Solution approach 2: For each tag, place the tag into one of 26 substacks, one for last names starting with A, one for B, etc. Then, for each substack's tags (like the A stack), insert that tag into the proper location of a last-name-sorted stack for that letter. Finally combine the stacks in order (A's stack on top, then B's stack, etc.).

Solution approach 1 will be very hard; finding the correct insertion location in the new sorted stack will take time once that stack has about 100 or more items. Solution approach 2 is faster, because initially dividing into the 26 stacks is easy, and then each stack is relatively small, so insertions are easier to do.

In fact, sorting is a common problem in programming, and solution approach 2 is similar to a well-known sorting approach called radix sort.

PARTICIPATION ACTIVITY 1.12.3: Sorting name tags.

1,000 name tags are to be sorted by last name by first placing tags into 26 unsorted substacks (for A's, B's, etc.), then sorting each substack.

1) If last names are equally distributed among the alphabet, what is the largest number of name tags in any one substack?
   - ○ 1
   - ○ 39
   - ○ 1,000

2) Suppose the time to place an item into one of the 26 sub-stacks is 1 second. How many seconds are required to place all 1000 name tags onto a sub-stack?
   - ○ 26 sec
   - ○ 1,000 sec
   - ○ 26,000 sec

3) When sorting each substack, suppose

3) When sorting each substack, suppose the time to insert a name tag into the appropriate location of a sorted N-item sub-stack is N * 0.1 sec. If the largest substack is 50 tags, what is the longest time to insert a tag?

- 5 sec
- 50 sec

4) Suppose the time to insert a name tag into an N-item stack is N * 0.1 sec. How many seconds are required to insert a name tag into the appropriate location of a 500-item stack?

- 5 sec
- 50 sec

A programmer usually should carefully create a solution approach *before* writing a program. Like English being used to describe a recipe, the programming language is just a description of a solution approach to a problem; creating a good solution should be done first.

1.13 Why programming

Computing careers

While careers in law, medicine, and engineering have existed for hundreds of years, computers are relatively new so careers in computing are new too. Today, computing jobs are often ranked among the best jobs, in terms of opportunity, salary, work-life balance, job security, job satisfaction, work conditions, etc. Nearly all computing jobs require some training in programming; some jobs then focus on programming, while others instead focus on related aspects.

In a 2019 ranking (below), the top job is software developer. In another ranking, 3 of the top 20 were computing jobs. Note: Rankings from different sources vary greatly; some have more engineers, human resources managers, data scientists, marketing, etc. Also, the specific ordering in a ranking is not usually substantial (like rank #2 vs. #5), and rankings change every year. However, note that

most rankings consistently have several computing jobs in the top tier.

Table 1.13.1: Best jobs of 2019, per U.S. News and World Report.

The rankings are based off growth potential, work-life balance, and salary.

Ranking	Occupation	Description
1	Software developer	Designs computer programs, combining creativity and technical know-how, often working in teams.
2-4	Statistician, physician's assistant, dentist	
5 (tie)	Orthodontist, Nurse Anesthetist	
7-8	Nurse Practitioner, Pediatrician	
9 (tie)	Obstetrician and Gynecologist, Oral and Maxillofacial Surgeon, Prosthodontist, Physician	

Source: U.S. News and World Report (includes links to expanded descriptions), 2020.

PARTICIPATION ACTIVITY 1.13.1: Computing jobs are often ranked among the best jobs.

1) What factor was used to rank the best jobs?
   - Salary
   - Job security
   - Multiple factors were considered

2) Software developers spend nearly all their time alone at a computer.

- ○ True
- ○ False

3) Interestingly, the above list is dominated by jobs in what two general areas?

- ○ Computing, and health care
- ○ Computing, and manufacturing

Types of computing jobs

Table 1.13.2: Computing jobs.

A wide variety of computing jobs exist.

Occupation	Job Summary	Entry-level education	2018 median pay
Computer and Information Research Scientists	Computer and information research scientists invent and design new approaches to computing technology and find innovative uses for existing technology. They study and solve complex problems in computing for business, medicine, science, and other fields.	Doctoral or professional degree	$111,370

Computer Network Architects	Computer network architects design and build data communication networks, including local area networks (LANs), wide area networks (WANs), and intranets. These networks range from a small connection between two offices to a multinational series of globally distributed communications systems.	Bachelor's degree	$109,020
Computer Programmers	Computer programmers write code to create software programs. They turn the program designs created by software developers and engineers into instructions that a computer can follow.	Bachelor's degree	$84,280
	Computer		

Computer Support Specialists	support specialists provide help and advice to people and organizations using computer software or equipment. Some, called computer network support specialists, support information technology (IT) employees within their organization. Others, called computer user support specialists, assist non-IT users who are having computer problems.	Varies: High-school degree and higher	$53,470
Computer Systems	Computer systems analysts study an organization's current computer systems and procedures and design information systems solutions to help		

Computer Systems Analysts	solutions to help the organization operate more efficiently and effectively. They bring business and information technology (IT) together by understanding the needs and limitations of both.	Bachelor's degree	$88,740
Database Administrators	Database administrators (DBAs) use specialized software to store and organize data, such as financial information and customer shipping records. They make sure that data are available to users and are secure from unauthorized access.	Bachelor's degree	$90,070
	Information security analysts plan and carry out security measures to protect an		

Information Security Analysts	organization's computer networks and systems. Their responsibilities are continually expanding as the number of cyberattacks increase.	Bachelor's degree	$98,350
Network and Computer Systems Administrators	Computer networks are critical parts of almost every organization. Network and computer systems administrators are responsible for the day-to-day operation of these networks.	Bachelor's degree	$82,050
Software Developers	Software developers are the creative minds behind computer programs. Some develop the applications that allow people to do specific tasks on a computer or other device. Others develop the underlying	Bachelor's degree	$105,590

	systems that run the devices or control networks.		
Web Developers	Web developers design and create websites. They are responsible for the look of the site. They are also responsible for the site's technical aspects, such as performance and capacity, which are measures of a website's speed and how much traffic the site can handle. They also may create content for the site.	Associate's degree	$69,430

Source: bls.gov (includes links to detailed descriptions and outlooks for each occupation).

PARTICIPATION ACTIVITY 1.13.2: Computing jobs.

Refer to the above BLS table of computing jobs.

Computer systems analysts **Software developers** **Information security analysts**

Computer support specialists | Web developers | Computer programmers

	Likely requires both a strong knowledge of computer technology, and excellent interpersonal skills due to dealing with non-technical users.
	Create, design, and program software.
	Help write programs created by software developers.
	Help organizations use computing technology to operate effectively. Requires strong combination of business and computing technology knowledge.
	Focus on protecting an organization's computers and data. Increasingly important as "hackers" continue to steal huge amounts of data, as widely-publicized in recent years.
	Build websites, which may involve the look/feel, the content, the performance of the site, and more.

Reset

For many non-computing jobs (dentist, attorney, nurse, business, etc.), computer usage is high, and thus knowledge of computing technology can yield strong advantages even for people not in a computing career.

Programming and non-computing jobs

Programming and non-computing jobs

Many people in non-computing jobs find that knowing some programming can benefit their careers. Some examples:

- *Kelly* majored in chemistry, and now works as a scientist in a pharmaceutical company. Kelly helps analyze clinical trials. Her company uses commercial statistical software, but she found that writing small custom programs yielded even better analyses. Her co-workers now come to her for help. She is glad she took a required programming class in college, though at the time she wasn't as happy about it.
- *Paul* majored in civil engineering, and now authors technical content for a large company. Paul noticed that several authoring tasks done in Google Docs by the in-house 25-person authoring team could be automated. Building on the programming he learned in a required college course, Paul spent several hours online learning about Google Docs "add on" programming, and wrote two small add-ons. His add-on programs have become part of the standard authoring process for the entire team, who frequently thanks Paul for saving them time and relieving them of tedious tasks.
- *Ethan* majored in business, and got a job in sales operations of a Silicon Valley startup company. Building on the C++ programming he learned from a college course, he started tinkering with writing database query programs using "SQL", and discovered he had a knack for it. His job duties have expanded to include running database reports, and he has automated dozens of reports via programming, helping people throughout the company be more productive.
- *Eva* (pictured above) majored in environmental science. She voluntarily took a programming course in college believing the knowledge/skills could be important to her. She took a job at a startup company doing various marketing tasks. She began to manage the company's website, and realized that a few small programs could make the web pages dynamic and interactive. She wrote the code herself, which was reviewed and approved by the engineering team and became part of the company's live website. She plans on getting a graduate degree in environmental science and expects programming will be useful in her research.

PARTICIPATION ACTIVITY | 1.13.3: Programming in non-computing jobs.

Consider the examples above.

1) Kelly voluntarily took a programming course in college.

- True
- False

2) Ethan learned SQL programming in a college course and now applies SQL programming in his job.

- True
- False

3) Eva voluntarily took a programming class in college.

- True
- False

Precision, logic, and computational thinking

Many people find that programming encourages precise, logical thought that can lead to better writing and speaking, clearer processes, and more. The thought processes needed to build correct, precise, logical programs is sometimes called ***computational thinking*** and has benefits beyond programming.

PARTICIPATION ACTIVITY | 1.13.4: Learning programming tends to aid in precise, logical thought, aspects of computational thinking.

Animation captions:

1. Common English usage may be vague. Are workers, painters, and contractors the same people or different? What exactly is white and brown?
2. Programs use one word per item; no synonyms, no pronouns. In English, using "painters" consistently, and replacing "they" with "The IDs", yields precise info.
3. Policies and other documents often aren't logical, with conflicting or missing info. How can a person 20 miles away take a taxi if they must drive? What about 100 miles?
4. Programmers use precise structures like "If-else" statements. When used in English, the result is logical, unambiguous info. Some call this "computational thinking".

New programmers often complain about how unforgiving programming is, but such attention to detail is one of the benefits of learning programming.

detail is one of the benefits of learning programming.

PARTICIPATION ACTIVITY 1.13.5: Computational thinking.

1) What's wrong with this survey question?

How many minutes did you spend?
__ Under 5
__ 6 or more

- Should say "More than 6" instead of "6 or more".
- Exactly 5 minutes is not a choice

2) An online shopping site allows setting up a recurring order. A person needs to determine the order frequency for laundry detergent. One bottle does 64 loads. He does a load a week. His wife does a load a week. His daughter does a load every two weeks. What's the best frequency?

- Every 24 weeks
- Every 32 weeks
- Every 64 weeks

You've never done anything like this

Programming is different than nearly anything most students have done before. Most new programmers initially struggle. Just as a child learning to walk will stumble and fall, a student learning to program will stumble and fall many times as well.

Programs have literally transformed the world in the past few decades. But, *correct programs are hard to create*. Programs are among the most sophisticated of human creations. Even one wrong symbol in a program with thousands of characters can cause the program to entirely fail. And programs deal with doing long sequences of tasks over time. Such features are not common in

programs deal with doing long sequences of tasks over time. Such features are not common in other aspects of life.

Programming is a combination of concepts and skill. The skill part is not as common in other "academic" subjects. Learning to program thus requires practice. A student cannot watch a piano teacher play and then walk away playing piano. Writing correct expressions, properly formed if-else branches, correctly working loops, etc., requires repeated attempts, and, like the new piano player, lots of mistakes along the way.

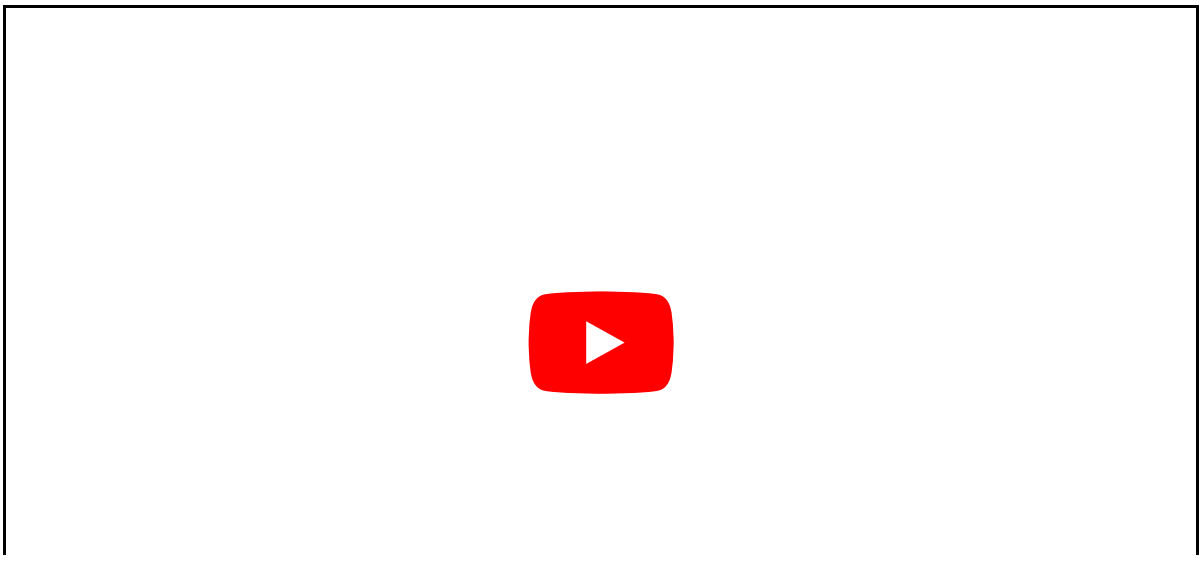
Programming also requires a lot of mental energy. No easy steps exist for how to solve a given problem by writing a program. Many students are not accustomed to having to think so hard to solve a problem, instead looking to follow standard steps or just trying to "look up the answer".

Students studying programming are about to embark on one of the most rewarding but also the most challenging of human endeavours. When stuck, students may wish to take solace that everyone struggles. Like the child learning to walk, each fall hurts, but know that each fall brings one closer to learning a powerful skill.

Even the best programmers make mistakes

Even the best programmers make mistakes. In San Diego 2012, a software bug caused 17-minutes of fireworks to launch nearly simultaneously.

Video 1.13.1: When software goes wrong...



PARTICIPATION ACTIVITY 1.13.6: Programming.

1) For most people, programming comes easy.
   - True
   - False

2) If a student has trouble converting a problem statement into a program, the teacher and/or learning content must have done a poor job.
   - True
   - False

1.14 Language history

In 1978, Brian Kernighan and Dennis Ritchie at AT&T Bell Labs (which used computers extensively for automatic phone call routing) published a book describing a new high-level language with the simple name ***C***, being named after another language called B (whose name came from a language called BCPL). C became the dominant programming language in the 1980s and 1990s.

In 1985, Bjarne Stroustrup published a book describing a C-based language called ***C++***, adding constructs to support a style of programming known as *object-oriented programming*, along with other improvements. The unusual ++ part of the name comes from ++ being an operator in C that increases a number, so the name C++ suggests an increase or improvement over C.

