2025-2026 / FIRST STAGE

COMPUTER FUNDAMENTALS

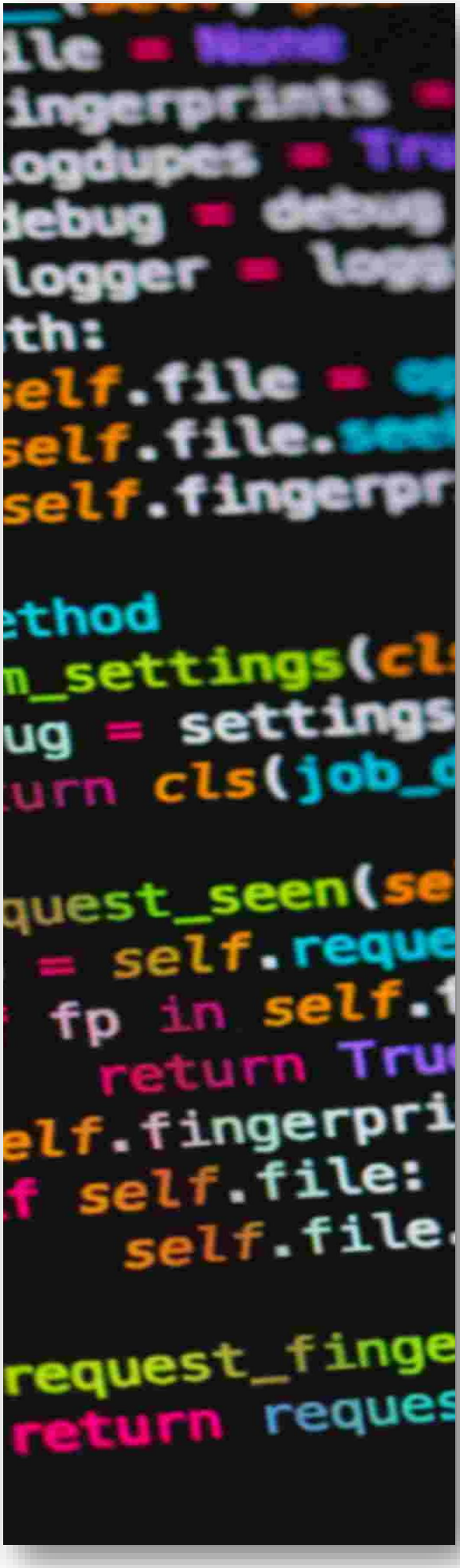
كلية الذكاء الاصطناعي
COLLEGE OF ARTIFICIAL INTELLIGENCE

UNIVERSITY OF BAGHDAD

TABLE OF CONTENTS

1. **WELCOME TO THE COURSE**
2. **SCRATCH**
   - 2.1 COMPUTER SCIENCE AND PROBLEM SOLVING
   - 2.2 ASCII
   - 2.3 UNICODE
   - 2.4 RGB
   - 2.5 ALGORITHMS
   - 2.6 PSEUDOCODE
   - 2.7 SCRATCH
     - 2.7.1 HELLO WORLD
     - 2.7.2 ABSTRACTION
     - 2.7.3 CONDITIONALS
   - 2.8 TUTORIAL
   - 2.9 Q&A
   - 2.10 LAB
3. **C**
   - 3.1 INTRODUCTION
   - 3.2 VISUAL STUDIO CODE
   - 3.3 FORM SCRATCH TO C
   - 3.4 HEADER FILES
   - 3.5 TYPES
   - 3.6 CONDITIONALS
   - 3.7 OPERATORS
   - 3.8 VARIABLES
   - 3.9 LOOPS
   - 3.10 FUNCTIONS
   - 3.11 MORE ABOUT OPERATORS
   - 3.12 TUTORIAL
   - 3.13 Q&A
   - 3.14 LAB
4. **ARRAY**
   - 4.1 INTRODUCTION
   - 4.2 COMPILING
   - 4.3 DEBUGGING
   - 4.4 ARRAYS
   - 4.5 STRINGS
     - 4.5.1 STRING LENGTH
   - 4.6 COMMAND-LINE ARGUMENTS
   - 4.7 EXIT STATUS
   - 4.8 CRYPTOGRAPHY
   - 4.9 TUTORIAL
   - 4.10 Q&A
   - 4.11 LAB
5. **PYTHON**
   - 5.1 INTRODUCTION
   - 5.2 HELLO PYTHON!
   - 5.3 SPELLER
   - 5.4 FUNCTIONS, LIBRARIES, MODULES, AND PACKAGES

