Este é o CS50

Introdução à Ciência da Computação (CS50)

OpenCourseWare

Doar (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

Aula 0

- Bem-vindo!
- Comunidade!
- Ciência da Computação e Resolução de Problemas
- ASCII
- Unicode
- RGB
- Algoritmos
- Pseudocódigo
- Inteligência artificial
- O que vem a seguir
- Arranhar
- Olá, mundo!
- Olá, você!
- Miau e Abstração
- Condicionais
- Oscartime
- O Jogo Mais Difícil de Ivy
- Resumindo

Bem-vindo!

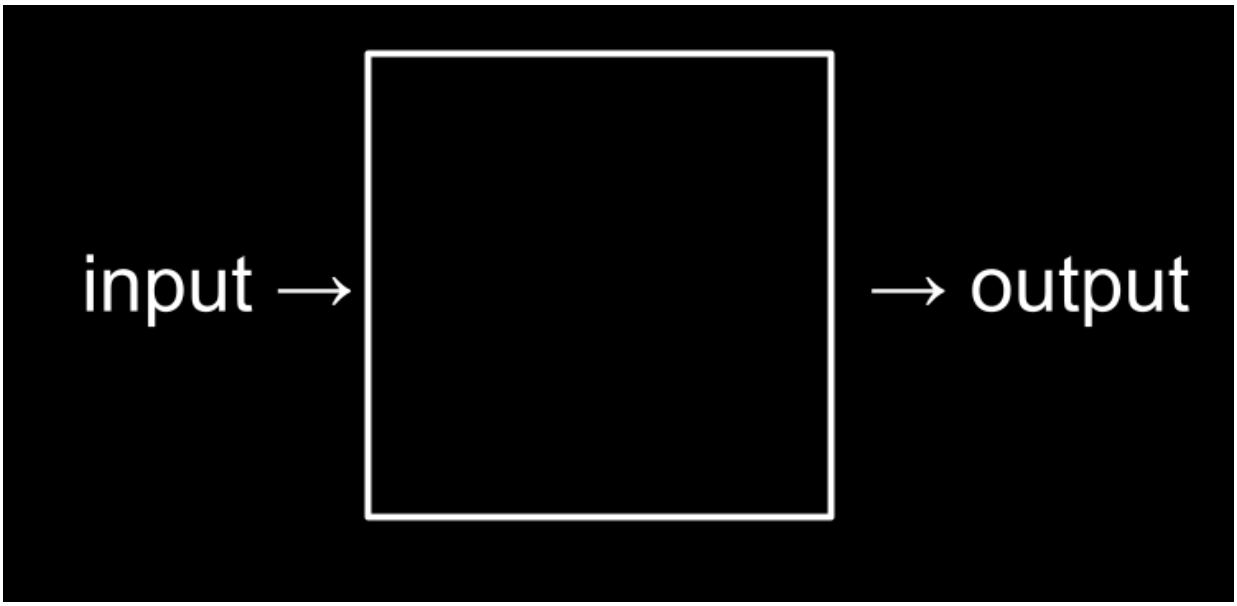
- Esta aula vai muito além da programação de computadores! As habilidades práticas que você aprenderá aqui podem ter um impacto na sua vida e no seu aprendizado muito além da ciência da computação.
- De fato, esta aula trata da resolução de problemas de uma forma extremamente empoderadora! É provável que você leve consigo as habilidades de resolução de problemas aprendidas aqui, que provavelmente serão imediatamente aplicáveis ao seu trabalho após este curso e até mesmo à sua carreira como um todo!
- No entanto, não será fácil! Você absorverá um fluxo intenso de conhecimento durante este curso. Você ficará surpreso com o que será capaz de realizar nas próximas semanas.
- Este curso tem muito mais a ver com o seu progresso pessoal, partindo do ponto em que você está hoje, do que com atingir algum padrão imaginário.
- A consideração inicial mais importante neste curso é: dedique o tempo necessário para aprender. Cada pessoa aprende de uma maneira diferente. Se algo não funcionar bem no começo, saiba que com o tempo você irá se aprimorar cada vez mais.
- Não se assuste se esta for sua primeira aula de ciência da computação! Para a maioria dos seus colegas, também é a primeira vez! Além disso, os monitores, assistentes de curso e seus colegas estão aqui para te ajudar!

Comunidade!

- Você faz parte de uma comunidade de pessoas que estão cursando esta disciplina no Harvard College, na Harvard Extension School e através do edX.org.
- Esperamos que você se junte a nós (seja presencialmente ou virtualmente) no CS50 Puzzle Day (https://cs50.harvard.edu/college/2024/fall/puzzles/) e na Feira CS50 (https://www.youtube.com/watch?v=JJPbXou4-0o&list=PLhQjrBD2T381cvtjW82tGZBjIdanUUnk9) .
- Se você for aluno do campus de Harvard, pode participar dos almoços CS50 e do Hackathon CS50 . (https://youtu.be/wTT5ahmaUAc?si=C1h4vW3OYM6NVwKu)

Ciência da Computação e Resolução de Problemas

- Essencialmente, a programação de computadores consiste em receber uma entrada e gerar uma saída — resolvendo, assim, um problema. O que acontece entre a entrada e a saída, o que podemos chamar *de caixa preta,* é o foco deste curso.



- Por exemplo, podemos precisar fazer a chamada em uma aula. Poderíamos usar um sistema chamado *unário* (também chamado de *base 1*) para contar um dedo de cada vez.
- Os computadores atuais utilizam um sistema de contagem chamado *binário* . É a partir do termo " *dígito binário*" que deriva o termo familiar " *bit*" . Um *bit* é zero ou um: ligado ou desligado.
- Os computadores só se comunicam em termos de zeros e uns. Zeros representam *desligado*. Uns representam *ligado*. Os computadores são milhões, e talvez bilhões, de transistores que são ligados e desligados.
- Se você imaginar usar uma lâmpada, uma única lâmpada só pode contar de zero a um.
- No entanto, se você tiver três lâmpadas, terá mais opções à sua disposição!
- Dentro do seu iPhone, existem milhões de lâmpadas chamadas *transistores* que possibilitam as atividades que este dispositivo realiza no dia a dia, muitas vezes sem que percebamos.
- Como heurística, podemos imaginar que os seguintes valores representam cada posição possível em nosso *dígito binário* :

```
4 2 1
```

- Usando três lâmpadas, o seguinte poderia representar o zero:

```
4 2 1
0 0 0
```