In 1991, James Gosling and a team of engineers at Sun Microsystems (acquired by Oracle in 2010) began developing the ***Java*** language with the intent of creating a language whose executable could be ported to different processors, with the first public release in 1995.

The language had a C/C++ style and for portability reasons was designed to execute on a virtual machine. Java was initially intended to be run on consumer appliances like interactive TVs. Web

browsers like Netscape Navigator began providing support for running Java programs within the browser, bringing much attention to the language. The Java language continues to evolve for the programming of traditional computers and consumer devices. Java should not be confused with JavaScript, which is an entirely different language used for developing web applications that was named similarly.

A June 2019 survey ranking languages by their popularity, based on programming related searches using popular search engines, yielded the following:

Table 1.14.1: Top languages ranked by popularity.

Language	Percentage
Java	15%
C	13%
Python	9%
C++	7%
Visual Basic .NET	5%
C#	4%
JavaScript	3%
PHP	3%
SQL	2%
Assembly language	1%
Swift	1%

(Source: https://www.tiobe.com/tiobe-index/, 2019)

PARTICIPATION ACTIVITY 1.14.1: C/C++ history.

1) In what year was the first C book published?

Check Show answer

2) In what year was the first C++ book published?

Check Show answer

PARTICIPATION ACTIVITY 1.14.2: Java history.

1) When was the first public release of Java?

Check Show answer

1.15 Course Expectations

How much time should you expect to spend on this class each week?

This course is designed for students with no prior Computer Science or programming experience. However, you should be aware that this course is consistently rated by students to be one of the most difficult and challenging (but rewarding) classes that they take.

This is a 3 credit hour course, and it is expected that you will spend **9 to 12 hours per week**

working through the materials for this course - that includes reading, videos, activities, challenges, quizzes, labs, programming assignments, review, and Study Hall and Office Hours.

Some students may have prior experience with programming. If so, you may be able to work through this course a little quicker (9 hours per week), while students with less or no prior experience will need to spend more time (12 hour per week). On average you should expect to spend 10 hours per week on the materials and assignments for this course.

You should put these weekly hours on your calendar (when, specifically, you will be working through the course content, activities, and assignments). It is best to spread these hours across the week and spend some time each day, rather than trying to complete all the work in only one or two days. We recommend scheduling 2 hours per day for Monday through Friday.

During the hours over each week, you should prioritize your time and activities in this way:

- 5 hr/wk - Content (reading, videos, textbook activities)
- 1 hr/wk - Lab Activity
- 3 hr/wk - Individual Coding Assignments
- 1 hr/wk - Study Hall and Office Hours

Especially be sure to give yourself plenty of time to work through the zyLab Individual Coding Assignments. Do not expect that you will be able to simply start and finish these Individual Coding Assignments in one setting. It is expected and completely normal that you should get stuck while working through these assignments, and you will need time to get unstuck. We will discuss some strategies for getting unstuck below.

How should you spend your time?

Content (reading, videos, textbook activities):
You should spend about **50% of your time** (~5 hrs/wk) for this course by working through the interactive textbook in order. In the interactive textbook you will encounter text, videos, activities, quizzes, and challenges. You should work through these in the order in which they are presented. Each item builds on the previous ones. Do not rush through the items and activities, or you will find that you are not able to complete the subsequent challenges, quizzes, and coding assignments. By going slowly and thoroughly, rather than quickly and carelessly, you will find that you save more time and make quicker progress in the long run.

time and make quicker progress in the long run.

Even when you go slowly and deliberately and take your time through the textbook, you will encounter challenges and quizzes, and programming assignments that you will struggle with. This is normal and expected. In these cases your first step should be to back up and review the previous relevant sections of the textbook. Additional Help Resources will also be available.

Labs:
Your labs will be facilitated through online video activities that lead you through a coding activity. To prepare you should, at a minimum, read through the lab instructions prior to the lab meeting and ensure that you are confident in your understanding of the Required Skills needed to successfully complete the lab assignment.

About **10% of your time** (~1 hr/wk) each week should be spent on these labs.

Individual Coding Assignments:
Each week you will have one or more individual coding assignments that you must complete. Unlike the lab assignments which are collaborative, you must work on and complete these individual assignments on your own. You **may not** collaborate or share work with other students or anyone outside of this class. You **may not** use any online resources outside of those directly provided in the content for this course. If you need help, you must use only the Help Resources directly provided in this course.

About **30% of your time** (~3 hrs/wk) each week should be spent on these individual coding assignments. Some weeks may take more or less time, depending on many factors, including how well you understood the content, prior experience, etc. On average, you should expect to spend ~3 hrs/wk working on these Individual Coding Assignments.

It can be very helpful to read through some of the Individual Coding Assignments - even before engaging the other textbook content for the week. Previewing the assignments can help you identify and focus on the topics, concepts, and skill you will need in order to succeed on the assignments. Do not wait until too late in the week to start the Individual Coding Assignments. Starting early will give you the time you need to struggle with the assignments (which is expected and normal), and to review content and get additional help when you need it.

Study Hall and Office Hours:

Study Hall and Office Hours:

TAs and Instructors will hold regularly scheduled study hall and office hours each week. These sessions will be available across a large range of days and times. The study hall sessions are an excellent place to work on your individual coding assignments. This strategy ensures that there is someone immediately available to help you if you may need it. Times, meeting locations and links will be posted in the announcements on the course website.

About **10% of your time** (~1 hr/wk) each week should be spent attending study hall and/or office hours each week. It is highly recommended that you should spend at least 1 hr/wk engaging in these Study Hall and Office Hours sessions. The Study Hall sessions are an excellent time and place to work on your Individual Coding Assignments. This way, if you do get stuck, there is always someone directly available who can help you. You can learn a lot in these Study Hall and Office Hours sessions, even if you feel you do not need any help.

What should you do if you get stuck while working on a challenge, quiz, or assignment? How to get help?

Getting stuck on an assignment is normal, and will happen to everyone at some point in this course.

First, look through the previous chapter and activities to see if there is content that may help answer your question.

Your next step should be to review the posts in Inscribe related to that topic. The topic discussion is linked by the **blue Get Help Button** in each lab and Individual Assignment. You may very well find that your questions have already been posted, and answers have already been given. If not, then you can be the first to post these questions. **Course TAs and Instructors will be monitoring the discussion on a regular basis, and you should generally get a reply on the same day**. Students are also encouraged to help each other and answer questions posted in the discussions when they are able.

If you're not able to get an answer through Inscribe within a day, try some of these options:

1. Visit the Programming Tutoring Center on campus, or online!
2. Visit the CSE110 Study Hall. This is a great place to be when you're working on your individual assignments. See announcements on the course website for details

assignments. See announcements on the course website for details.

3. Attend CSE110 Office Hours or Meet with an instructor. See announcements on the course website for details.

Do not turn to google for help. Why? Many resources online are not tailored to help you learn how to program on your own, and may confuse you or lead you to copy solutions. **Our teaching assistants are highly trained** to support your learning this material - and your tuition fees are already paying for it! Talk to someone before turning to Google or Stackoverflow.

1.16 Basics

Participation activities

In a zyBook, a **participation activity** (PA) is an activity used in the initial learning of a topic. A PA is a more engaging way of reading. As one student put it, "It's like the material is working *with* me, rather than talking *to* me." Anyone can get all PA points just by participating (hence the name).

PA's are labeled and numbered as: "*Participation Activity* 1.1.1". A zyBook records student completion of each PA. An instructor may provide course points for completing PA's. Completing means:

- Learning questions: *Eventually* entering a correct answer.
- Animations: Watching all steps in their entirety.
- Tools: Interacting with the tool; some tools require reaching "done".

Learning questions have no penalty if a student answers wrong the first time (or multiple times) or if a student shows themselves the answer.

Some answers involve typing text like x + 1 or a number like 20. The tool that checks correctness uses text matching (rather than expression evaluation) to enable convenient use on various platforms with varying Internet access. Supporting all possible (non-standard) answers isn't usually practical. Following instructions carefully helps. If a student thinks an answer is correct but isn't being accepted, the student can just click "Show answer".

PARTICIPATION ACTIVITY 1.16.1: Participation activities.

1) A small point deduction applies if a participation activity's question is not answered correctly the first time

answered correctly the first time.

- True
- False

2) Clicking "Show answer" means no points can later be gained for this question.

- True
- False

3) When a participation activity's question is answered, an explanation appears. Such explanations can usually be skipped.

- True
- False

4) Answering a question, even incorrectly, counts as *completing* the question.

- True
- False

5) Animations can be completed by clicking on each step, but not necessarily watching each step entirely.

- True
- False

Extremely fast clicking through activities, always clicking "Show answer" before answering, or first typing bogus answers, are easily detectable. Most students really try the activities, which are designed to help. Feedback is excellent, and research shows students learn from such activities (research summaries). About 90% of students earnestly attempt the PA's, realizing PA's are effective and respect student time (SIGCSE 2017 paper).

Viewing PA completion in a section

PA completion can be viewed by the filled-in icons next to each activity.

Figure 1.16.1: PA completion icons filling in as learning questions are correctly answered.

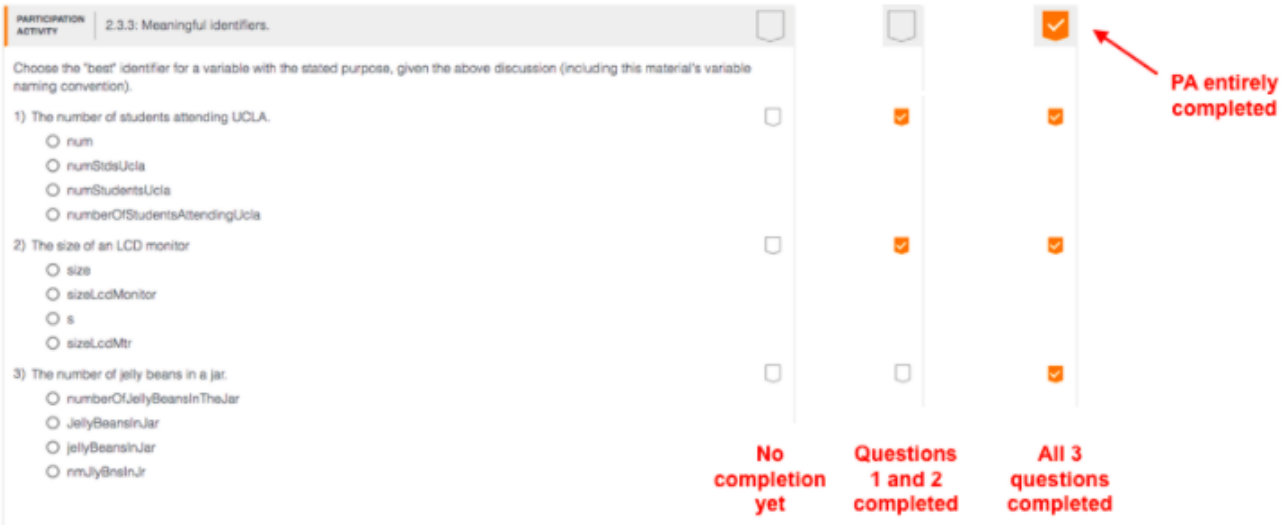
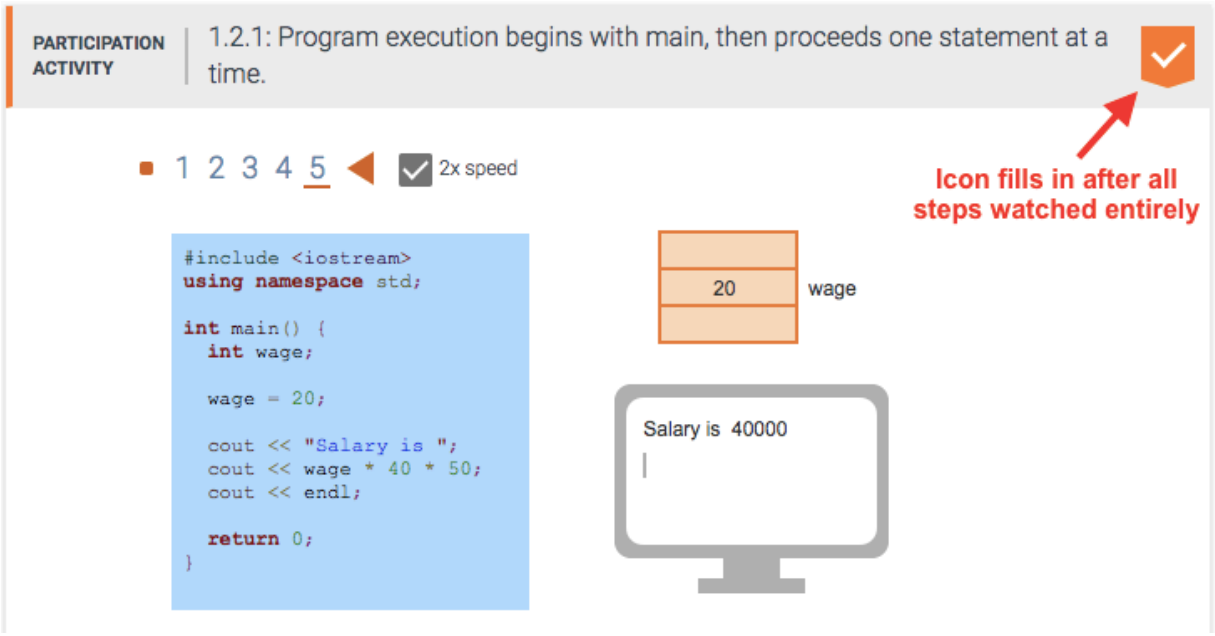


Figure 1.16.2: PA completion icon filling in as an animation's steps are watched entirely.



The 'return 0' statement ends the program.

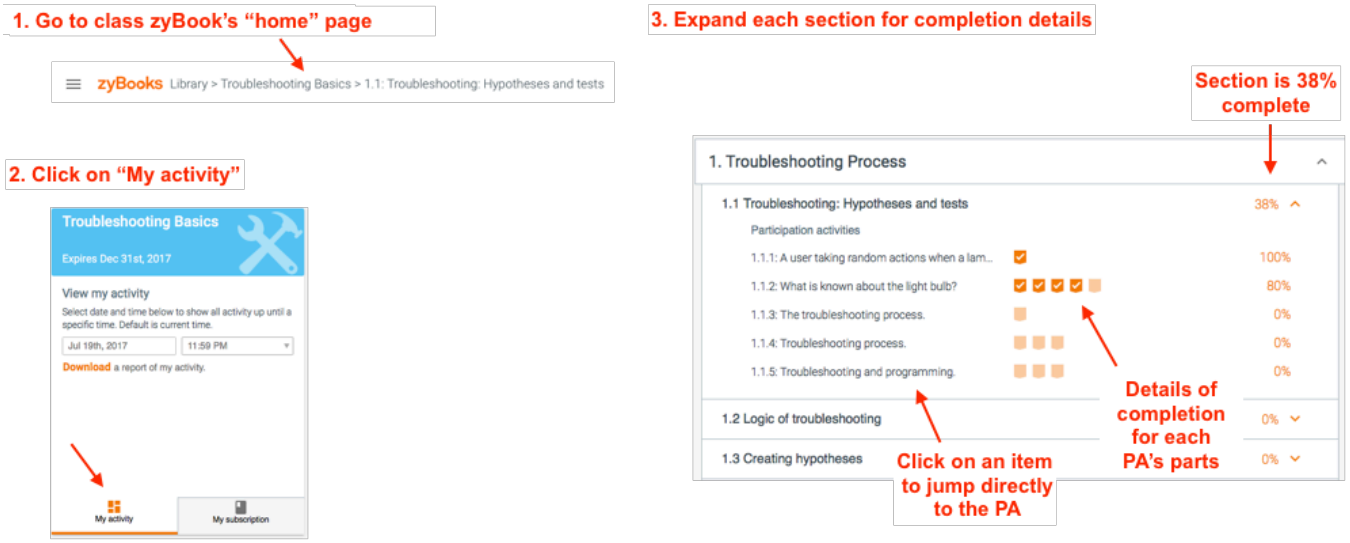
Feedback?