5.5 STRINGS
5.6 POSITIONAL PARAMETERS AND NAMED PARAMETERS
5.7 VARIABLES AND TYPES
5.8 CALCULATOR
5.9 CONDITIONALS
5.10 LOOPS
5.11 ABSTRACTION
5.12 TRUNCATION AND FLOATING-POINT IMPRECISION
5.13 EXCEPTIONS
5.14 LISTS
5.14.1 METHODS
5.14.2 SEARCHING
5.15 COMMAND-LINE ARGUMENTS
5.16 EXIT STATUS
5.17 MORE ABOUT STRINGS
5.18 TUPLE, SET, AND DICTIONARY
5.19 TUTORIAL
5.20 Q&A
5.21 LAB
6. WHAT'S NEXT



1. WELCOME TO THE COURSE

This course is about much more than computer programming. The practical skills you'll gain here will shape how you learn, think, and solve problems — skills that extend far beyond computer science itself.

You'll begin with Scratch, a visual programming language that builds your understanding of logic and structure. Then, you'll move to C, where you'll explore how computers manage data and memory. Finally, you'll learn Python, a modern and powerful language that makes programming more readable and efficient — all without the complexities of object-oriented programming.

At its core, this course is about empowered problem-solving — developing the mindset and confidence to approach challenges creatively, logically, and systematically. The strategies you learn here will serve you not only in your studies and career, but also in everyday life.

Remember, this journey isn't about meeting a rigid standard — it's about growing from where you are today. Learning takes time, patience, and practice. Even when challenges arise, trust that consistent effort will lead to mastery.

And if this is your first experience with programming, don't worry, most of your classmates are starting here too. You're joining a supportive community of instructors, teaching assistants, and peers all committed to helping you learn, create, and succeed.


SCRATCH
when clicked
move 10 steps
say Hello! for 2 seconds

2. SCRATCH

By the end of this chapter, we will gain:

- Problem-solving is the essence of the work of computer scientists.
- This course is not simply about programming – this course will introduce you to a new way of learning that you can apply to almost every area of life.
- How numbers, text, images, and video are understood and represented by computers.
- The fundamental programming skill of pseudocoding.
- How will abstraction play a role in your future work in this course?
- The basic building blocks of programming include functions, conditionals, loops, and variables.
- How to build a project in Scratch.

2.1 COMPUTER SCIENCE AND PROBLEM SOLVING

Essentially, computer programming is about taking some input and creating some output - thus solving a problem. What happens in between the input and output, what we could call a black box, is the focus of this course.

Computers today count using a system called binary. It's from the term binary digit that we get a familiar term called bit. A bit is a zero or one: on or off. Computers only speak in terms of zeros and ones. Zeros represent off. Ones represent on. Computers are millions, and perhaps billions, of transistors that are being turned on and off.

If you imagine using a light bulb, a single bulb can only count from zero to one. However, if you were to have three light bulbs, there are more options open to you!

Inside your iPhone, there are millions of light bulbs called transistors that enable the activities this device one may take for granted each day.

As a heuristic, we could imagine that the following values represent each possible place in our binary digit:

4 2 1

Using three light bulbs, the following could represent zero:

4 2 1
0 0 0

Similarly, the following would represent one:

4 2 1
0 0 1

By this logic, we could propose that the following equals two:

4 2 1
0 1 0

Extending this logic further, the following represents three:

4 2 1
0 1 1

Four would appear as:

4 2 1
1 0 0

We could, in fact, using only three light bulbs count as high as seven!

4 2 1
1 1 1

Computers use base-2 to count. This can be pictured as follows:

2^2 2^1 2^0
4 2 1

Therefore, you could say that it would require three bits (the four's place, the two's place, and the one's place) to represent a number as high as seven.