- Da mesma forma, o seguinte representaria um exemplo:

```
4 2 1
0 0 1
```

- Seguindo essa lógica, poderíamos propor que o seguinte seja igual a dois:

```
4 2 1
0 1 0
```

- Estendendo ainda mais essa lógica, o seguinte representa três:

```
4 2 1
0 1 1
```

- Quatro apareceriam como:

```
4 2 1
1 0 0
```

- Na verdade, usando apenas três lâmpadas, poderíamos contar como se fossem sete!

```
4 2 1
1 1 1
```

- Os computadores usam o sistema binário (base 2) para contar. Isso pode ser visualizado da seguinte forma:

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

- Portanto, pode-se dizer que seriam necessários três bits (o bit do quatro, o bit do dois e o bit do um) para representar um número tão alto quanto sete.
- Da mesma forma, para contar um número tão grande quanto oito, os valores seriam representados da seguinte maneira:

```
8 4 2 1
1 0 0 0
```

- Os computadores geralmente usam oito bits (também conhecidos como *bytes*) para representar um número. Por exemplo, 5 `00000101` representa o número 5 em *binário* . `11111111` 0 representa o número 255. Você pode imaginar o zero da seguinte forma:

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

ASCII

- Assim como os números são padrões binários de uns e zeros, as letras também são representadas usando uns e zeros!
- Como existe uma sobreposição entre os uns e zeros que representam números e letras, o padrão *ASCII* foi criado para mapear letras específicas para números específicos.
- Por exemplo, `A` decidiu-se que a letra corresponderia ao número 65. `01000001` representa o número 65 em binário. Você pode visualizar isso da seguinte forma:

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

- Se você recebeu uma mensagem de texto, o código binário dessa mensagem pode representar os números 72, 73 e 33. Convertendo esses números para ASCII, sua mensagem ficaria assim:

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

- Ainda bem que existem padrões como o ASCII que nos permitem concordar com esses valores!
- Segue abaixo um mapa expandido dos valores ASCII:

0	NULO	16	DLE	32	SP	48	0	64	@	80	P	9
1	SOH	17	DC1	33	!	49	1	65	UM	81	Q	9
2	STX	18	DC2	34	”	50	2	66	B	82	R	9
3	ETX	19	DC3	35	#	51	3	67	C	83	S	9
4	EOT	20	DC4	36	$	52	4	68	D	84	T	1
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	1
6	ACK	22	SINÔNIMO	38	&	54	6	70	F	86	V	1
7	BEL	23	ETB	39	'	55	7	71	G	87	C	1
8	BS	24	PODE	40	(	56	8	72	H	88	X	1
9	HT	25	EM	41	)	57	9	73	EU	89	Y	1
10	LF	26	SUB	42	*	58	:	74	J	90	Z	1
11	VT	27	ESC	43	+	59	;	75	K	91	[	1
12	FF	28	FS	44	,	60	<	76	L	92	\	1
13	CR	29	GS	45	-	61	=	77	M	93	]	1
14	ENTÃO	30	RS	46	.	62	>	78	N	94	^	1

0	NULO	16	DLE	32	SP	48	0	64	@	80	P	9(
15	SI	31	NÓS	47	/	63	?	79	O	95	_	11

- Se desejar, você pode aprender mais sobre ASCII (https://en.wikipedia.org/wiki/ASCII) .
- Como o sistema binário só consegue contar até *255,* estamos limitados ao número de caracteres representados pelo ASCII.

Unicode

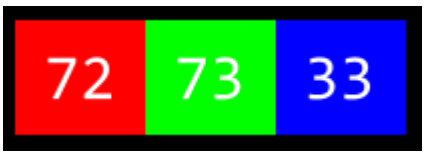
- Com o passar do tempo, surgiram cada vez mais maneiras de se comunicar por mensagem de texto.
- Como o sistema binário não possuía dígitos suficientes para representar todos os caracteres que poderiam ser representados por humanos, o padrão *Unicode* expandiu o número de bits que podem ser transmitidos e compreendidos pelos computadores. O Unicode inclui não apenas caracteres especiais, mas também emojis.
- Existem emojis que você provavelmente usa todos os dias. Os seguintes podem lhe parecer familiares:

  😀 😃 😄 😁 😆 😅 😂 🙂 🙃 😉 😊 😇 😍 😘 😗 😙 😚 😋 😛 😜 😝 🤑 🤓 😎 🤗 😏 😶
  😐 😑 😒 🙄 😬 😕 ☹ 😟 😮 😯 😲 😳 😦 😧 😨

- Embora o padrão de zeros e uns seja padronizado dentro do Unicode, cada fabricante de dispositivo pode exibir cada emoji de maneira ligeiramente diferente de outro fabricante.
- Cada vez mais recursos estão sendo adicionados ao padrão Unicode para representar mais caracteres e emojis.
- Se desejar, você pode aprender mais sobre Unicode (https://en.wikipedia.org/wiki/Unicode) .
- Se desejar, você pode aprender mais sobre emojis (https://en.wikipedia.org/wiki/Emoji) .

RGB

- Os números zero e uns podem ser usados para representar cores.
- Vermelho, verde e azul (chamados de `RGB`) são uma combinação de três números.