Viewing PA completion for entire sections

Students can also check section-level activity completion status at a zyBook's "Home/TOC", which is accessed from within a zyBook by clicking the zyBook's name at the top.

If you are completing zyBook activities in another browser tab, you may have to reload the Home/TOC page to see updated activity completion.

Figure 1.16.3: Checking section activity completion on a zyBook's Home/TOC page.



PARTICIPATION ACTIVITY 1.16.2: PA completion.

1) A student's section-level activity completion data can be found by _____ .

- clicking on the zyBook's name at the top to go to the zyBook's "Home".
- clicking on the student's name

2) If the activity completion does not appear updated, then _____ .
   - "refreshing" the page may help
   - send email to support@zybooks.com

3) Within a section, a student can quickly detect if an activity is not fully completed by _____ .
   - looking for unfilled icons next to each activity
   - contacting support

Returning to a section: Previous answers intentionally aren't shown

When a student leaves and then returns to a section, the system intentionally does *not show previous answers to learning questions*. That way, students can redo a section to help better learn the material. Research shows that approach is better for learning (one article discussing such research).

The completion icons on the side remain filled in, though, so students can know what PA's have already been completed.

PARTICIPATION ACTIVITY 1.16.3: zyBook question answers.

1) If a student returns to a section and notices all previous answers are missing, then _____ .
   - the zyBook system must be broken
   - waiting long enough should cause the answers to appear

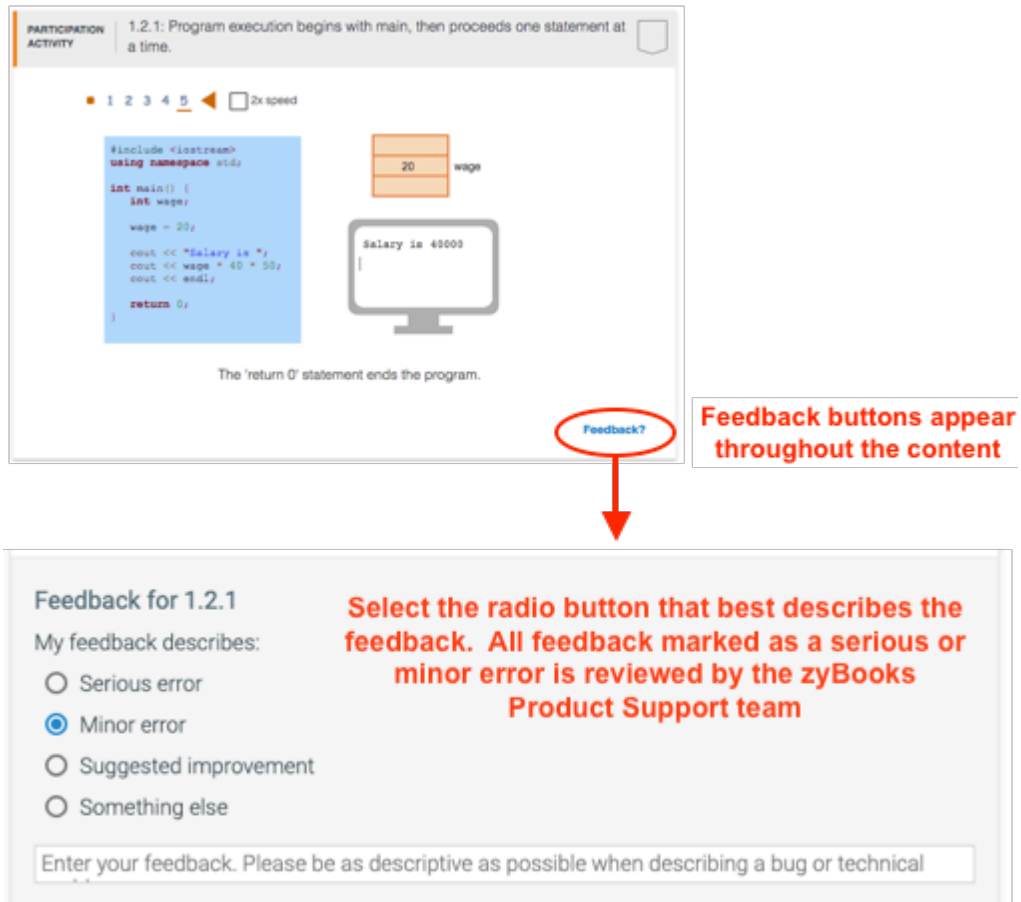
- this is normal

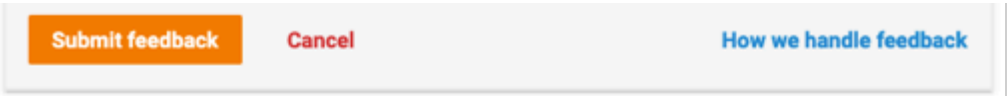
Feedback

Use the "Feedback" links throughout the material to let us know what was helpful and what could be improved. The feedback links appear throughout the material; if one doesn't exist for an item you'd like to report on, just find the closest one and explain what item you're referring to. Our authoring team reviews such feedback a couple times each year, which helps guide updates/improvements to the material.

Errors: While all material goes through a review process, if you find an error, please let us know by selecting "My feedback describes a possible content or technical error" when submitting feedback. Your feedback then goes straight to our support team as a "support ticket". The team typically responds within a day (usually in less than an hour, especially during California working hours), and often fixes issues in the live zyBook right away.

Figure 1.16.4: Feedback buttons appear throughout the material.





PARTICIPATION ACTIVITY 1.16.4: zyBook activity feedback.

1) Our team requests that you make extensive use of the _____ link to tell us what you liked and what could be improved.

Check Show answer

2) Selecting "My feedback describes a possible content or technical _____" within the feedback form causes the support team to be immediately notified.

Check Show answer

3) Describing an actual bug like "A small insect" or even "An error in a program", and then selecting "My comments describe a bug", is a funny joke. (Answer true or false).

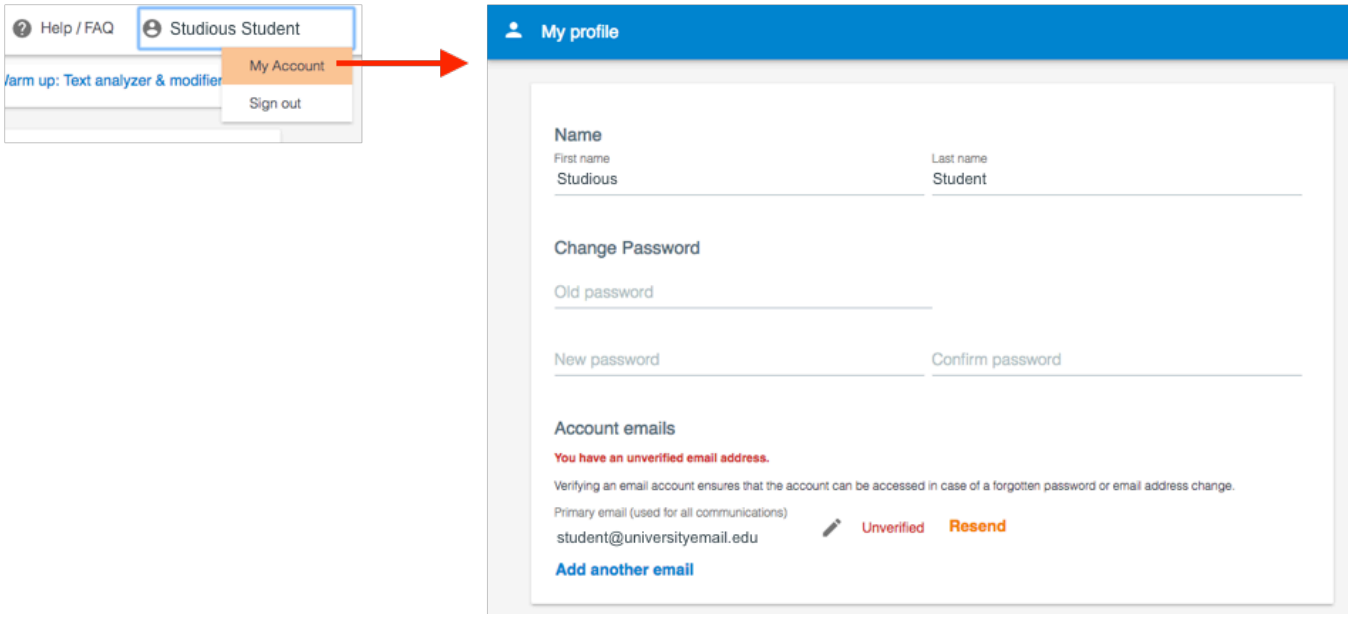
Check Show answer

1.17 Account and platform basics

Managing your profile

Your zyBooks.com profile can be viewed by clicking on the person icon in the upper right.

Figure 1.17.1: Viewing one's zyBooks.com profile.



The profile page allows you to modify your password, email, and other items associated with your zyBooks.com account.

An "unverified email" message appears until you click on the link in the message sent to your email. You can have multiple emails associated with your zyBooks.com account. Verifying your email address ensures you signed up with a correct email address (typos are common, so verifying is wise).

PARTICIPATION ACTIVITY 1.17.1: Profile.

1) Changing your email address requires contacting support.

contacting support.

- True
- False

2) Verifying your email address is mostly a waste of your time.

- True
- False

3) If you don't receive an email verification message in your email inbox, that means the zyBook system is broken.

- Likely
- Unlikely

4) Multiple email addresses can be associated with one account.

- True
- False

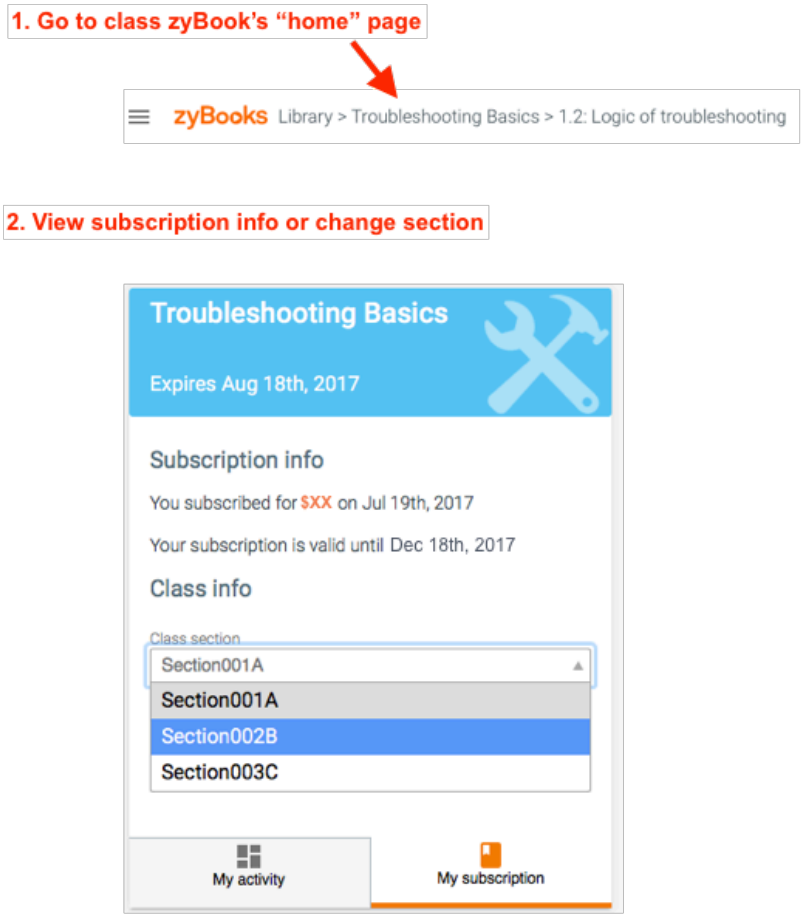
5) If you forgot your password, you can edit your profile to change the password.

- True
- False

Managing your subscription

After subscribing to a class zyBook, each subscription has information distinct from your zyBooks.com profile. You can access that zyBook's subscription information by navigating to your zyBook's "Home/TOC", which is accessed from within a zyBook by clicking the zyBook's name at the top. The "My subscription" tab contains the relevant subscription information, including amount paid, subscription begin/end dates, etc.

Figure 1.17.2: Viewing subscription information.



While the subscription info page has some info, the page is not a receipt. Your receipt was emailed to you when you subscribed. If you need a receipt, check your email (and be sure to check your spam folder).

Some classes have multiple class sections, which you typically select when subscribing, but which you can change in the subscription info page, as shown above.

PARTICIPATION ACTIVITY 1.17.2: Subscription info.