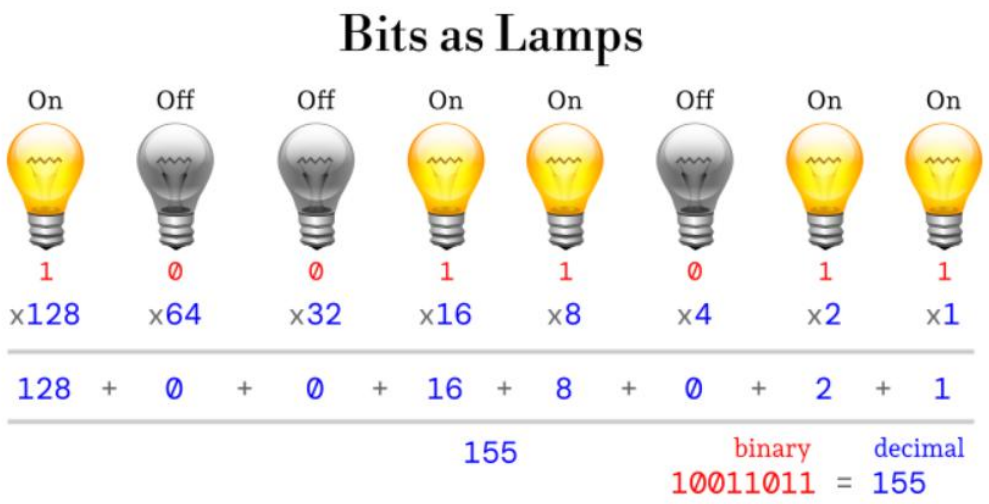
Similarly, to count a number as high as eight, values would be represented as follows:

8 4 2 1
1 0 0 0

Computers generally use eight bits (also known as a byte) to represent a number. For example, 00000101 is the number 5 in binary. 11111111 represents the number 255. You can imagine zero as follows:

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0

The number 255 is the maximum value that fits in a single byte. One byte has 8 bits of data. Each bit may only have two values: 0 or 1. With 2 bits, there are 4 possible combinations: 00, 01, 10, and 11. With 8 bits, there would be 256 different combinations, or 2 to the 8th power. We transform this combination to decimal numbers by interpreting them as binary numbers. Since there are 256 combinations of 8 bits, there are 256 different numbers that we can represent with 8 bits. One of these numbers is the zero (00000000), so we subtract this 1 number from the total of combinations to obtain the maximum value that one byte representing a non-negative integer number may have (11111111) which would be 255, or $2^8 - 1$.



Why do we use 255, not 256, while $2^8=256$?

2.2 ACSII

Just as numbers are binary patterns of ones and zeros, letters are represented using ones and zeros, too!
Since there is an overlap between the ones and zeros that represent numbers and letters, the ASCII (American Standard Code for Information Interchange) standard was created to map specific letters to specific numbers.
For example, the letter A was decided to map to the number 65. 01000001 represents the number 65 in binary. You can visualize this as follows:

128	64	32	16	8	4	2	1
0	1	0	0	0	0	0	1

If you received a text message, the binary under that message might represent the numbers 72, 73, and 33. Mapping these out to ASCII, your message would look as follows:

```
H   I   !
72  73  33
```

Here is an expanded map of ASCII values:

0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	”	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	’	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Q: Now, what exactly does ASCII represent?

A: ASCII is an acronym for American Standard Code for Information Interchange. This character encoding standard assigns a unique numerical value to letters, numbers, punctuation, and control characters for digital communication. The original 7-bit ASCII system represents 128 characters, including the English alphabet (both cases), digits, and common symbols, serving as the basis for modern character sets like Unicode.

Using one byte (8 bits) limits a single value to 0–255; computers can use more than 8 bits.

2.3 UNICODE

As time has rolled on, there are more and more ways to communicate via text.

Since there were not enough digits in binary to represent all the various characters that could be represented by humans, the Unicode standard expanded the number of bits that can be transmitted and understood by computers. Unicode includes not only special characters but also emojis.

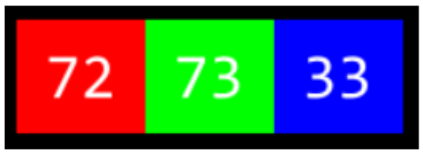
While the pattern of zeros and ones is standardized within Unicode, each device manufacturer may display each emoji slightly differently from another manufacturer. More and more features are being added to the Unicode standard to represent further characters and emoji.

Q: Do you know what emojis are?

A: There are emoji that you probably use every day. The following may look familiar to you:

2.4 RGB

Zeros and ones can be used to represent color. Red, green, and blue (called RGB) are a combination of three numbers.



Taking our previously used 72, 73, and 33, which said HI! via text, would be interpreted by image readers as a light shade of yellow. The red value would be 72, the green value would be 73, and the blue would be 33.