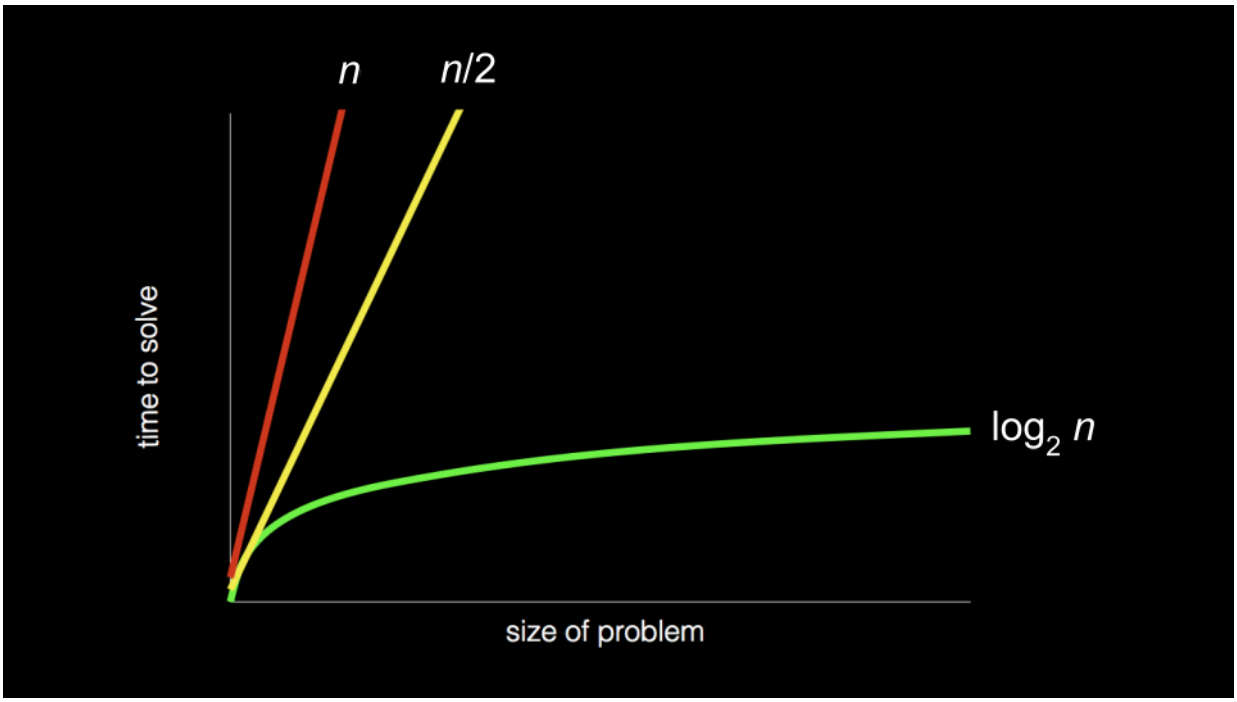
  

- Considerando os valores 72, 73 e 33 que usamos anteriormente e que foram expressos `HI!` em texto, os leitores de imagem os interpretariam como um tom claro de amarelo. O valor vermelho seria 72, o valor verde seria 73 e o azul seria 33.

- Os três bytes necessários para representar as diversas cores vermelha, azul e verde (ou *RGB*) compõem cada *pixel* (ou ponto) de cor em qualquer imagem digital. Imagens são simplesmente coleções de valores RGB.
- Os números zero e uns podem ser usados para representar imagens, vídeos e música!
- Vídeos são sequências de várias imagens armazenadas juntas, como um flipbook.
- A música pode ser representada de forma semelhante usando várias combinações de bytes.

Algoritmos

- A resolução de problemas é fundamental para a ciência da computação e a programação de computadores. Um *algoritmo* é um conjunto de instruções passo a passo para resolver um problema.
- Imagine o problema básico de tentar localizar um único nome em uma lista telefônica.
- Como se poderia fazer isso?
- Uma abordagem possível seria simplesmente ler da página um para a seguinte, até chegar à última página.
- Outra abordagem seria pesquisar duas páginas por vez.
- Uma abordagem final, e talvez melhor, seria ir até o meio da lista telefônica e perguntar: "O nome que estou procurando está à esquerda ou à direita?" Em seguida, repita esse processo, dividindo o problema ao meio, depois ao meio novamente e depois ao meio novamente.
- Cada uma dessas abordagens pode ser chamada de algoritmo. A velocidade de cada um desses algoritmos pode ser representada da seguinte forma, na chamada *notação Big-O* :



Observe que o primeiro algoritmo, destacado em vermelho, tem uma complexidade de notação Big-O de O(n log n), `n` pois se houver 100 nomes na lista telefônica, pode levar até 100 tentativas para encontrar o nome correto. O segundo algoritmo, em que duas páginas foram pesquisadas por vez, tem uma complexidade de notação Big-O de O(n log n), `n/2` pois a busca nas páginas foi realizada duas vezes mais rápido. O algoritmo final tem uma complexidade de notação Big-O de $\log_2 n$, já que dobrar o problema resultaria apenas em mais um passo para resolvê-lo.

- Programadores traduzem instruções humanas baseadas em texto para código.

Pseudocódigo

- Esse processo de conversão de instruções em código é chamado de *pseudocódigo* .
- A capacidade de criar *pseudocódigo* é fundamental para o sucesso tanto nesta disciplina quanto na programação de computadores.
- O pseudocódigo é uma versão do seu código que pode ser lida por humanos. Por exemplo, considerando o terceiro algoritmo acima, poderíamos compor o pseudocódigo da seguinte forma:

```
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
```

- A pseudocodificação é uma habilidade muito importante por pelo menos dois motivos. Primeiro, ao usar pseudocodificação antes de criar o código formal, você consegue pensar na lógica do problema antecipadamente. Segundo, ao usar pseudocodificação, você pode fornecer essas informações posteriormente a outras pessoas que desejam entender suas decisões de codificação e como seu código funciona.
- Observe que a linguagem em nosso pseudocódigo possui algumas características únicas. Primeiro, algumas dessas linhas começam com verbos como *pegar, abrir, olhar.* Mais tarde, chamaremos essas *funções de* .
- Em segundo lugar, observe que algumas linhas incluem declarações como `if` ou . `else if.` Estas são chamadas *de condicionais* .
- Terceiro, observe como existem expressões que podem ser declaradas como *verdadeiras* ou *falsas,* como "a pessoa está mais cedo no livro". Chamamos essas *expressões de expressões booleanas* .
- Por fim, observe como existem instruções como "volte para a linha 3". Chamamos isso de *loops* .
- Esses elementos básicos são os fundamentos da programação.
- No contexto do *Scratch* , que será discutido a seguir, utilizaremos cada um dos blocos de construção básicos da programação mencionados acima.

Inteligência artificial

- Considere como podemos utilizar os elementos básicos acima para começar a criar nossa própria inteligência artificial. Observe o seguinte pseudocódigo:

  ```
  If student says hello
      Say hello
  Else if student says goodbye
      Say goodbye
  Else if student asks how you are
      Say well
  Else if student asks why 111 in binary is 7 in decimal
  ...
  ```

  Observe como, para programar apenas algumas interações, seriam necessárias muitas linhas de código. Quantas linhas de código seriam necessárias para milhares ou dezenas de milhares de interações possíveis?
- Em vez de programar IA conversacional como a descrita acima, os programadores de IA treinam *grandes modelos de linguagem* (LLMs) em grandes conjuntos de dados.
- Os Modelos de Linguagem Linguística (LLMs) analisam padrões em grandes blocos de linguagem. Esses modelos tentam criar a melhor estimativa possível de quais palavras vêm umas após as outras ou lado a lado.
- Embora o software baseado em IA seja muito útil em muitas áreas da vida e do trabalho, estipulamos que o uso de software baseado em IA que não seja o próprio da CS50 *não é razoável* .