1) To change a class section in the zyBook, a student should contact their instructor.
   - ○ True
   - ○ False

2) To obtain a receipt, a student should

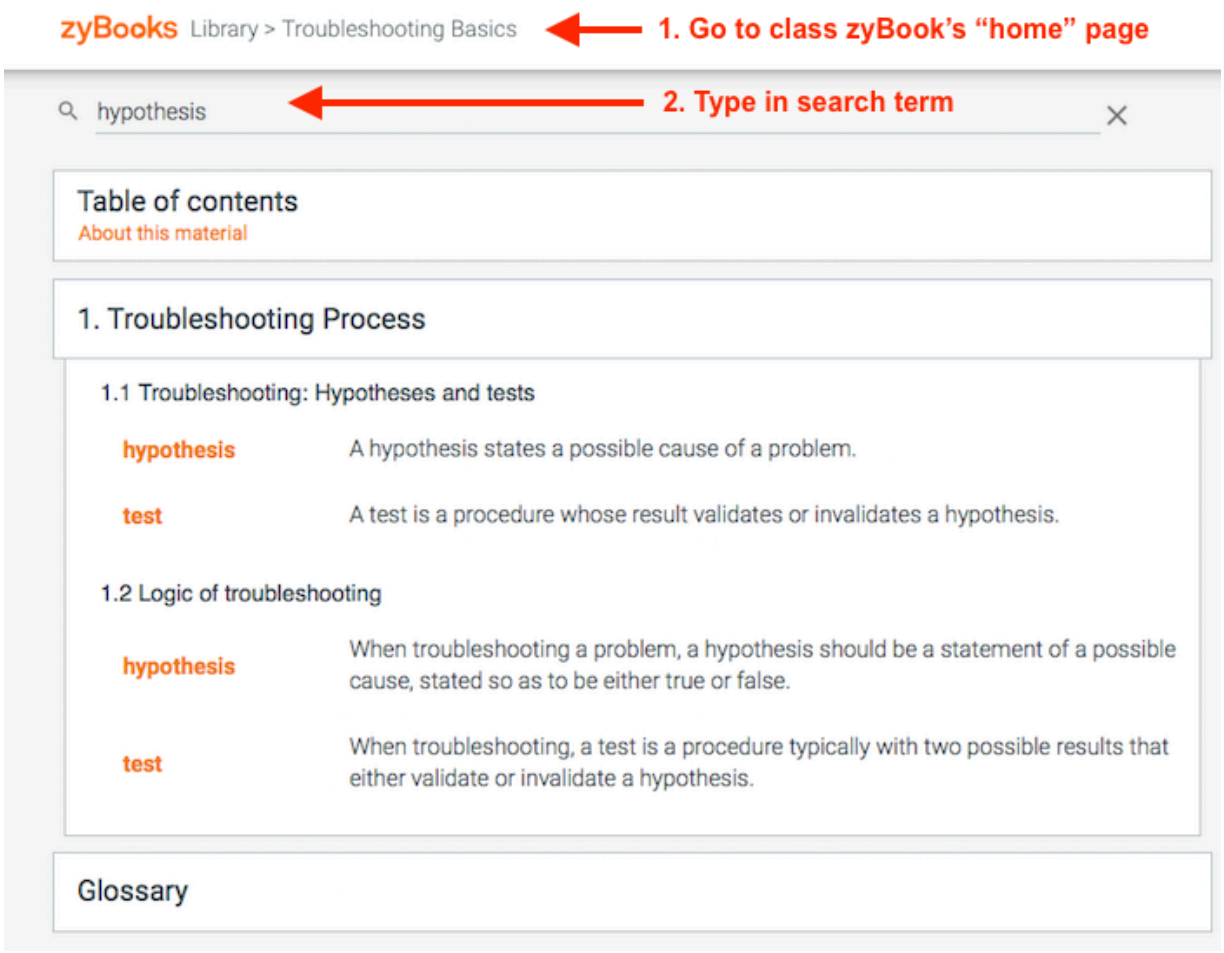
2) To obtain a receipt, a student should just print the subscription info page.

- True
- False

Search

The zyBook has a unique Table of Contents (TOC) that includes defined terms. Your zyBook's "Home/TOC" is accessed by clicking the zyBook's name in the upper left. The combined TOC and terms can be searched using the search field. Clicking on any result jumps you to that portion of the zyBook. Clicking on the "Glossary" link at the bottom also provides a list of defined terms, which can be similarly searched by using the search field.

Figure 1.17.3: Table of contents and index terms.

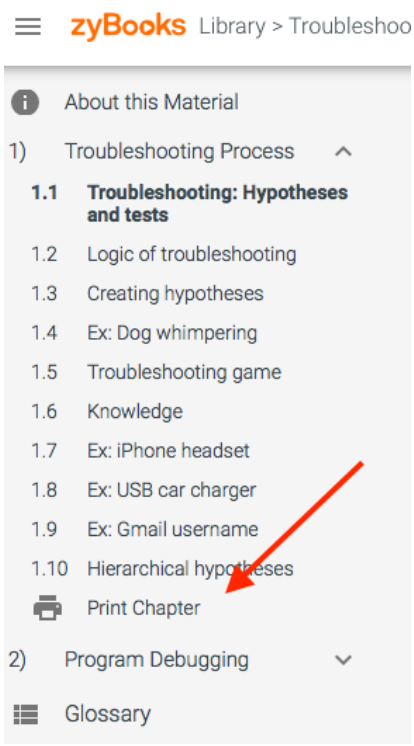


Subscription extensions / Printing to PDF

Subscriptions to class zyBooks typically last for the duration of the class term, plus a week or so. Subscription to catalog zyBooks (not part of a class) last a specified number of months (like 6). At the end of a subscription, low-cost renewals are usually available, typically for a couple dollars a month. An automated option should appear as the subscription period nears the end, or email support@zybooks.com.

Most zyBooks can also be printed to PDF, on a per-chapter basis. In the chapter menu, select the "Print Chapter" link located at the end of the chapter. zyBooks are interactive, so obviously those interactive features won't appear in the static PDF, but rather the text, figures, tables, and similar items will be present. Such a static version enables a form of permanent access to at least the base material. Such a static version is also sometimes printed out for use by instructors preparing for lecture, students studying for exams, or for reading when internet is not available (like in a car or on a plane).

Figure 1.17.4: Printing a chapter (typically to PDF).



System and platform issues

If you have any issues with our system, please email support@zybooks.com. *Please don't bother your instructor;* we want them to focus on teaching. If you are having a technical issue, please tell us what browser (like Chrome, Firefox, Safari, etc.), operating system (Windows 10, MacOS, iOS, etc.), and zyBook you are using (just copy-paste the URL or the zyBook code).

Web browsers aren't perfect. zyBooks make extensive use of HTML5 features, which browsers are supposed to support. But browsers do have bugs. If something doesn't work, before emailing support, please try refreshing or restarting the web browser. You might even try a different web browser, such as Firefox or Chrome.

PARTICIPATION ACTIVITY 1.17.3: System issues.

1) If you have a system issue, send email to ____@zybooks.com.

Check Show answer

2) When emailing support@zybooks.com, good information to include is your browser, your _____ , and your zyBook code.

Check Show answer

3) Refreshing, restarting, or trying a different web _____ fixes many issues.

Check Show answer

1.18 Progression challenge activities

In a zyBook, a *participation activity* (PA) is an activity used in the initial learning of a topic. Anyone can get all PA points just by participating (hence the name). In contrast, a **challenge activity** (CA) is the zyBook version of a homework problem. A CA allows students to apply and practice what they've learned. Students must eventually get a CA right on their own to get points, and thus CAs serve as a lightweight assessment for instructors as well.

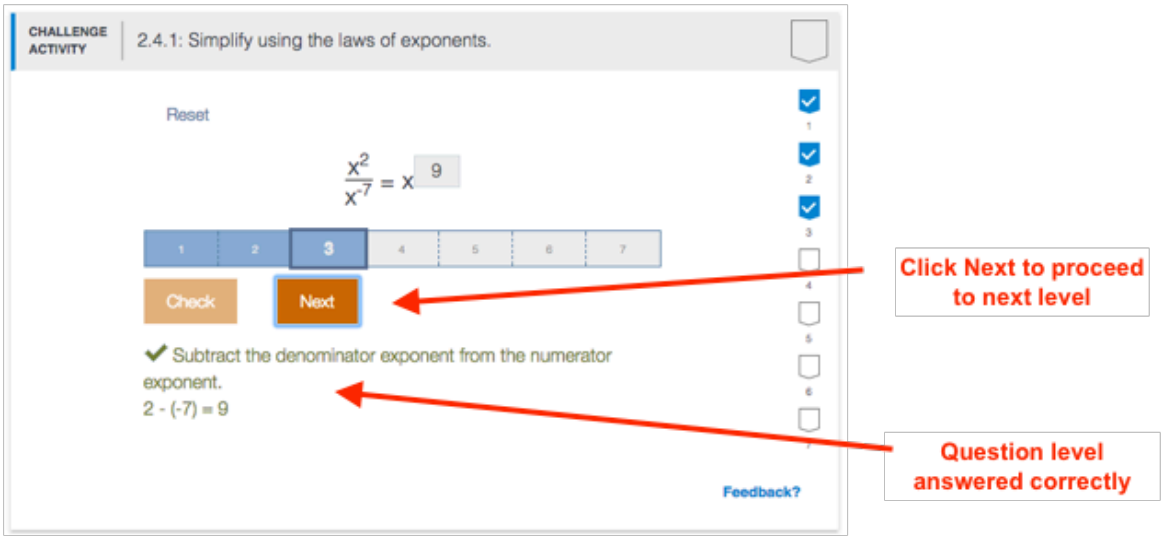
Different types of CAs exist. A particular zyBook may have one type of CA, different types of CAs, or no CAs at all.

Progression challenge activities

Some zyBooks include *progression challenge activities* that consist of a series of auto-generated questions, each progressively more difficult. Students must correctly answer a question at each level before proceeding to the next higher level. If answered correctly, a green checkmark appears, usually along with an explanation. Clicking "Next" proceeds to the next higher level.

Such an incremental approach represents a "structured adaptivity" approach used in zyBooks, teaching specific concepts in an incremental manner to help students progress.

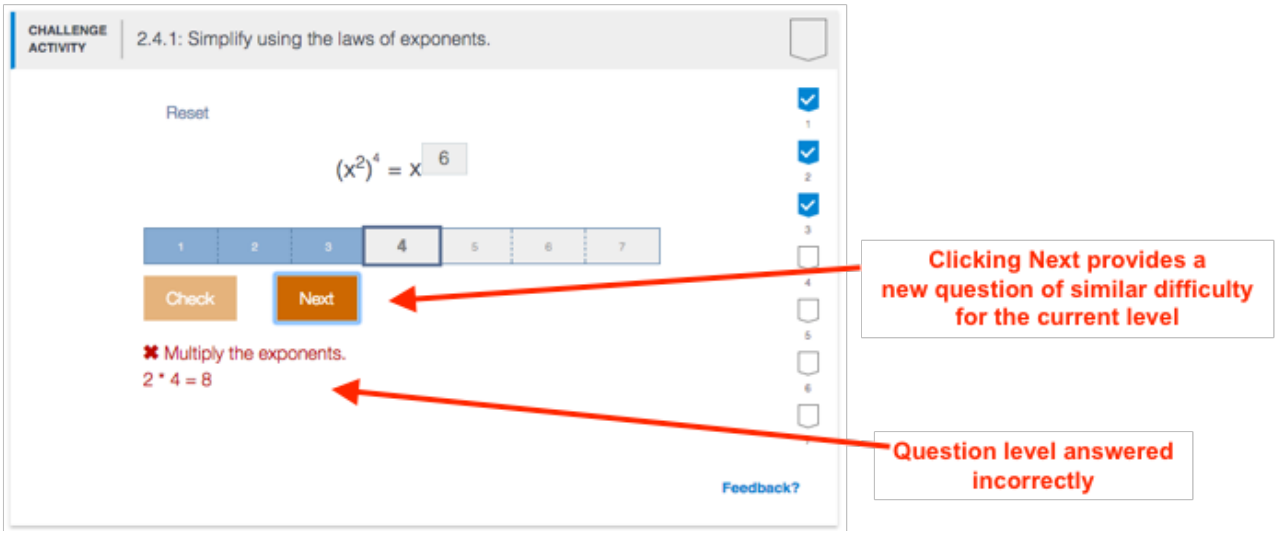
Figure 1.18.1: Progression challenge activities: Proceeding to next level.



If answered incorrectly, the correct answer and possibly an explanation are shown. Clicking "Next"

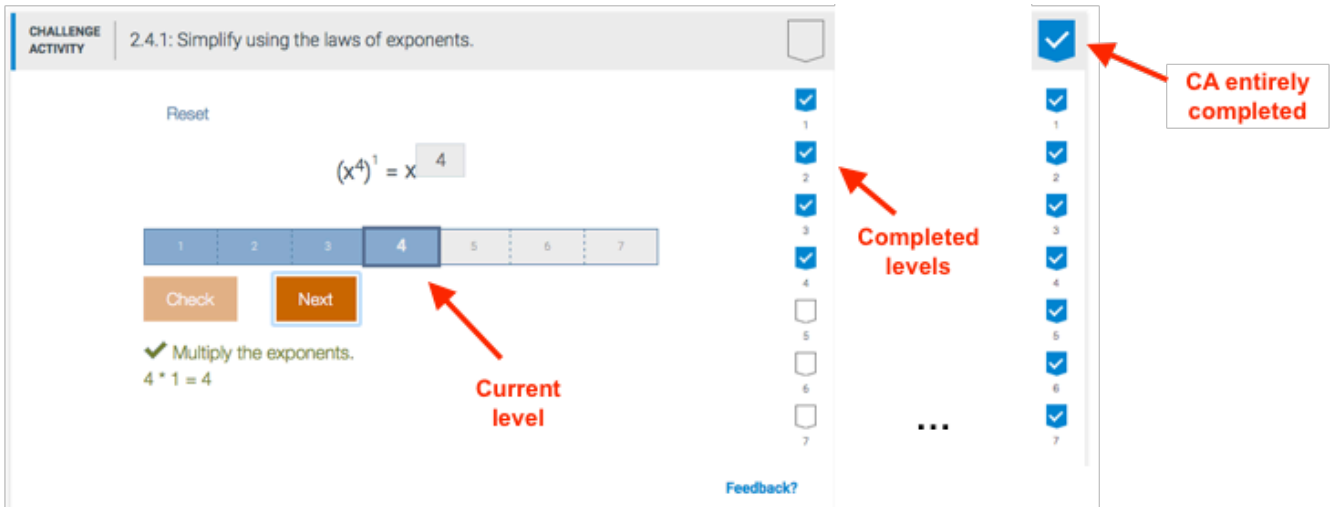
generates a new question of the same difficulty for that level. Students may attempt questions at each level as many times as needed.

Figure 1.18.2: Progression challenge activities: Incorrect answers.



Completion of progression challenge activities can be viewed by filled-in levels, as well as the filled-in icons next to the activity. If a student leaves the page and returns, the completed levels remain filled.

Figure 1.18.3: Activity completed shown by filled-in level and icons next to the activity.



Students may click on any previously-completed level to do more problems for extra practice. Such practice is useful to build confidence, study for an exam, etc. For students, clicking on a level beyond the next uncompleted level yields an error message (instructors may jump to any level, however); each level typically builds on earlier levels, so earlier levels must be completed first.

PARTICIPATION ACTIVITY | 1.18.1: Progression challenge activities.

1) If a user has previously completed questions for level 1, 2, and 3, the user may jump to question level 4.
   - True
   - False

2) A user may only attempt a question at each level once.
   - True
   - False

CHALLENGE ACTIVITY | 1.18.1: Sample progression challenge activity: Simplify using the laws of exponents.

426488.2706032.qx3zqy7

Start

$x^6x^1 = x$ ☐

1	2	3	4	5	6	7

Check Next

1.19 Programming challenge activities

In a zyBook, a *participation activity* (PA) is an activity used in the initial learning of a topic. Anyone can get all PA points just by participating (hence the name). In contrast, a **Challenge activity** (CA) is the zyBook version of a homework problem. A CA allows students to apply and practice what they've learned. Students must eventually get a CA right on their own to get points, and thus CAs serve as a lightweight assessment for instructors as well.