The three bytes required to represent various colors of red, blue, and green (or RGB) make up each pixel (or dot) of color in any digital image. Images are simply collections of RGB values.

Zeros and ones can be used to represent images, videos, and music! Videos are sequences of many images that are stored together, just like a flipbook.

Music can be represented similarly using various combinations of bytes.

2.5 ALGORITHMS

Problem-solving is central to computer science and computer programming. An algorithm is a step-by-step set of instructions to solve a problem. Imagine the basic problem of trying to locate a single name in a phone book.

Q: How might one go about this?

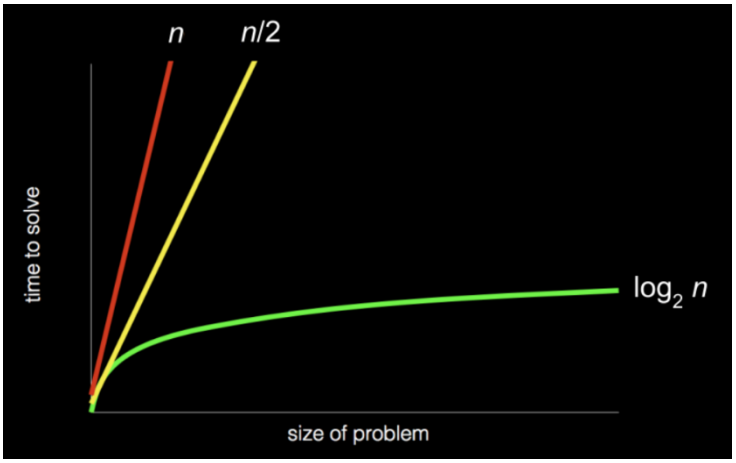
A: One approach could be to simply read from page one to the next to the next until reaching the last page.

Another approach could be to search two pages at a time.

A final and perhaps better approach could be to go to the middle of the phone book and ask, "Is the name I am looking for to the left or to the right?" Then, repeat this process, cutting the problem in half and half and half.

Each of these approaches could be called algorithms. The speed of each of these algorithms can be pictured as follows in what is called big-O notation:

Notice that the first algorithm, highlighted in red, has a big-O of n because if there are 100 names in the phone book, it could take up to 100 tries to find the correct name. The second algorithm, where two pages were searched at a time, has a big-O of n/2 because we searched twice as fast through the pages. The final algorithm has a big-O of log2n, as doubling the problem would only result in one more step to solve the problem.



Programmers translate text-based, human instructions into code.

2.6 PSEUDOCODE

This process of converting instructions into code is called pseudocode. The ability to create pseudocode is central to one's success in both this class and in computer programming.

Pseudocode is a human-readable version of your code. For example, considering the algorithm example in 2.5, we could compose pseudocode as follows:

```
1  Pick up phone book
2  Open to middle of phone book
3  Look at page
4  If person is on page
5      Call person
6  Else if person is earlier in book
7      Open to middle of left half of book
8      Go back to line 3
9  Else if person is later in book
10     Open to middle of right half of book
11     Go back to line 3
12 Else
13     Quit
```

Pseudocoding is such an important skill for at least two reasons. First, when your pseudocode before you create formal code, it allows you to think through the logic of your problem in advance. Second, when your pseudocode, you can later provide this information to others that are seeking to understand your coding decisions and how your code works.

Notice that the language within our pseudocode has some unique features. First, some of these lines begin with verbs like pick up, open, look at. Later, we will call these **Functions**.

Second, notice that some lines include statements like if or else if. These are called **Conditionals**.

Third, notice how there are expressions that can be stated as true or false, such as “person is earlier in the book.” We call these **Boolean expressions**.

Finally, notice how there are statements like “go back to line 3.” We call these **Loops**.