- A ferramenta de software própria da CS50, baseada em IA, chamada CS50.ai (https://cs50.ai) , é um auxiliar de IA que você pode usar durante este curso. Ela irá ajudá-lo(a), mas não fornecerá todas as respostas para os problemas do curso.
- Neste curso, não é permitido usar nenhuma IA, exceto a CS50.ai. (https://cs50.ai)

O que vem a seguir

- Nesta semana você aprenderá sobre o Scratch, uma linguagem de programação visual.
- Nas próximas semanas, você aprenderá sobre a linguagem C. Será algo parecido com isto:

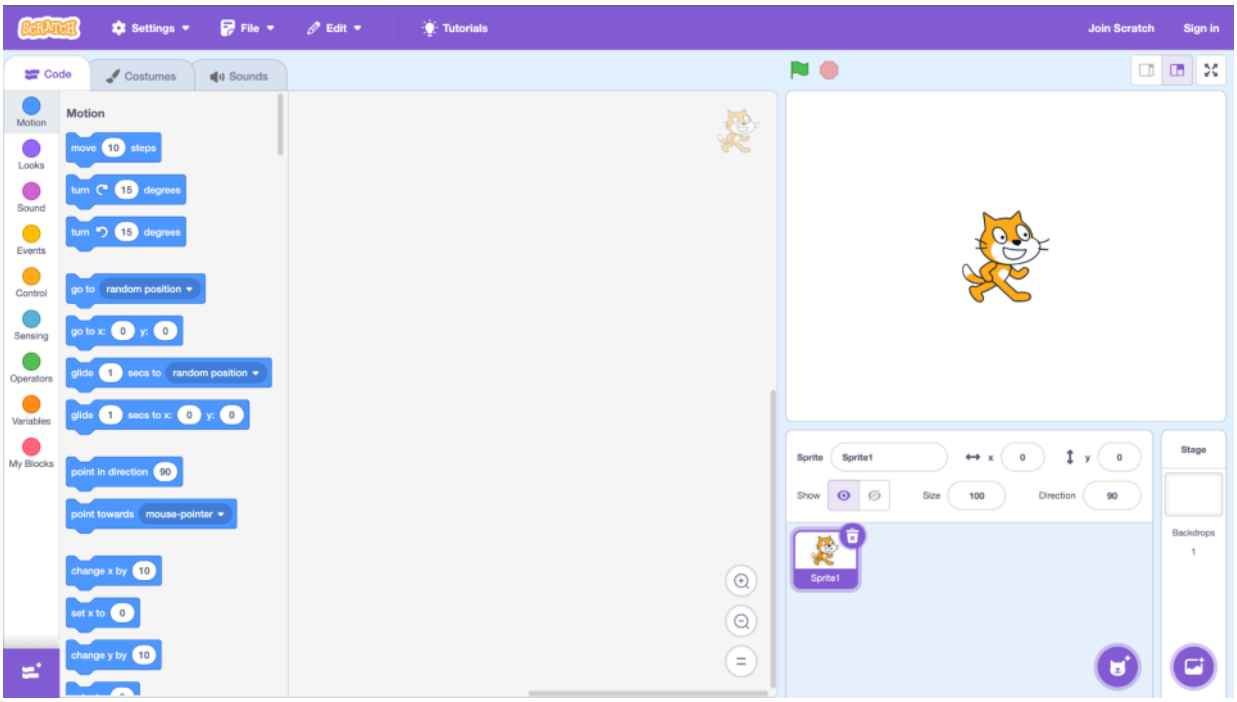
```c
#include <stdio.h>

int main(void)
{
  printf("hello, world\n");
}
```

- Ao aprender C, você estará muito mais preparado para aprender outras linguagens de programação no futuro, como *Python* .
- Além disso, conforme as semanas forem passando, você aprenderá sobre algoritmos.
- O que torna a linguagem C tão desafiadora é a pontuação. Deixando de lado a pontuação e a sintaxe por hoje, vamos trabalhar exclusivamente com ideias em uma linguagem de programação chamada Scratch.