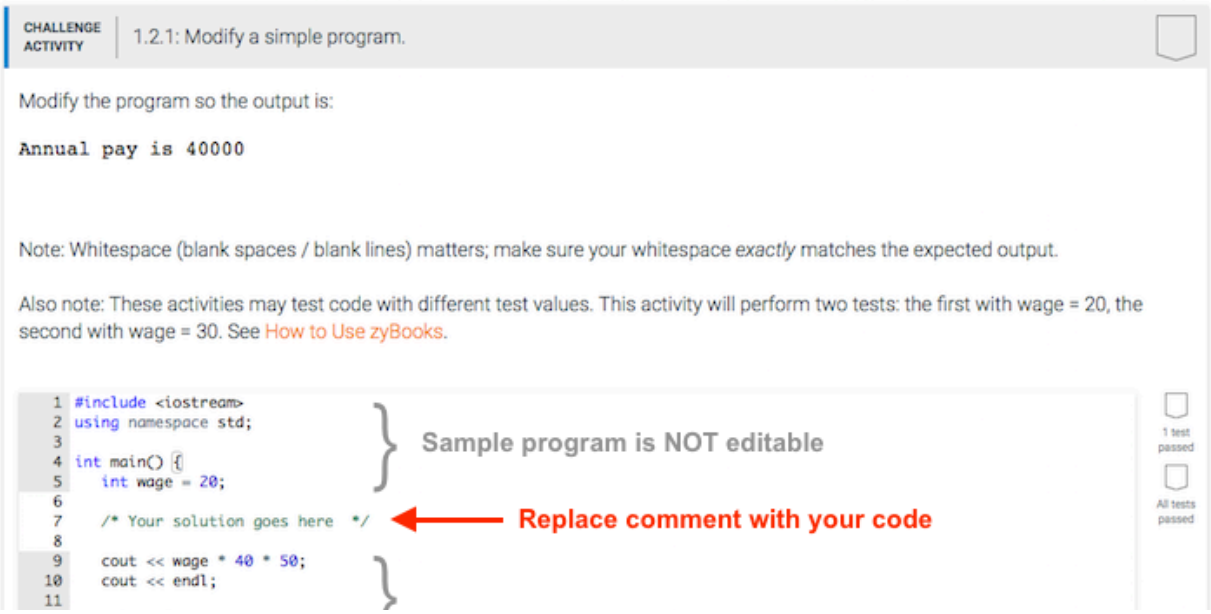
Different types of CAs exist. A particular zyBook may have one type of CA, different types of CAs, or no CAs at all.

Programming challenge activities

Some zyBooks (C, C++, Java, Python, etc.) may include programming "challenge activities" that ask a student to type a code portion of a program into a text box. Such activities are auto-graded, by compiling and running the program with various test cases.

- Replace `/* Your solution goes here */` (or `'''Your solution goes here'''`) with your code.
- The surrounding sample program's code is NOT editable.

Figure 1.19.1: Sample C++ programming challenge homework activity: Replace comment with your code.



```
12   return 0;
13 }
```

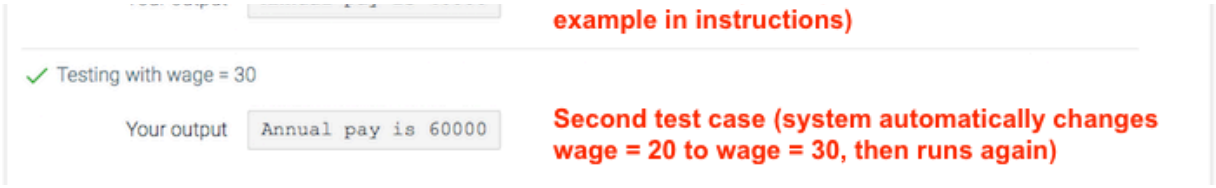
Run

A programming challenge activity appears at this section's end. The activity uses C++, but the lesson applies to C, C++, Java, Python, etc.

After clicking "Run", the program is automatically tested with various *values that may differ from what appears in the sample program*, to ensure the student's code properly solved the problem.

The result of each test case is shown below the "Run" button. If all test cases are passed, a green checkmark appears with the message "All tests passed".

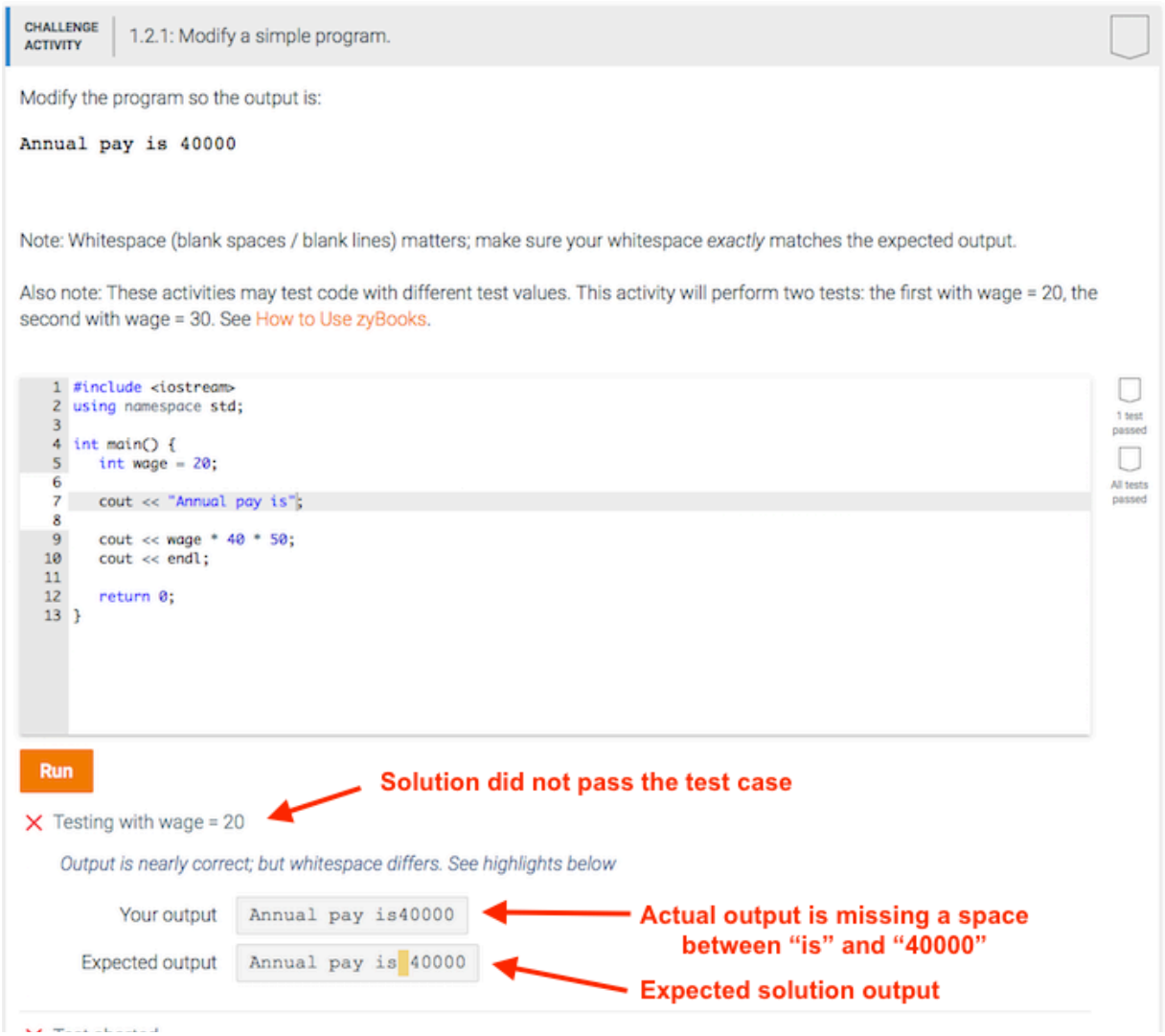
Figure 1.19.2: Sample C++ programming challenge activity: Passed all test cases.



If any of the test cases did not pass, the expected output (or value) is shown along with the actual output produced by your code.

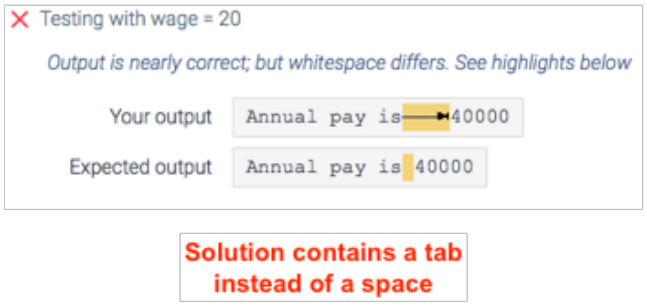
If the program's output differs from the expected output in the whitespace characters, such as tabs and newlines, the differences will be highlighted in yellow.

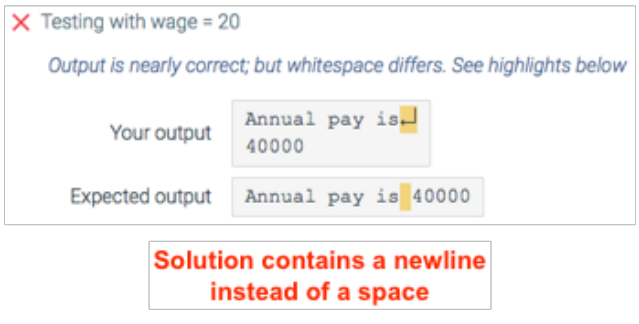
Figure 1.19.3: Sample C++ challenge homework activity: Failed test cases.



Certain whitespace characters, such as a newline or tab, that are in the student's output but are not in the expected output, will be shown using special arrow symbols.

Figure 1.19.4: Newlines and tabs are shown using special arrow symbols.



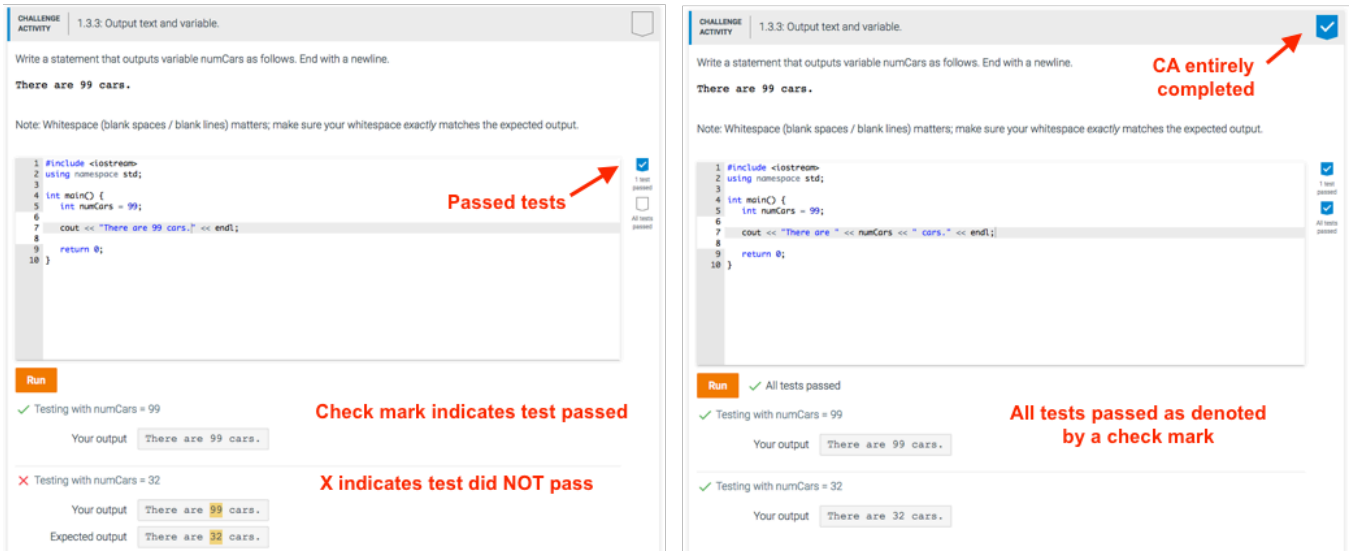


Like other zyBook activities, programming challenge activities are primarily for *learning* and not testing. However, programming challenge activities do not reveal answers to students, to challenge the student. Students (and some instructors) sometimes ask that our system provide solutions to students, but many instructors ask us *not* to provide such solutions, because *much learning occurs in a student's attempts to solve the problem*. Of course, sometimes students just get stuck, in which case students are encouraged to utilize their class' help resources, whatever those may be: discussion board, office hours, classmates (if allowed), email to instructors, etc.

Note that our system may introduce minor version changes to such activities. Version differences are quite hard for a student to notice, but analysis can easily detect that an answer was copied from a different version.

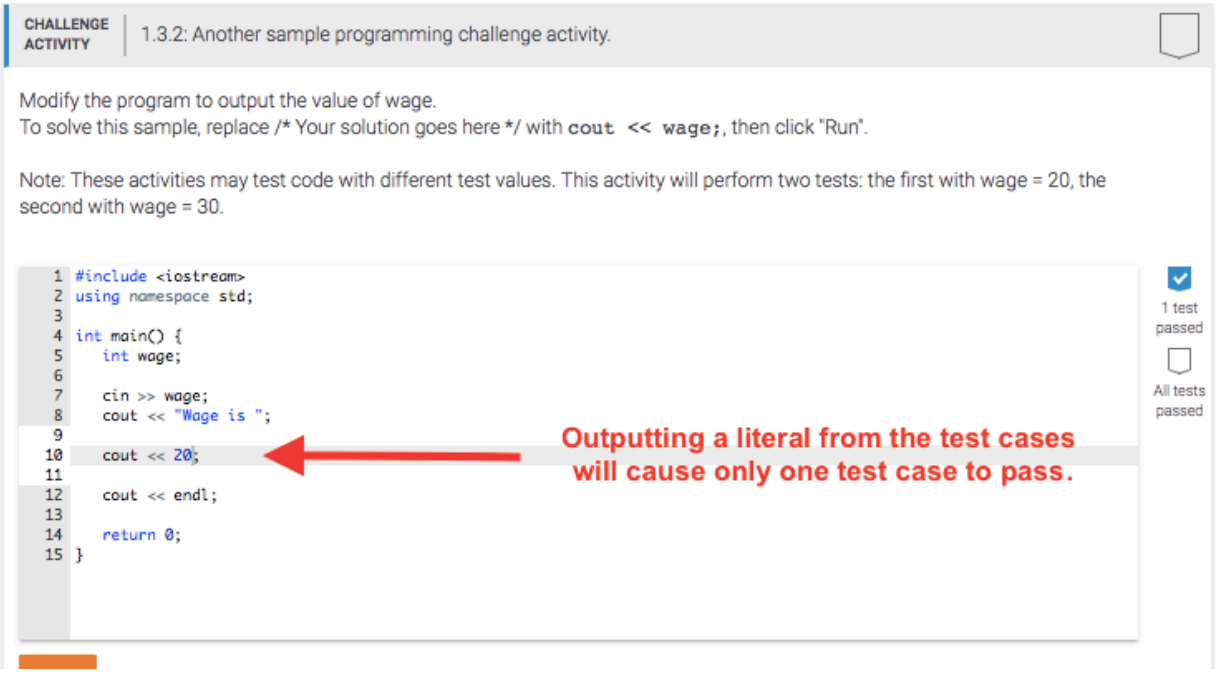
Programming challenge activities are scored as 2 points. (Note: The MATLAB programming challenge activities differ slightly and are explained in the MATLAB zyBook). One point is awarded for passing the first test case. The second point is awarded for passing all the remaining test cases. The number of test cases may differ for each programming challenge activity. Completion of programming challenge activities can be viewed by the filled-in icons next to each activity.