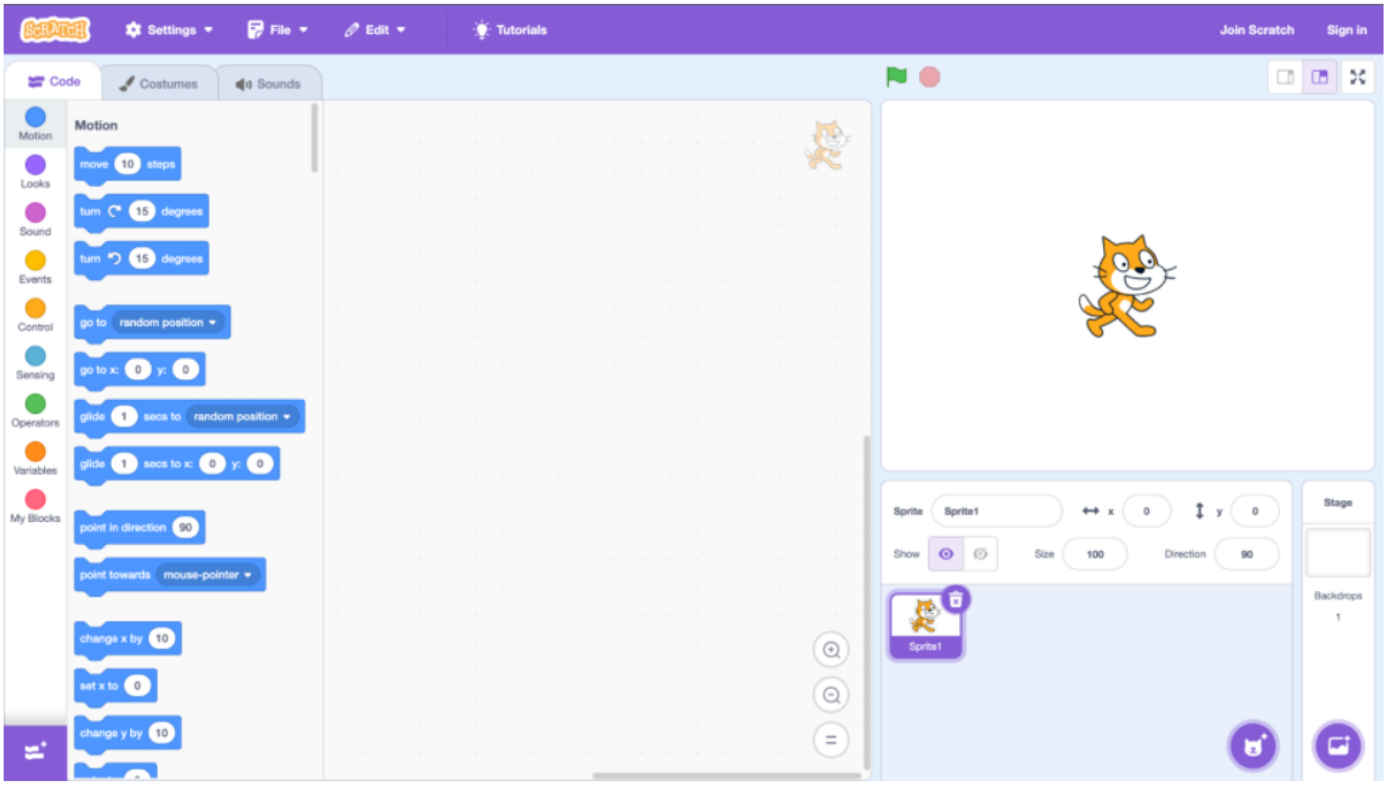
These **building blocks** are the fundamentals of programming.

In the context of Scratch, which is discussed later, we will use each of the above basic building blocks of programming.

2.7 SCRATCH

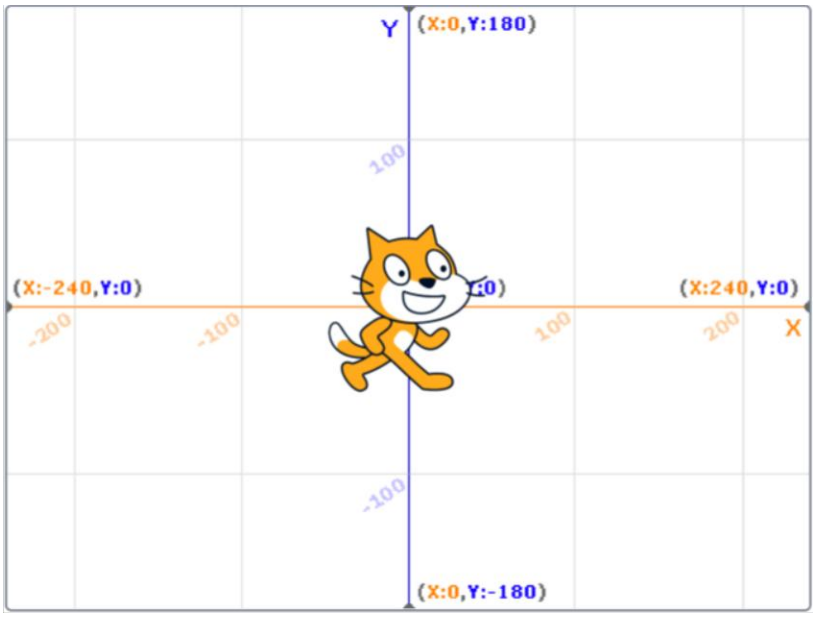
Scratch is a visual programming language developed by MIT. Scratch utilizes the same essential coding building blocks that we covered earlier in this lecture.