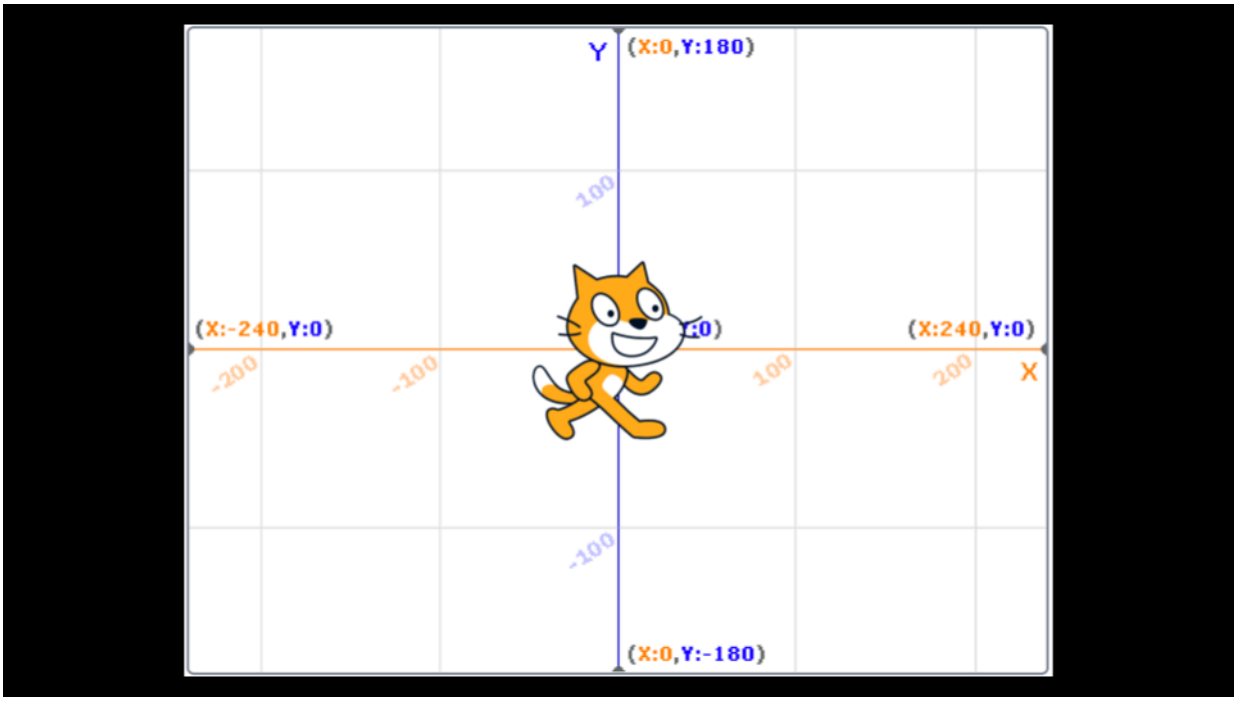
Arranhar

- *Scratch* é uma linguagem de programação visual desenvolvida pelo MIT.
- O Scratch utiliza os mesmos blocos de construção essenciais de programação que abordamos anteriormente nesta aula.
- O Scratch é uma ótima maneira de começar a programar, pois permite que você brinque com esses blocos de construção de forma visual, sem precisar se preocupar com a sintaxe de chaves, ponto e vírgula, parênteses e coisas do tipo.
- O Scratch `IDE` (ambiente de desenvolvimento integrado) tem a seguinte aparência:



Observe que, à esquerda, há uma paleta de *blocos de construção* que você pode usar em sua programação. Imediatamente à direita dos blocos de construção, há uma área para a qual você pode arrastar blocos para construir um programa. À direita disso, você vê o *palco* onde um gato está. O palco é onde sua programação ganha vida.

- Scratch operates on a coordinate system as follows:

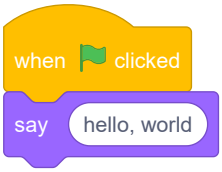


Notice that the center of the stage is at coordinate (0,0). Right now, the cat's position is at that same position.

Hello World

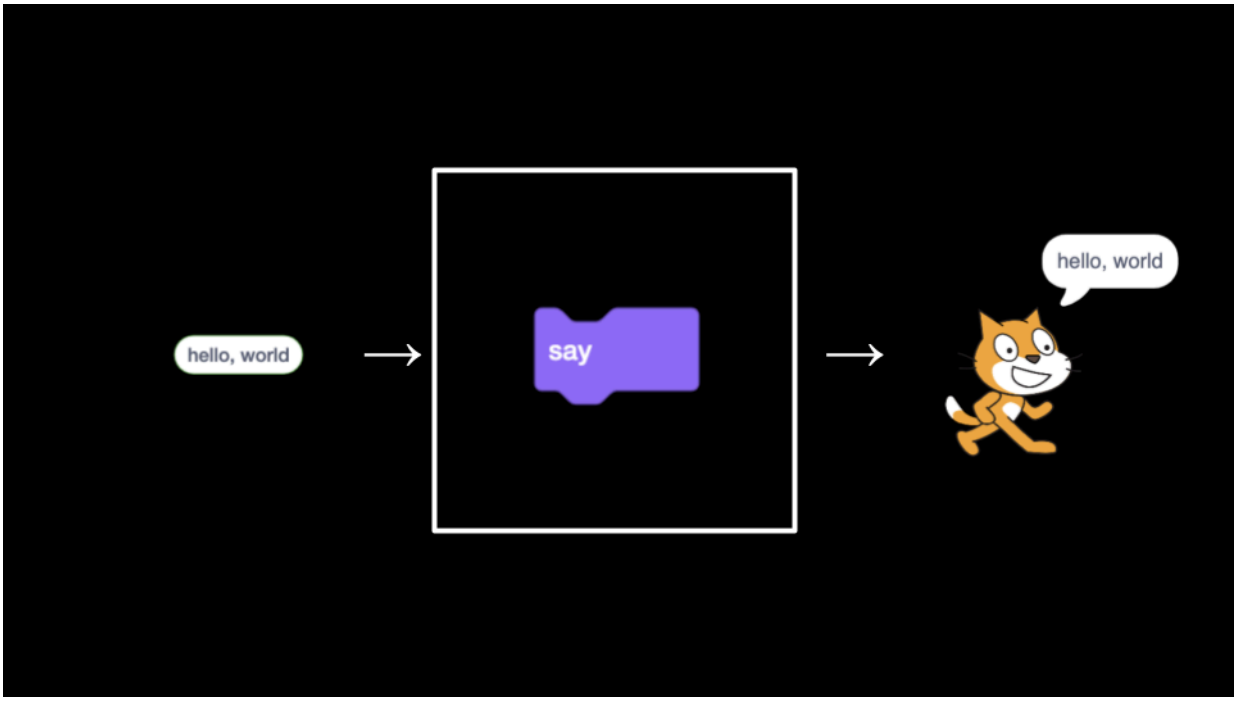
- To begin, drag the "when green flag clicked" building block to the programming area. Then, drag the `say` building block to the programming area and attach it to the previous

block.



Notice that when you click the green flag now on the stage, the cat says, "hello, world."

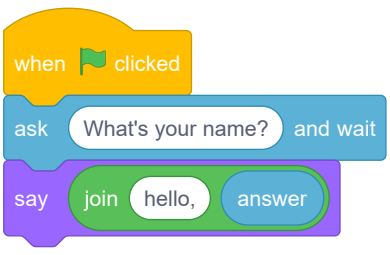
- This illustrates quite well what we were discussing earlier regarding programming:



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`.