Figure 1.19.5: Programming challenge activity completion.



Beginning students sometimes don't understand that the system may change values to test the student's code. Below, the student was supposed to output the variable wage, but instead output 20. The first test case passes, because wage happened to be 20. But then the system, behind the scenes, *changes wage = 20 to wage = 30* in the program, and then reruns the program a second time. Because the student "hardcoded" 20 rather than outputting wage as instructed, the student's program produces incorrect output.

Figure 1.19.6: Failing test cases due to using a literal.



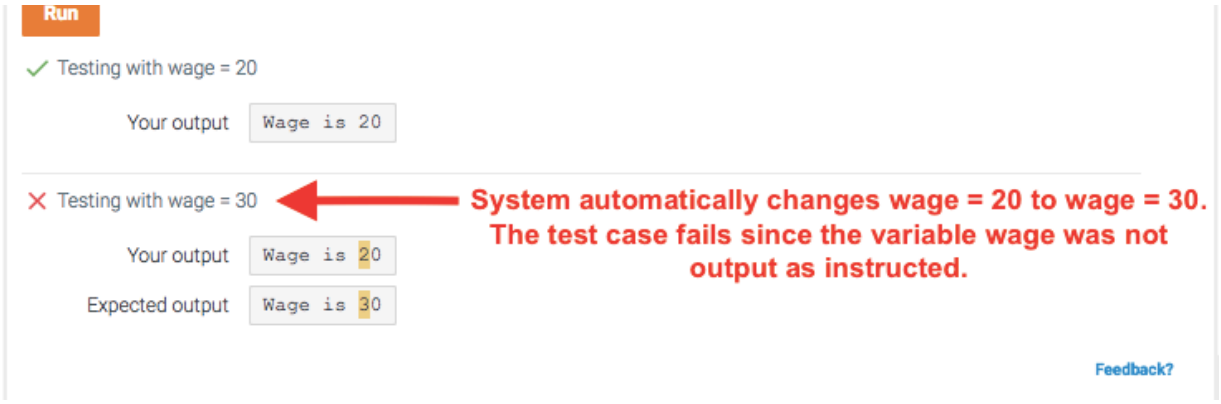
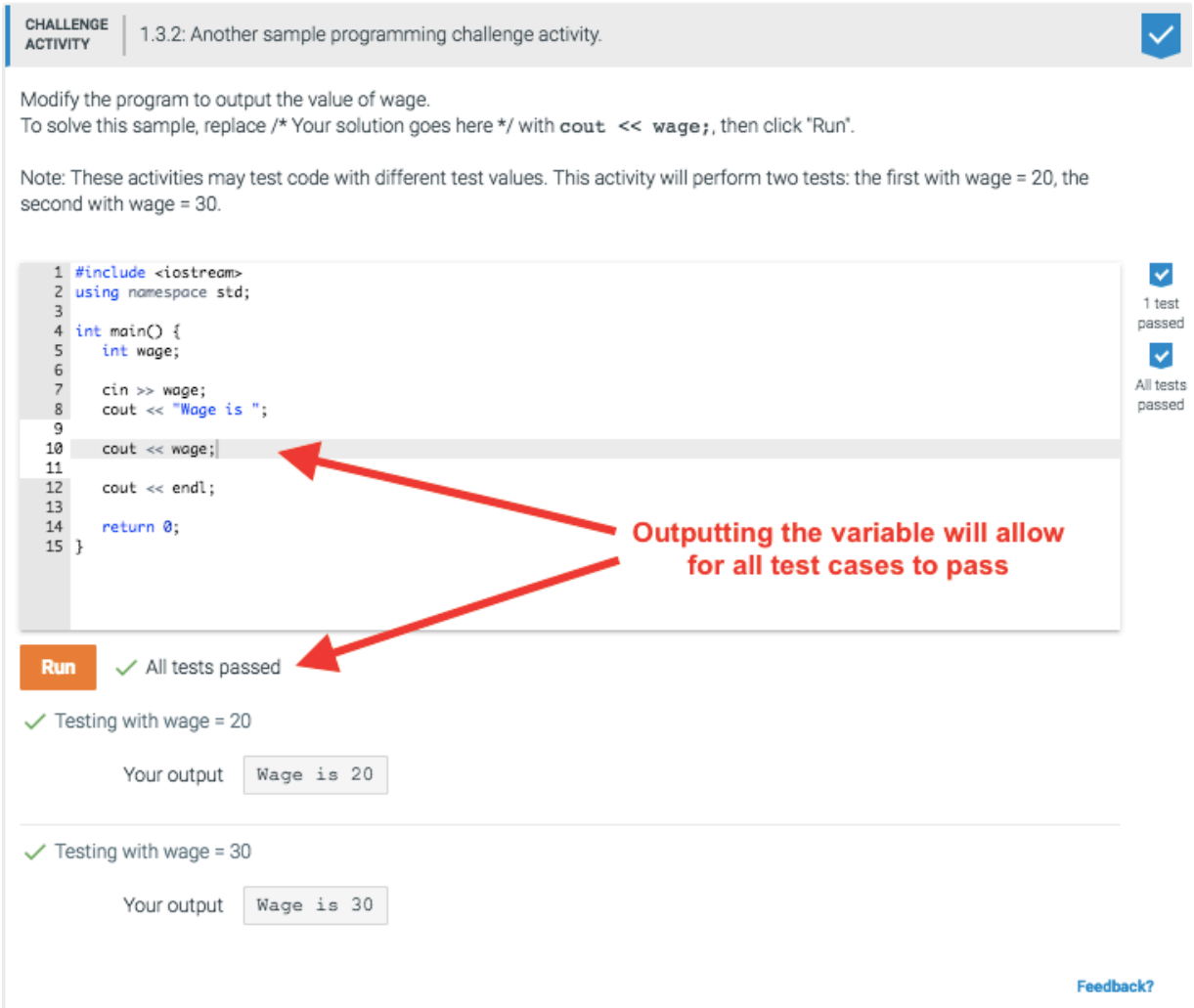


Figure 1.19.7: Passing test cases by correctly outputting a variable.



PARTICIPATION ACTIVITY | 1.19.1: Programming challenge activities.

1) Only type the program portion to replace the /* Your solution goes here */ portion.
   - True
   - False

2) If a user's program passed an activity's sample test case, the program is deemed correct.
   - True
   - False

When running a program, a user may see this error message:

> Program end never reached (commonly due to an infinite loop or infinite recursion).

The user's code may be run with various test cases. If one of those test cases causes the program to exit, the above message is printed. For example, a user may write an infinite loop, such as `while (x = 1) ...` (the user used = instead of ==, a common error in C, C++, and Java). In such a case, the system will stop running the program after a few seconds. Or, the user may access an invalid array/vector index, as in `myArray[i] = 99` , where i is 10 but myArray is size 10 so only has valid indices 0..9 (another common error). In such a case, the program may terminate immediately.

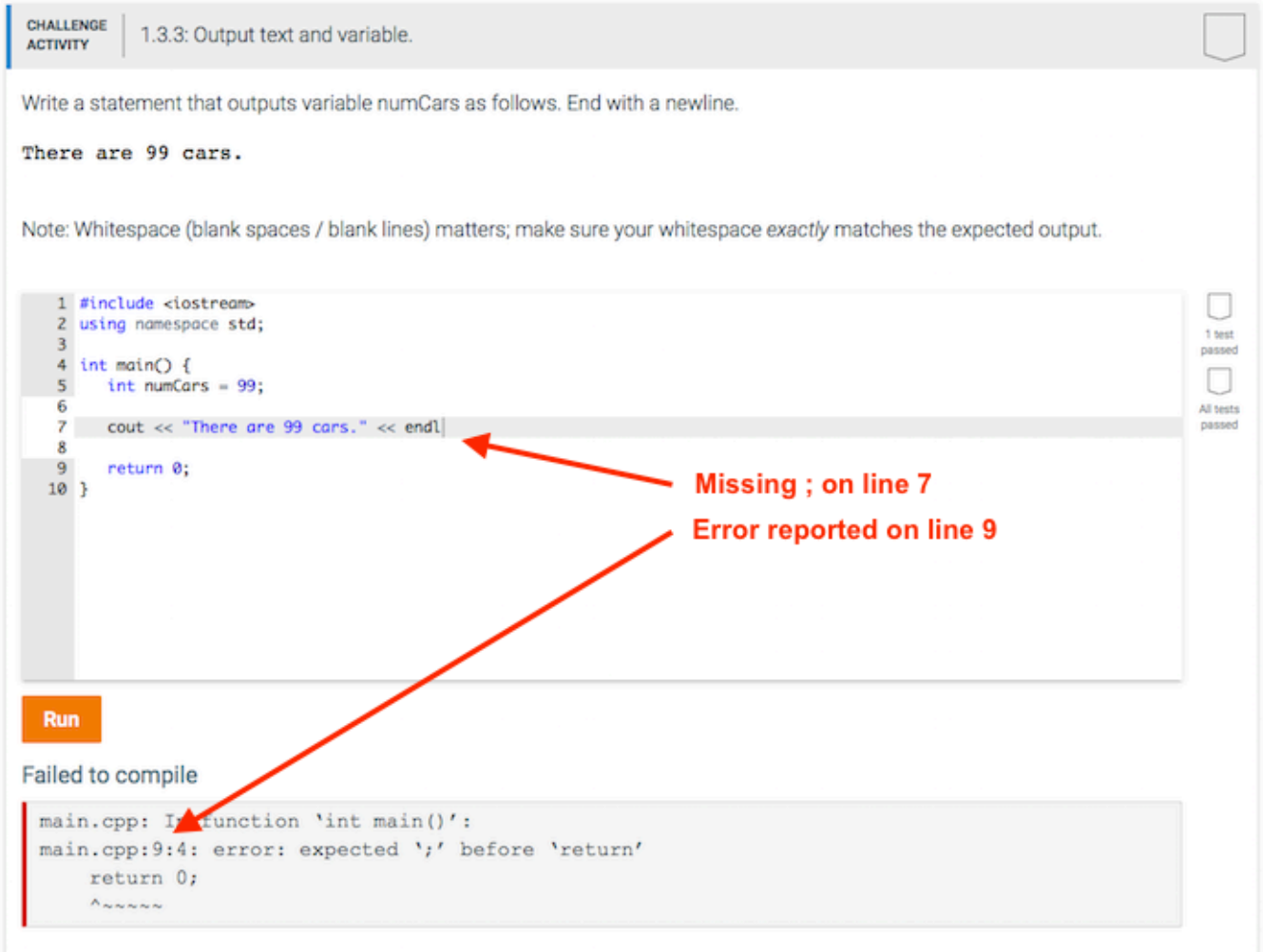
The way the programming challenge activities currently work, the test case that causes the program to fail does not get printed. For some activities where such errors are common, the activity's instructions may indicate the test cases. Whether such test cases are provided or not, the user should carefully examine the code for possible errors.

Students will sometimes submit feedback indicating that the user's program ran fine in a separate environment but fails in the zyBook's environment. "Running fine" depends on what test cases are used. Often, the problem lies in the program not correctly executing for a particular set of test cases.

Students sometimes complain that the activity is reporting a compiler error on a line number the student cannot edit. Students should note that the actual error often exists before the reported line number: the compiler may not notice the error until reaching a later line, such as when a semicolon

number, the compiler may not notice the error until reaching a later line, such as when a semicolon is missing.

Figure 1.19.8: Error appears on line 7, but the compiler reports the error on line 9.



PARTICIPATION ACTIVITY 1.19.2: Program end never reached.

1) The message "Program end never reached" means that the user's program generated output that didn't match the expected output.

   ○ True

- False

2) If a student runs a program successfully in a separate environment, but the program fails in the zyBook, the most common reason is a problem in the zyBook system.
   - True
   - False

3) If the compiler reports an error on a line of code the user cannot edit, the compiler must be broken.
   - True
   - False

CHALLENGE ACTIVITY 1.19.1: Sample programming challenge activity.

Modify the program so the output is:

```
Annual pay is 40000
```

To solve this sample, replace /* Your solution goes here */ with `cout << "Annual pay is ";`, then click "Run".

426488.2706032.qx3zqy7

```
#include <iostream>
using namespace std;

int main() {
   int wage = 20;

   /* Your solution goes here  */

   cout << wage * 40 * 50;
   cout << endl;

   return 0;
```

```
13 }
```

Run

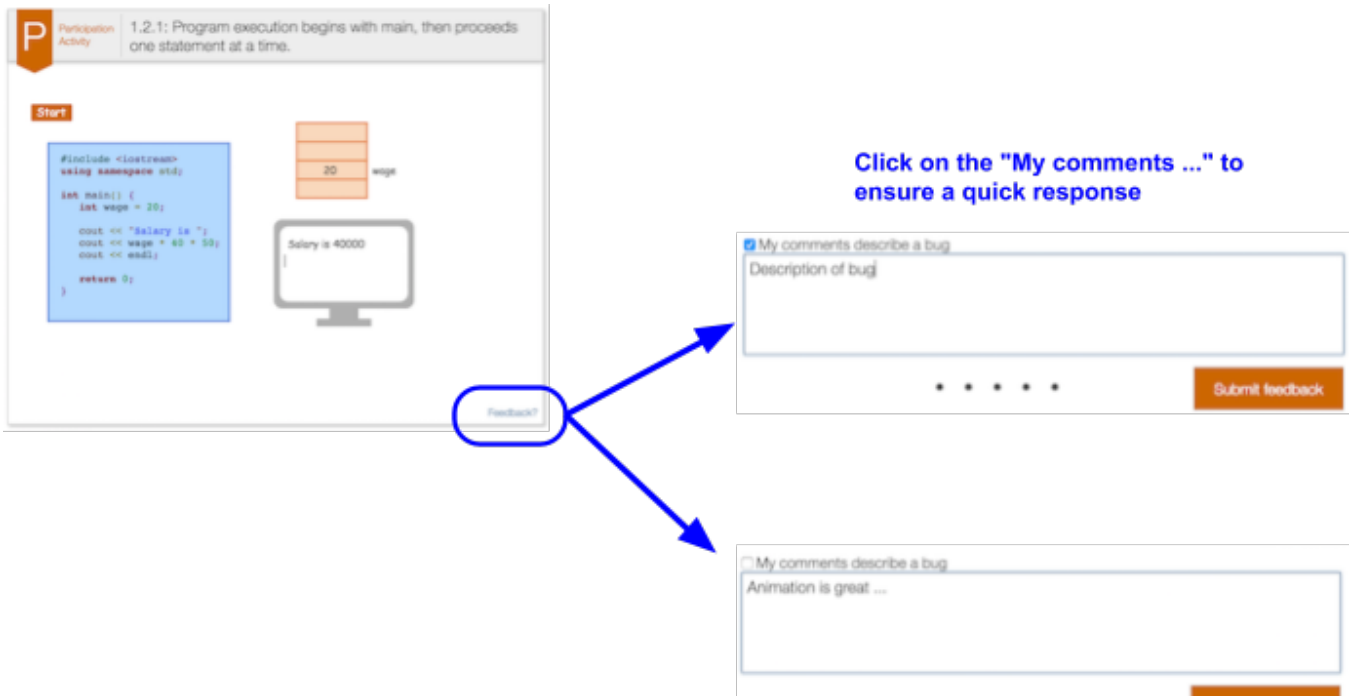
View your last submission

1.20 Feedback

Please help us by using the "Feedback" links to let us know what was helpful and what could be improved. While we carefully check the material, if you find an error, *please* let us know by selecting "My comments describe a bug" when submitting feedback. Your submission goes straight to our support team, who typically responds within a day, and often fixes issues in the zyBook right away.