Scratch is a great way to get into computer programming because it allows you to play with these building blocks visually, not having to be concerned about the syntax of curly braces, semicolons, parentheses, and the like. Scratch **IDE** (integrated development environment) looks like the following:



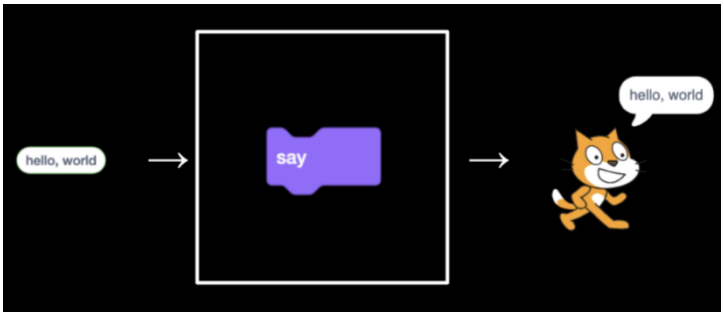
Notice that on the left, there is a palette of building blocks you can use in your programming. Immediately to the right of the building blocks, there is an area where you can drag blocks to construct a program. To the right of that, you see the stage where a cat stands. The stage is where your programming comes to life.

Scratch operates on a coordinate system, notice that the center of the stage is at coordinate (0,0). Right now, the cat's position is at that same position, as follows:



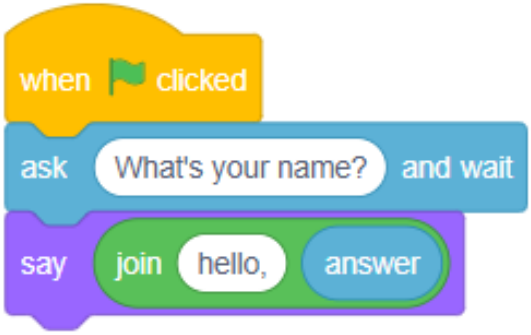
2.7.1 HELLO WORLD

To begin, drag the "when green flag clicked" building block to the programming area. Then, drag the say building block to the programming area and attach it to the previous block.



Notice that the input hello, world is passed to the function say, and the side effect of that function running is the cat saying hello, world.

Now, we can make your program more interactive by having the cat say hello to someone specific. Modify your program as below:

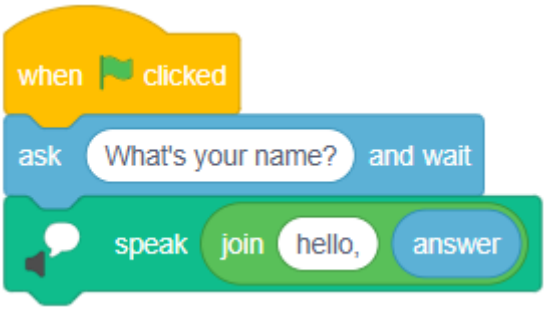


Notice that when the green flag is clicked, the function ask is run. The program prompts you, the user, What's your name? It then stores that name in the variable called answer. The program then passes answer to a special function called join, which combines two strings of text hello, and whatever name was provided. Quite literally, answer returns a value to join. These collectively are passed to the say function. The cat says, Hello, and a name. Your program is now interactive.

Why can we say that the previous program is interactive?

Notice that the inputs hello, and answer are provided to join, resulting in the side effect of hello, Mohammed.

Quite similarly, we can modify our program as follows:



Notice that this program, when the green flag is clicked, passes the same variable, joined with hello, to a function called speak.