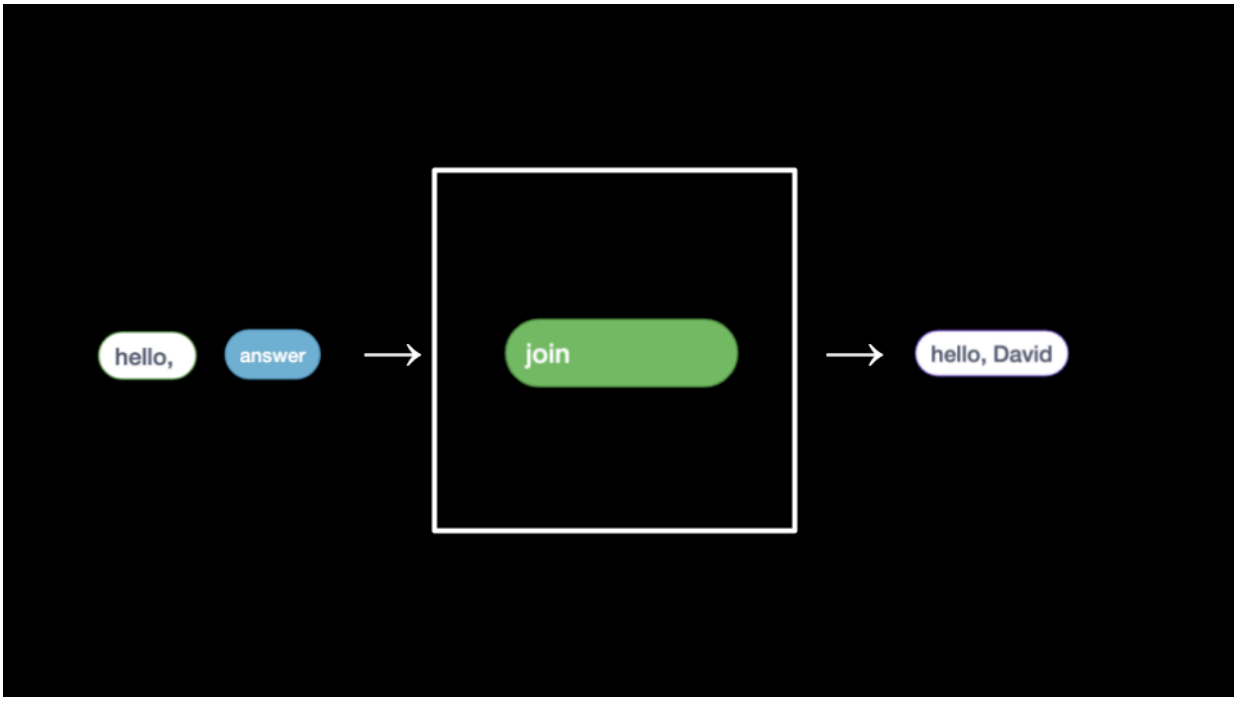
Hello, You

- We can make your program more interactive by having the cat say `hello` to someone specific. Modify your program as below:



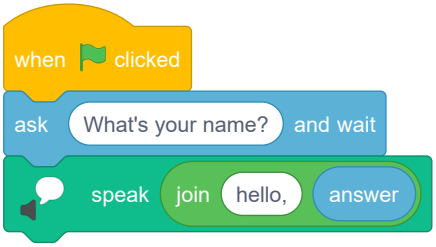
Notice that when the green flag is clicked, the function `ask` is run. The program prompts you, the user, `What's your name?` It then stores that name in the *variable* called `answer`. The program then passes `answer` to a special function called `join`, which combines two strings of text `hello`, and 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.

- Throughout this course, you will be providing inputs into an algorithm and getting outputs (or side effects). This can be pictured in terms of the above program as follows:



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

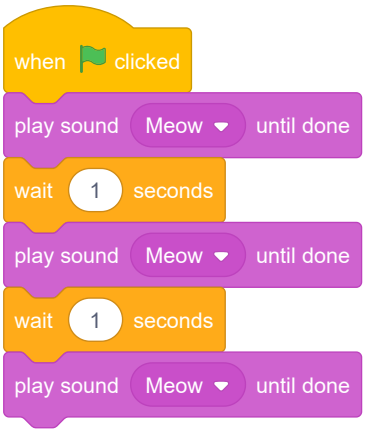
- Quite similarly, we can modify our program as follows:



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

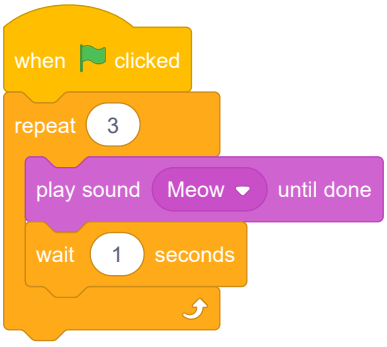
Meow and Abstraction

- 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.
- For example, if you were hosting a huge dinner for your friends, the *problem* of having to cook the entire meal could be quite overwhelming! However, if you break down the task of cooking the meal into smaller and smaller tasks (or problems), the big task of creating this delicious meal might feel less challenging.
- In programming, and even within Scratch, we can see abstraction in action. In your programming area, program as follows:



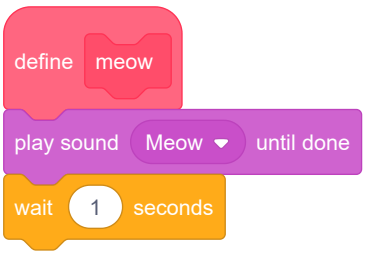
Notice that you are doing the same thing over and over again. Indeed, if you see yourself repeatedly coding the same statements, it's likely the case that you could program more artfully – abstracting away this repetitive code.

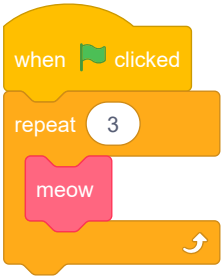
- You can modify your code as follows:



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.

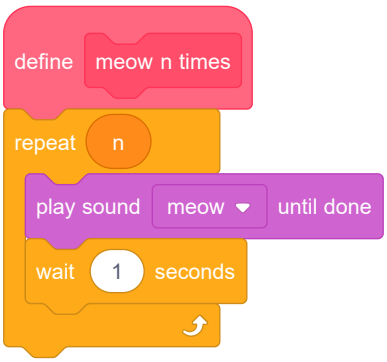
- We can even advance this further by using the `define` block, where you can create your own block (your own function)! Write code as follows:





Notice that we are defining our own block called `meow`. The function plays the sound `meow`, and then waits one second. Below that, you 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 a number of times:

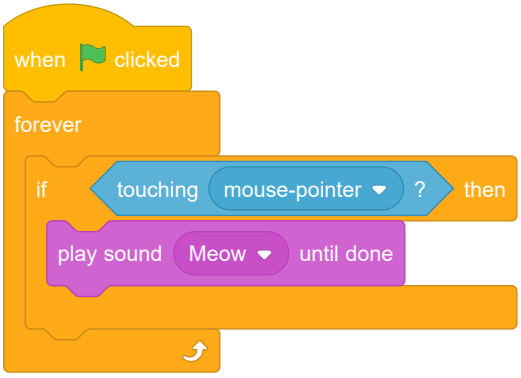


Notice how `n` is taken from "meow n times." `n` is passed to the meow function through the `define` block.