We pay a lot of attention to feedback, so please help us help you by submitting feedback on how the material can be improved as well as what you liked.

Figure 1.20.1: Each activity includes a feedback button to report bugs and provide comments.



Provide star rating to activities to help us improve content

PARTICIPATION ACTIVITY 1.20.1: zyBook activity feedback.

1) Our team requests that you make extensive use of the _____ link to tell us what you liked and what could be improved.

Check Show answer

2) Selecting "My comments describe a _____" within the feedback form causes the support team to be immediately notified.

Check Show answer

3) Describing an actual bug like "A small insect" or even "An error in a program", and then selecting "My comments describe a bug", is a funny joke. (Answer true or false).

Check Show answer

1.21 zyLab basics

Intro to program auto-graders

In the "old days", students self-tested their weekly programming assignment, then submitted it for grading. Instructors, TAs, or readers would grade the programs, running each program on the instructor's test cases. Perhaps a week later (or more), students would receive a grade for that program, sometimes (but not always) with feedback explaining point deductions and suggested improvements.

Unfortunately, getting grades/feedback a week later was not ideal for learning, as students had moved on to the next assignment and often ignored feedback (or such feedback was received too late, after the next assignment was already submitted). Plus, students were often frustrated due to unexpectedly low scores. Furthermore, much of a teacher's time was spent with such grading, meaning less time to interact with students.

In fact, that situation is not really the "old days" -- it is probably still the most common situation in programming classes today.

Instead, your instructor has chosen to use a state-of-the-art program auto-grader, called zyLabs. An auto-grader automatically runs a student's program on an instructor's test cases, and immediately provides a score to the student. In response, a student can immediately determine and fix mistakes, and resubmit. The learning is better because feedback is immediate (true, human feedback may be better, but not when received a week or more later), and scores are no longer an upsetting surprise.

Any auto-grader has potential issues as well. This section aims to help students become comfortable with the zyLab auto-grader, so students can enjoy auto-grading's benefits, with minimal issues.

zyLab assignments are created and/or controlled by instructors, not by zyBooks. Students having issues should usually first consult with their class' help resources. (Of course, if the zyLabs platform seems to be having issues (rare), please contact zyBooks support by clicking on the "Feedback" link at the bottom of every zyLab assignment).

One last note: Learning to program is hard. It is perfectly normal to struggle with converting an instructor's specification into an actual program, to experience bugs, etc. Even experienced programmers struggle. But the reward of a working program can be quite great. The zyLabs auto-grader is often the bearer of the bad news that "Your program isn't correct", but we hope you find that getting that news immediately so you can fix your program is better than getting that news

that getting that news immediately so you can fix your program is better than getting that news two weeks later.

PARTICIPATION ACTIVITY 1.21.1: Auto-graders.

1) An auto-grader runs a program on an instructor's test cases, thus providing immediate feedback as to the correctness of a program.
   - True
   - False

2) zyLab assignments are created and controlled by zyBooks staff.
   - True
   - False

3) zyLab test cases are created and controlled by zyBooks staff.
   - True
   - False

4) Clicking on the "Feedback" link at the bottom of a zyLab assignment sends an email to your instructor.
   - True
   - False

5) A student should expect to fail plenty of test cases as they learn to program.
   - True
   - False

zyLabs with programming window: Develop mode

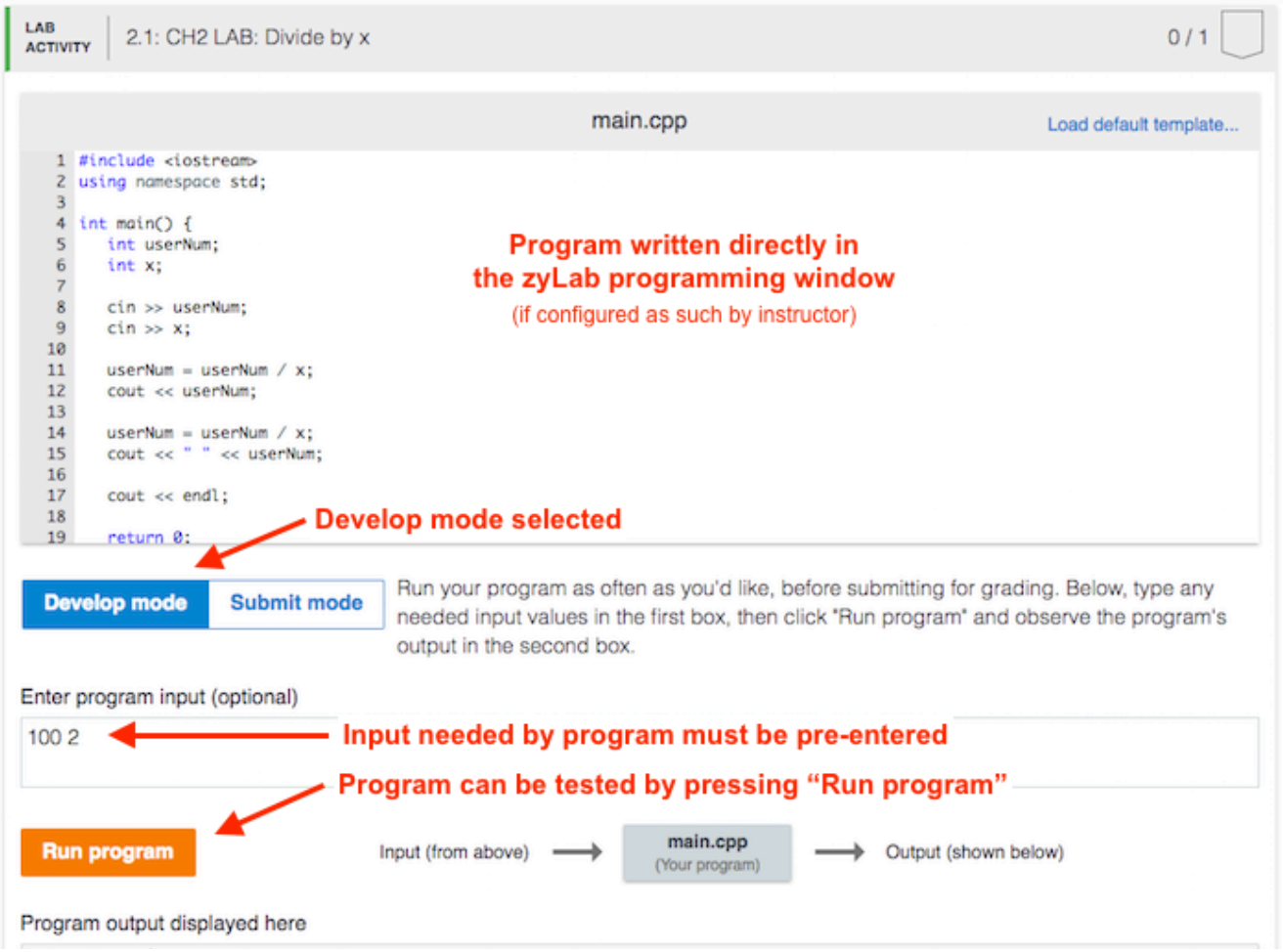
To ease a student's programming learning curve, an instructor can configure a zyLab assignment such that students program directly in a programming window in the zyLab assignment itself. As such, students need not use an external development tool (known as an IDE). Such simplicity can be especially helpful in the first weeks of a class; in fact, some intro classes use the zyLab programming window the entire term, teaching an IDE in a second class.

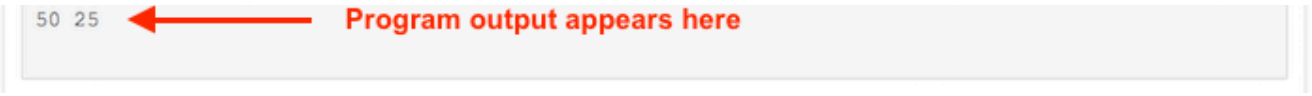
If configured as such, a student writes a program directly in the assignment's programming window. The student can test their program by pressing the "Develop mode" tab and then clicking "Run program". If the program requires input values, the student can pre-enter those values in the input box. The program's output (or error messages) appear in the output box.

A student can run their program as many times as desired while in develop mode, testing the program on different input values.

The student MUST pre-enter any required input. Otherwise, the program will run but may produce no output, or produce erroneous output.

Figure 1.21.1: zyLab with programming window.





PARTICIPATION ACTIVITY 1.21.2: zyLabs with programming window: Develop mode.

1) Any zyLab assignment can be programmed directly in a zyLab programming window.
   - True
   - False

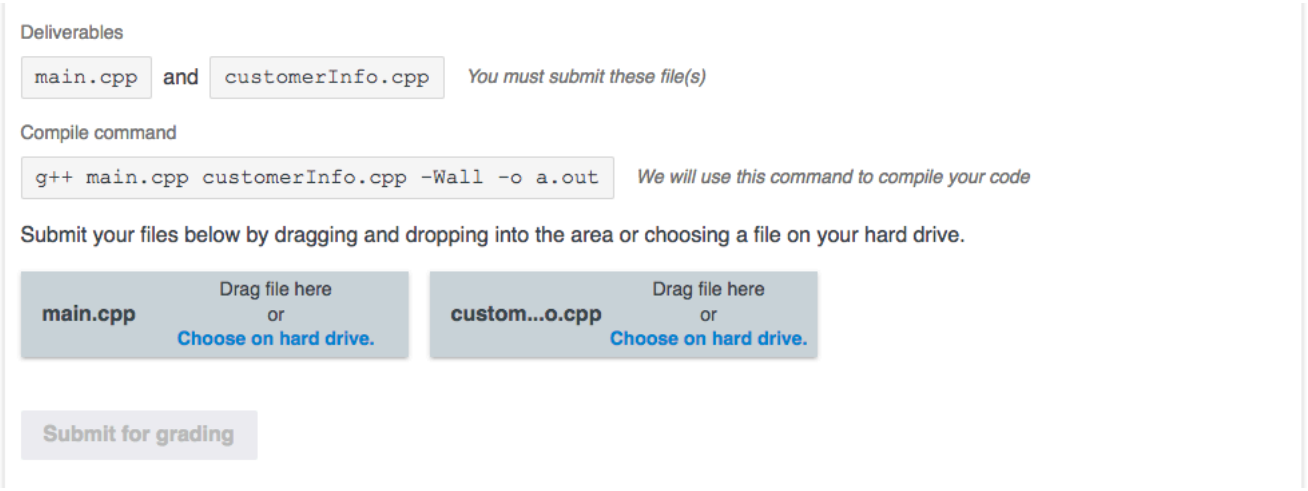
2) An instructor can limit the number of times a student can run a program in develop mode.
   - True
   - False

3) If the student does not pre-enter input values but a program expects input, strange behavior may occur.
   - True
   - False

zyLabs with program file upload

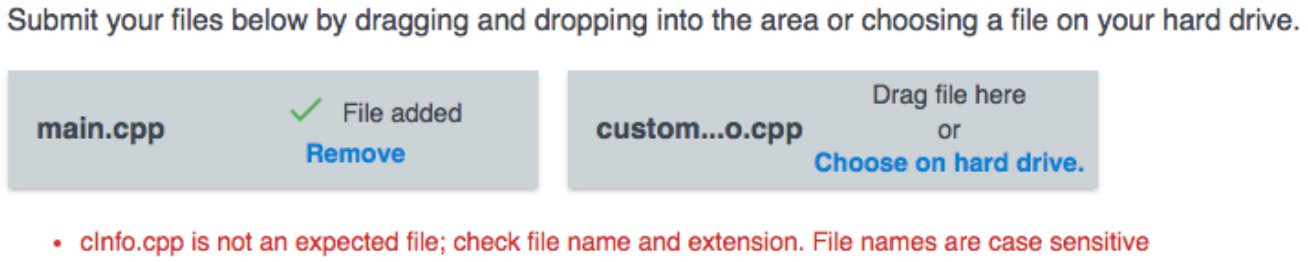
An instructor can configure an assignment to require students to upload one or more program files. The student can drag the file onto the corresponding box, or click to find the file on the student's drive.

Figure 1.21.2: Multiple file submission.



A file's name and extension must match exactly, or the zyLab platform will not recognize the file. Common errors include misspelled file names, use of uppercase instead of lowercase letters or vice-versa, and wrong file extensions (such as .doc rather than .cpp).

Figure 1.21.3: Filenames must match exactly.



PARTICIPATION ACTIVITY 1.21.3: Uploading program files.

1) When a zyLab assignment is configured by an instructor for program files to be uploaded, the student develops their program in another environment.
   - ○ True
   - ○ False

2) The uploaded filename must match the expected filename exactly.

- True
- False

3) In the figure above, the second file has been successfully uploaded.

- True
- False

4) Above, uploading CustomerInfo.cpp would result in a successful upload.

- True
- False

Submitting a program for grading

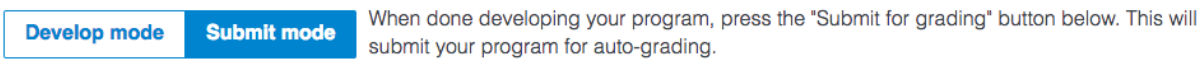
If an assignment is configured with a programming window, the student can select "Submit mode" to get ready to submit their program for grading. That mode does NOT actually submit yet. Rather, a "Submit for grading" button appears.