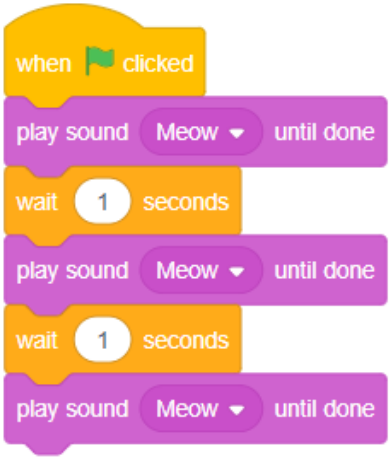
2.7.2 ABSTRACTION

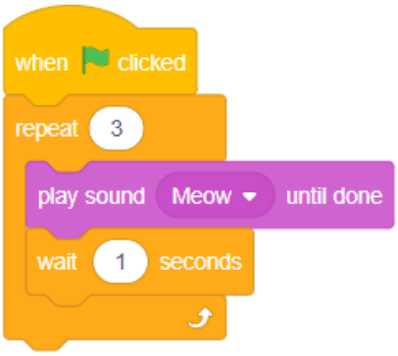
Along with pseudocoding, abstraction is an essential skill and concept within computer programming. Abstraction is the act of simplifying a problem into smaller and smaller problems.

In programming, and even within Scratch, we can see abstraction in action. In your programming area, program as follows:

Notice that you are doing the same thing repeatedly. Indeed, if you see yourself repeatedly coding the same statements, it's likely the case that you could program more artfully – abstracting away this repetitive code.

We can modify your code as follows:





Notice that the loop does exactly as the previous program did. However, the problem is simplified by abstracting away the repetition to a block that repeats the code for us.

We can even advance this further by using the define block, where you can create your own block (your own function)! Write code as follows:

Notice that we are defining our own block called meow. The function plays the sound meow, and then waits one second. Besides that, we can see that when the green flag is clicked, our meow function is repeated three times.

We can even provide a way by which the function can take an input n and repeat several times:

Notice how n is taken from “meow n times.” n is passed to the meow function through the define block.

Overall, notice how this process of refinement led to better and better-designed code. Further, notice how we created our own algorithm to solve a problem. You will be exercising both skills throughout this course.

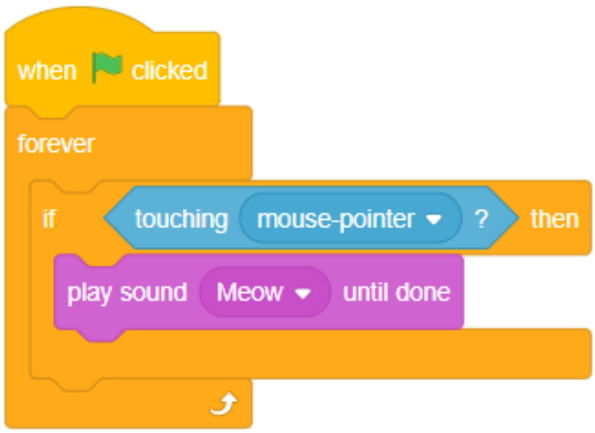
2.7.3 CONDITIONALS

Conditionals are an essential building block of programming, where the program looks to see if a specific condition has been met. If a condition is met, the program does something.

To illustrate a conditional, write code on the right:

Notice that the forever block is utilized such that the if block is triggered repeatedly, such that it can check continuously if the cat is touching the mouse pointer.

Remember, programming is often a process of trial and error. If you get frustrated, take time to talk yourself through the problem at hand. What is the specific problem that you are working on right now? What is working? What is not working?



2.8 TUTORIAL

Now we are familiar with the basics of Scratch, let's begin our first hands-on tutorial where we will apply these concepts to build and test our own interactive program.

2.8.1 Catch me if you can.

Visit: https://scratch.mit.edu/projects/565479840/editor/

Notice that when the green flag is clicked, our sprite moves to the center of the stage at coordinates (0,0) and then listens for the keyboard and checks for walls forever. we have created a custom *listen for keyboard* script. For each of our arrow keys on the keyboard, it will move the sprite around the screen, and have another *custom feel for walls* script. When a sprite touches a wall, it moves it back to a safe position – preventing it from walking off the screen. We can even make a sprite follow another sprite.