- 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 of these skills throughout this course.

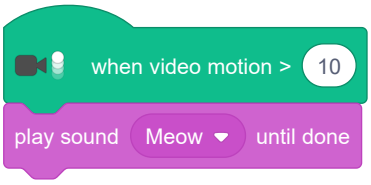
Conditionals

- *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 as follows:



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

- We can modify our program as follows to integrate video sensing:

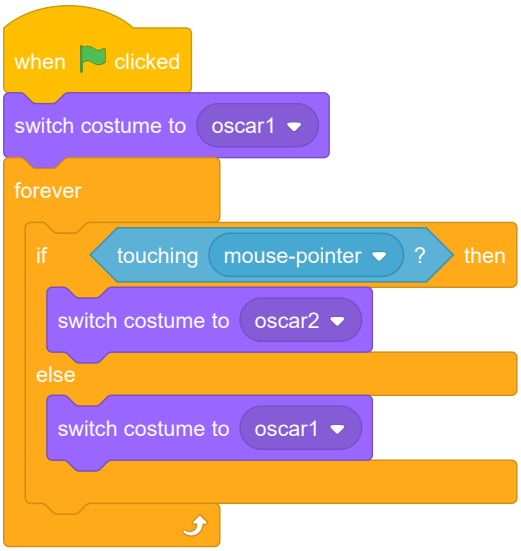


- Remember, programming is often a process of trial and error. If you get frustrated, take time to talk yourself through the problem at hand. What is the specific problem that you are working on right now? What is working? What is not working?

Oscartime

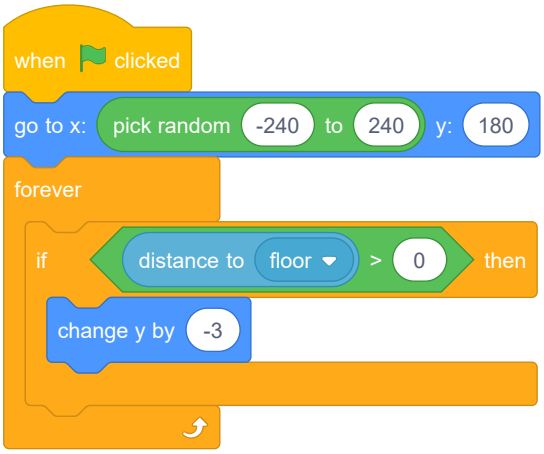
- *Oscartime* is one of David's own Scratch programs – though the music may haunt him because of the number of hours he listened to it while creating this program. Take a few moments to play through the game yourself.
- Building Oscartime ourselves, we first add the lamp post.

- Then, write code as follows:



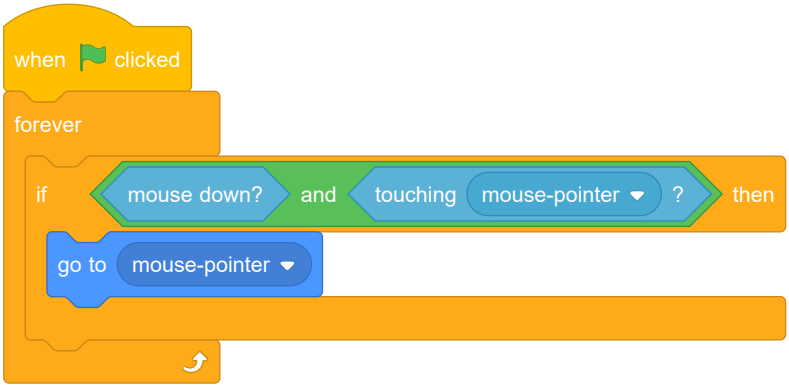
Notice that moving your mouse over Oscar changes his costume. You can learn more by exploring these code blocks (https://scratch.mit.edu/projects/565100517).

- Then, modify your code as follows to create a falling piece of trash:



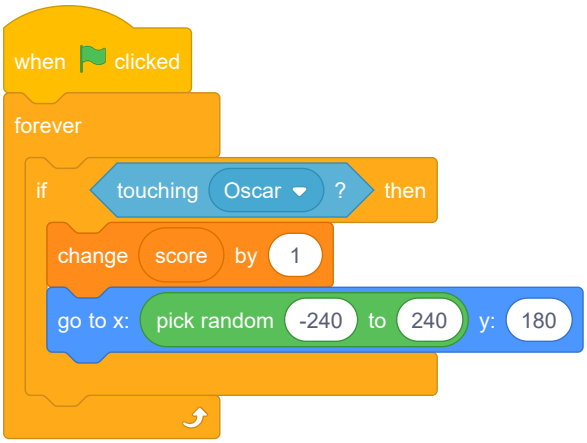
Notice that the trash's position on the y-axis always begins at 180. The x position is randomized. While the trash is above the floor, it goes down 3 pixels at a time. You can learn more by exploring these code blocks (https://scratch.mit.edu/projects/565117390).

- Next, modify your code as follows to allow for the possibility of dragging trash.



You can learn more by exploring these code blocks (https://scratch.mit.edu/projects/565119737).

- Next, we can implement the scoring variables as follows:

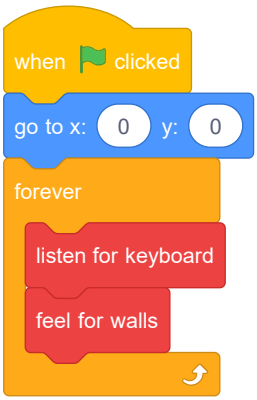


You can learn more by exploring these code blocks (https://scratch.mit.edu/projects/565472267).

- Go try the full game Oscartime (https://scratch.mit.edu/projects/277537196).