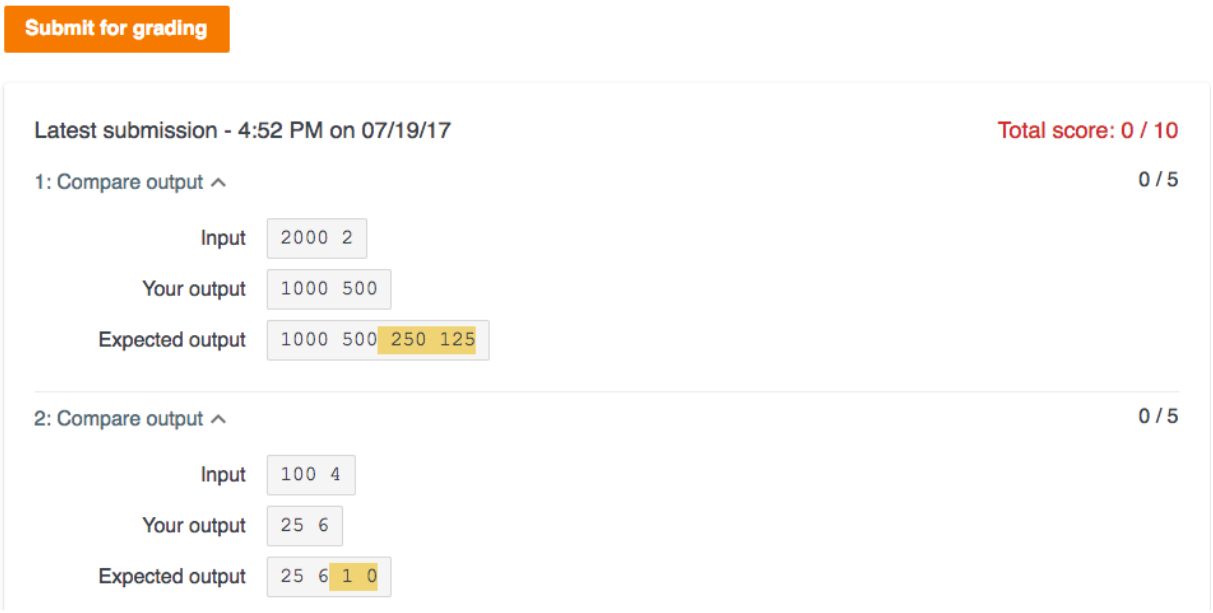
If an assignment is configured to upload program files, once all files have been uploaded correctly, the "Submit for grading" button highlights.

In either case, pressing the "Submit for grading" button compiles and runs the program on the instructor's test cases. The test cases are listed, with an indication of which cases passed, and points are awarded for each passed test case (as determined by the instructor).

When an instructor runs a grade report, the student's highest grade in the specified time window is reported. Thus, if a student submitted five times with scores of 3, 5, 9, 7, 7, the student's score is 9. Thus, students need not worry about receiving a lower score by trying to improve their program; their highest score in the time window is reported to the instructor.

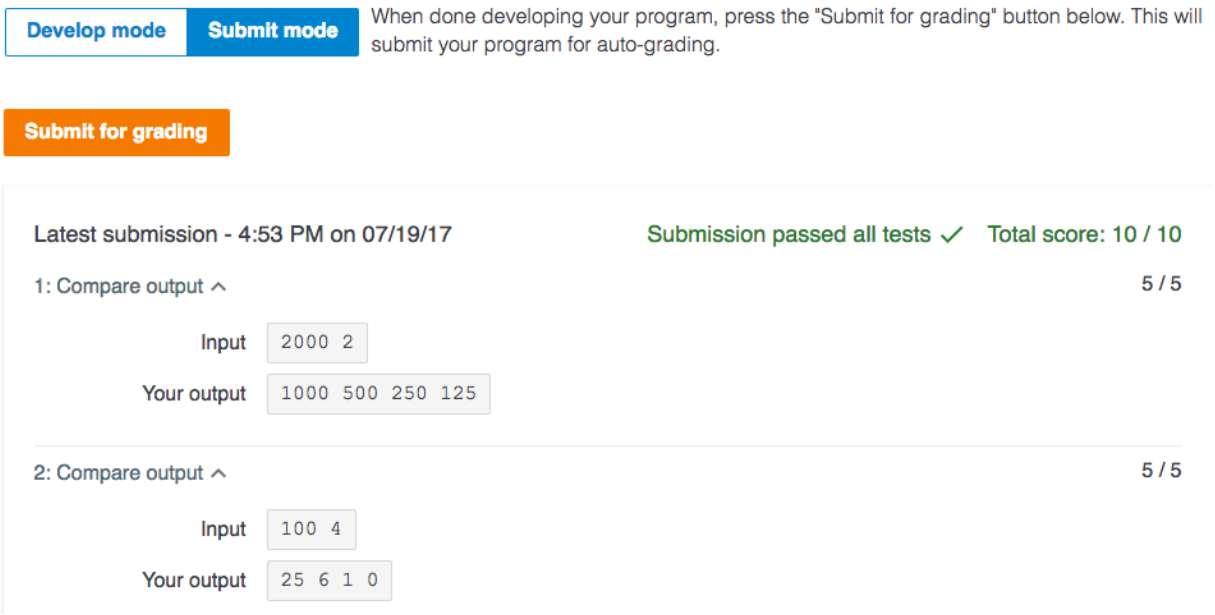
Figure 1.21.4: Submitted and graded lab: No test cases passed.





Noting the failed test cases can help the student understand how to correct their program. Above, the student realizes more numbers must be output, and proceeds to fix the program.

Figure 1.21.5: Submitted and graded lab: All test cases passed.



Instructors can configure a lab to allow a maximum number of submissions, or to accept the next submission only after some number of minutes, to help students think carefully about their

program before submitting (thus not turning programming into a guessing game -- "Let's see if making this change gets me more points.").

Instructors can choose to hide the details of a test case from students, so that students think carefully about why a program might not be correct (thus not over-relying on the auto-grader to do all the thinking for them).

PARTICIPATION ACTIVITY 1.21.4: Submitting for grading.

1) Upon submitting a program for grading, the student's program is run using the instructor's test cases.
   - True
   - False

2) Each passed test case earns some points.
   - True
   - False

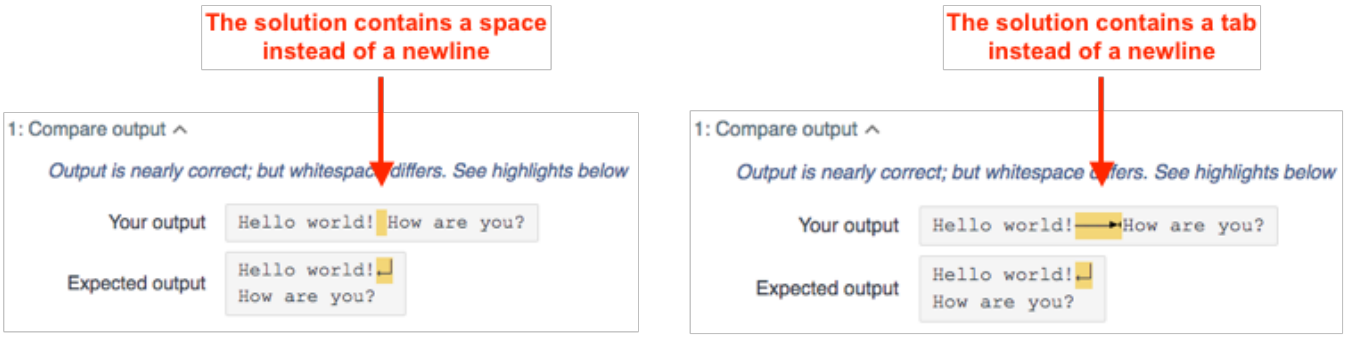
3) An instructor can limit the number of submissions, or the rate of submissions.
   - True
   - False

4) An instructor can hide details of a test case from students.
   - True
   - False

Compare output test

A "Compare output" test compares the student program's output vs. the expected output as specified by an instructor. If differences cause the test case to not be passed, differences are highlighted. Tab and newline characters (normally invisible) are shown using arrow symbols.

Figure 1.21.6: Whitespace characters are shown as control characters.



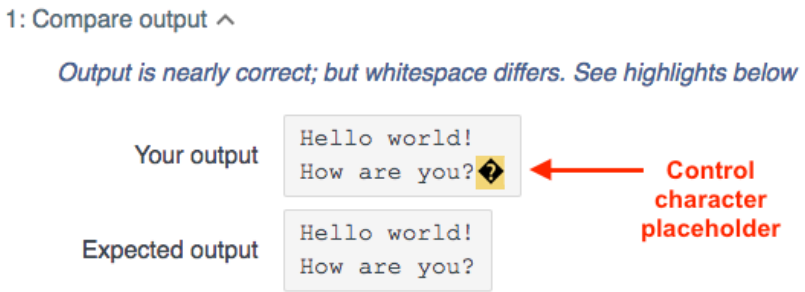
An instructor can configure any compare output test case to ignore whitespace. Some instructors believe precise output formatting is important (teaching precision), others don't wish to have students worrying about whitespace details.

Numerous other configuration options exist, such as only requiring the start of the output to match, or only the end of the output to match.

No perfect algorithm exists for determining and showing differences between student output and expected output. For certain situations, the highlighting may be somewhat confusing. But hopefully the immediate feedback still helps the student determine what is wrong.

If a program outputs an invalid character (typically due to a nasty bug), a "control character" is shown, such as a diamond symbol with a question mark (but other control characters may appear).

Figure 1.21.7: Control character placeholder for invalid characters.



PARTICIPATION ACTIVITY 1.21.5: Compare output tests.

1) If a test case requires output to exactly match "Hello there", then the output "Hello there!" will match.

   - True
   - False

2) If a student repeatedly fails test cases because of whitespace issues even though the program's logic is correct, the student can reasonably conclude that the zyBooks platform is too picky.

   - True
   - False

3) If a program runs fine on an IDE but fails when submitted for grading in zyLabs, the zyLabs platform probably has a bug.

   - True
   - False

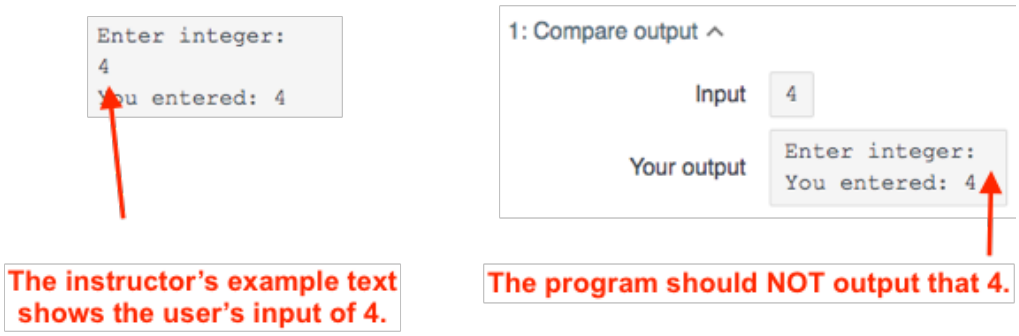
4) If a student's program outputs a diamond with a question mark inside, but the student didn't write such an output statement, then the zyLabs platform probably has a bug.

   - True
   - False

Students sometimes get confused when a program's example text has interleaved output and input. Students sometimes try to output the input values, to match the example text. Students should note that ONLY the output values should be output by the student's program.

Instructors may wish to limit the amount of "prompting" to reduce this common source of confusion, which exists for any auto-grading system.

Figure 1.21.8: Student programs should only output the output, and not also output the input to match example text.



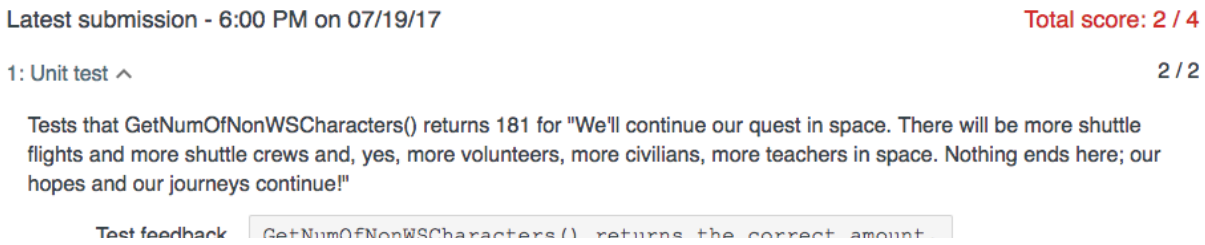
PARTICIPATION ACTIVITY 1.21.6: Compare output tests and interleaved input / output.

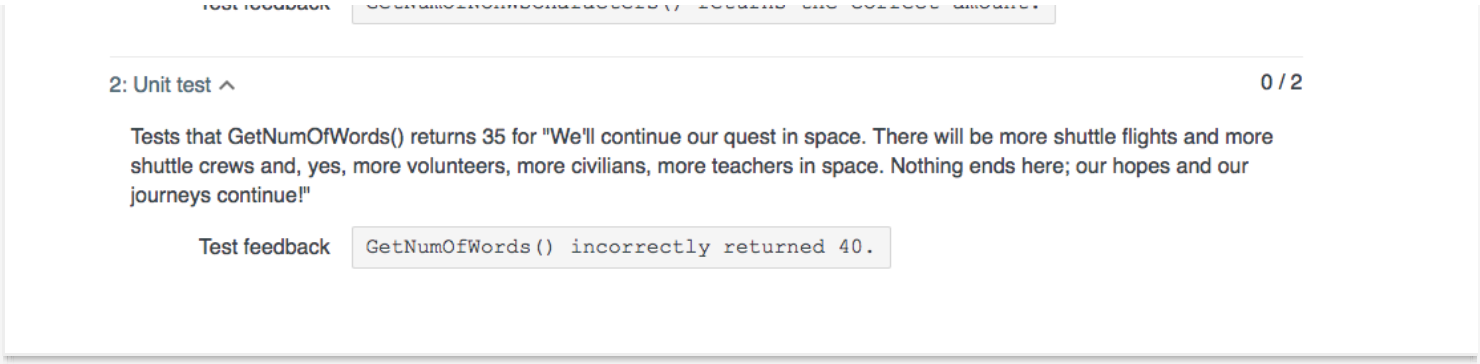
1) A student's program should output all the text shown in an instructor's example of the program running.
   - True
   - False

Unit test

For programs with functions (called "methods" in Java), a *unit test* may be used to verify that the function returns the correct value. A unit test calls a function, independent of the main code, and compares the function's return value against the expected value.

Figure 1.21.9: Unit tests: One passed and one failed unit test.





Miscellaneous

zyLabs doesn't close submissions based on time or date, instead allowing students to continue to practice and improve their code. However, an instructor is able to view submissions recorded before a specified due date and time. Therefore, later attempts may not be counted if beyond the specified due date.

You can view up to 5 of your previous lab submissions. Each previous submission details the number of points received and which tests passed and failed. Previously submitted code can also be downloaded.

Figure 1.21.10: Previous submissions.