2.8.2 Pong Game

Visit: https://scratch.mit.edu/projects/1191014517/editor/

We have three sprites: the ball, paddle, and ground line. The instructions are to move the paddle with your finger to prevent the ball from touching the ground line. When the green flag is clicked, the ball moves in a 45° direction. If it touches the paddle, it reflects its direction to 180 minus the current direction, and the score variable increases by 1. The paddle changes its x value according to the mouse x-axis.

For more projects, visit: https://scratch.mit.edu/explore/projects/all

2.9 Q&A

1. Why do computers use binary instead of the decimal system?
2. What does one byte represent, and what is its maximum value?
3. What is the advantage of representing data in binary?
4. What is ASCII used for in computer systems?
5. How does Unicode improve upon ASCII?
6. What color does the RGB combination (255, 0, 0) produce?
7. If all RGB values are equal (e.g., 128, 128, 128), what color do we get?
8. What is an algorithm?
9. Why is pseudocode important before writing a program?
10. What is the role of the “forever” block?
11. What does abstraction mean in programming?
12. What is the benefit of using loops?
13. What is a conditional statement?
14. What happens inside a “forever if” loop?

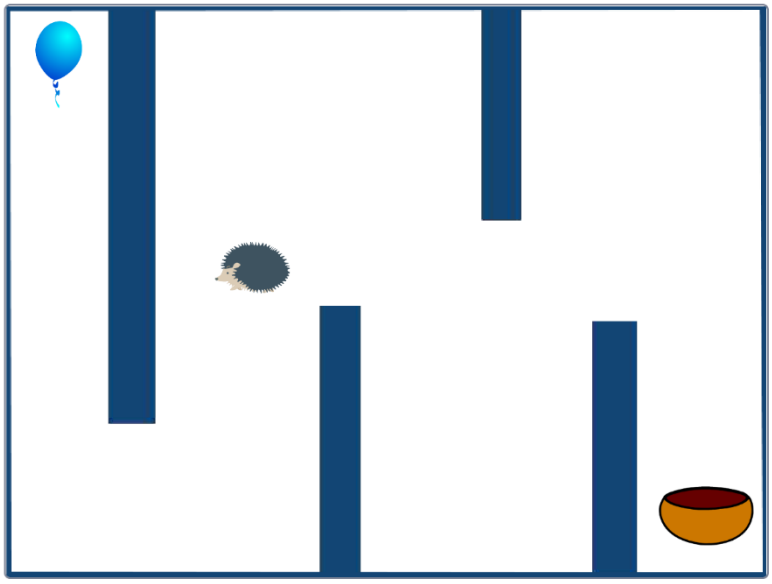
Answers

1. Because computer hardware is based on transistors that have two stable states — on (1) and off (0) — making binary representation the most efficient and reliable system.
2. One byte equals 8 bits and can represent 256 combinations (0–255).
3. Binary provides a simple, universal method for encoding all types of data — text, images, and sound — in a digital format that hardware can process.
4. ASCII encodes English letters, digits, and symbols into numeric codes that computers can understand (e.g., 'A' = 65).
5. Unicode provides a universal standard that supports characters from all languages, including Arabic and symbols like emojis.
6. Pure red.
7. A shade of gray.
8. A step-by-step procedure to solve a specific problem.
9. It helps plan and visualize the logic of the program in human-readable form before coding.
10. It repeats the enclosed commands continuously if the program is running.
11. Simplifying a complex system by breaking it into smaller, more manageable parts and hiding unnecessary details.
12. Loops reduce code repetition and make programs more efficient and readable.
13. A statement that allows a program to make decisions — executing certain actions only if a condition is true.
14. The program continuously checks the condition and executes the action whenever it becomes true.

2.10 LAB

1. Let's try to make some changes to modify the previous example (Pong game) by adding a paddle, one on the right and the other on the left. To control their movement, they should use the keyboard. The player who fails to block the ball three times will lose the game, as shown below. Keep in mind that the game should not fall into the redundancy issue (direction).

2. One of the most popular games is Maze, as shown below. Use the arrow keys to move your character through the maze and reach the bowl. If you reach it, the bowl will say "You win!" Avoid touching the maze walls—any contact sends you back to the starting point. Watch out for the horizontally moving enemy; if it touches you, the game is over.



Extra jobs:

- In Pong-game, add score to 10 (instead of 3 rounds) and increase speed after score number 5.
- In Pong-game, instead of two-players, try to challenge simple AI paddle (follows ball with delay).
- In Maze game, increase enemy speed after n second.