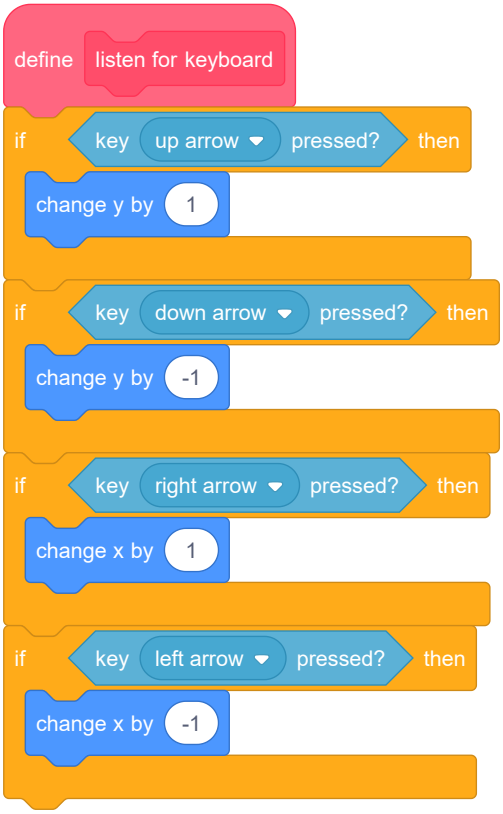
Ivy's Hardest Game

- Deixando de lado o período dedicado ao Oscar e passando para o jogo mais difícil da Ivy League, podemos agora imaginar como implementar o movimento dentro do nosso programa.
- Nosso programa possui três componentes principais.
- Primeiro, escreva o código da seguinte forma:



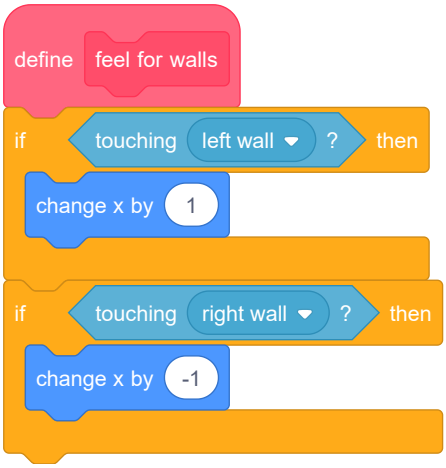
Observe que, ao clicar na bandeira verde, nosso sprite se move para o centro do palco nas coordenadas (0,0) e então fica aguardando o teclado e verificando paredes indefinidamente.

- Em segundo lugar, adicione este segundo grupo de blocos de código:



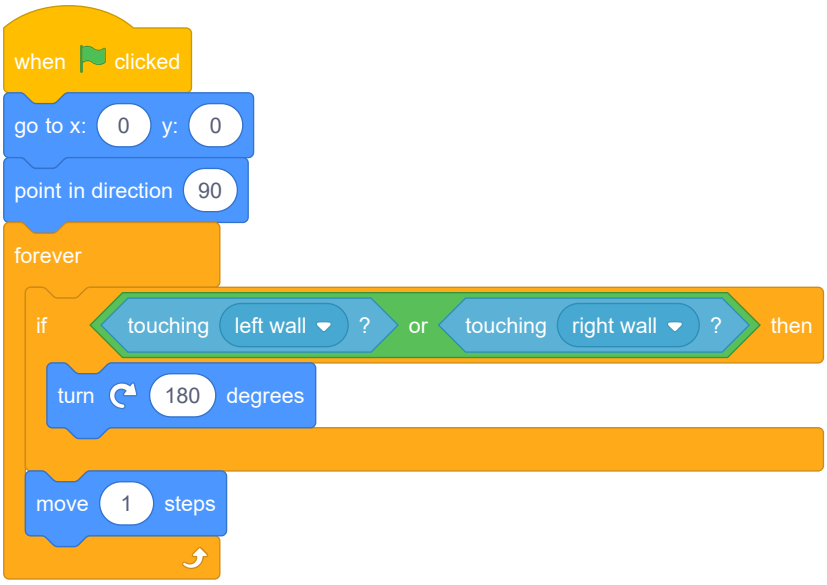
Observe como criamos um `listen for keyboard` script personalizado. Para cada uma das teclas de seta do teclado, ele moverá o sprite pela tela.

- Por fim, adicione este grupo de blocos de código:



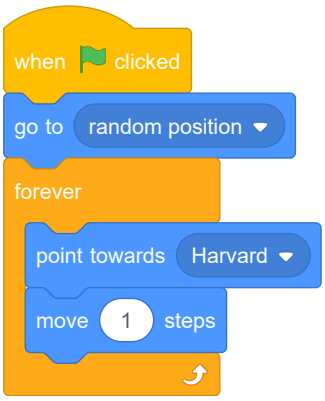
Observe que também temos um `feel for walls` script personalizado. Quando um sprite toca uma parede, ele o move de volta para uma posição segura, impedindo que ele saia da tela.

- Você pode aprender mais explorando esses blocos de código (https://scratch.mit.edu/projects/326129433) .
- O Scratch permite que vários sprites sejam exibidos na tela simultaneamente.
- Para adicionar outro sprite, inclua os seguintes blocos de código no seu programa:



Observe como o sprite de Yale parece atrapalhar o sprite de Harvard, movendo-se para frente e para trás. Quando ele bate em uma parede, ele gira até bater na parede novamente. Você pode aprender mais explorando esses blocos de código (https://scratch.mit.edu/projects/565127193) .

- Você pode até fazer um sprite seguir outro sprite. Para adicionar outro sprite, adicione os seguintes blocos de código ao seu programa:



Observe como o logotipo do MIT agora parece acompanhar o de Harvard. Você pode aprender mais explorando esses blocos de código (https://scratch.mit.edu/projects/565479840) .

- Experimente o jogo completo Ivy's Hardest Game (https://scratch.mit.edu/projects/565742837) .

Resumindo

Nesta lição, você aprendeu como este curso se encaixa no vasto mundo da ciência da computação e da programação. Você aprendeu...

- Poucos alunos chegam a esta aula com experiência prévia em programação!
- Você não está sozinho! Você faz parte de uma comunidade.
- A resolução de problemas é a essência do trabalho dos cientistas da computação.

- Este curso não se limita à programação – ele apresentará uma nova forma de aprendizado que você poderá aplicar a praticamente todas as áreas da vida.
- Como números, textos, imagens, música e vídeos são compreendidos e representados por computadores.
- A habilidade fundamental de programação conhecida como pseudocódigo.
- Formas razoáveis e não razoáveis de utilizar a IA neste curso.
- Como a abstração influenciará seu trabalho futuro neste curso.
- Os elementos básicos da programação incluem funções, condicionais, laços de repetição e variáveis.
- Como criar um projeto no Scratch.

Este foi o CS50! Bem-vindos a bordo! Até a próxima!