Esto es CS50x

OpenCourseWare

Donar (https://cs50.harvard.edu/donate) (https://community.alumni.harvard.edu/give/59206872)

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.quora.com/profile/David-J-Malan)
(https://www.reddit.com/user/davidjmalan) (https://twitter.com/davidjmalan)

Conferencia 0

Bienvenida

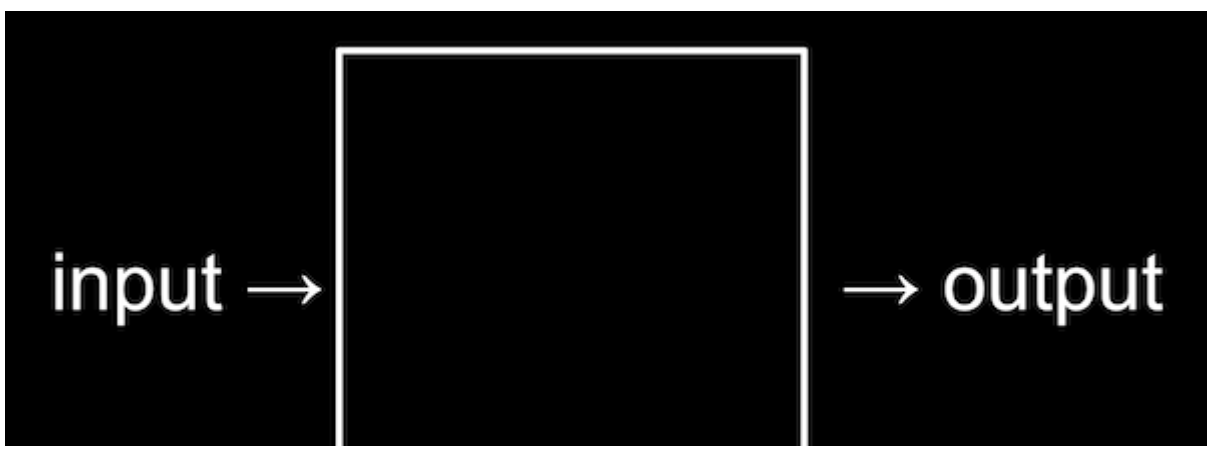
- Este año estaremos en el Loeb Drama Center (https://americanrepertorytheater.org/venue/loeb-drama-center-3/) de la Universidad de Harvard, donde, gracias a nuestra estrecha colaboración con el American Repertory Theatre (https://americanrepertorytheater.org/) , contamos con un escenario increíble e incluso utilería para demostraciones.
- Convertimos una pintura de acuarela del (https://images.hollis.harvard.edu/permalink/f/100kie6/HVD_VIAolvwork671391) siglo XVIII del campus de Harvard de (https://images.hollis.harvard.edu/permalink/f/100kie6/HVD_VIAolvwork671391) un estudiante, Jonathan Fisher, en el telón de fondo del escenario.
- Hace veinte años, cuando era estudiante, David superó su propia inquietud, salió de su zona de confort y tomó CS50 él mismo, descubriendo que el curso se trataba menos de programación que de resolución de problemas.
- De hecho, dos tercios de los estudiantes de CS50 nunca antes habían tomado un curso de informática.
- Y lo más importante también:

  > lo que en última instancia importa en este curso no es tanto dónde terminas en relación con tus compañeros de clase, sino dónde terminas en relación contigo mismo cuando comenzaste

- Comenzaremos el curso recreando un componente de un juego de Super Mario (https://en.wikipedia.org/wiki/Super_Mario_Bros.) , luego construiremos una aplicación web llamada CS50 Finance que permitirá a los usuarios comprar y vender acciones virtualmente, y terminaremos el curso con la creación de su propio proyecto final.

¿Qué es la informática?

- La informática es fundamentalmente resolución de problemas.
- Podemos pensar en la **resolución** de **problemas** como el proceso de tomar alguna entrada (detalles sobre nuestro problema) y generar alguna salida (la solución a nuestro problema). La "caja negra" en el medio es la informática, o el código que aprenderemos a escribir.



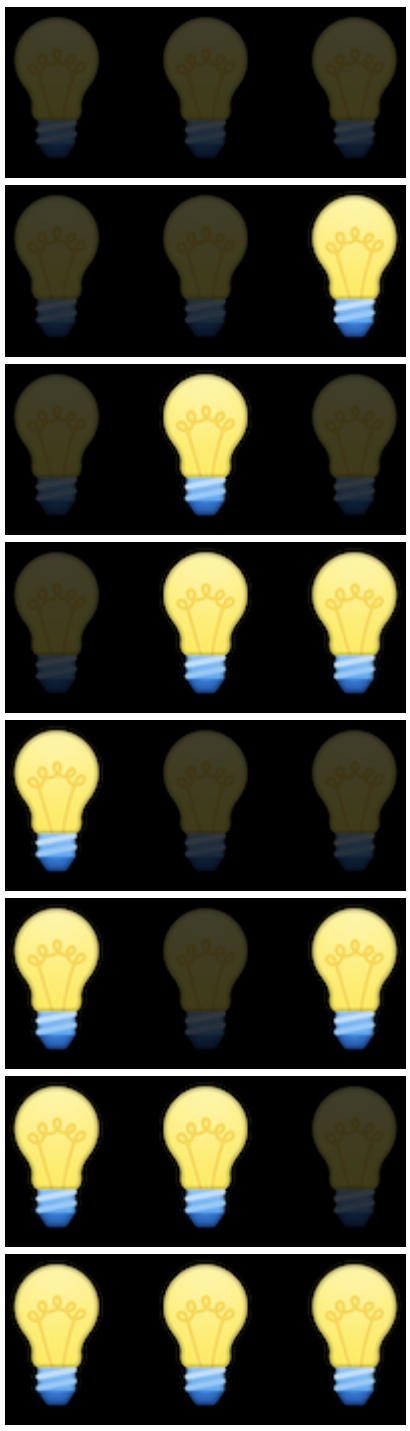
- Para comenzar a hacer eso, necesitaremos una forma de representar entradas y salidas, de modo que podamos almacenar y trabajar con información de manera estandarizada.

Representando números

- Podríamos comenzar con la tarea de tomar la asistencia contando el número de personas en una sala. Con nuestra mano, podríamos levantar un dedo a la vez para representar a cada persona, pero no podremos contar muy alto. Este sistema se llama **unario** , donde cada dígito representa un valor único de uno.
- Probablemente hemos aprendido un sistema más eficiente para representar números, donde tenemos diez dígitos, del 0 al 9:

```
0 1 2 3 4 5 6 7 8 9
```

  - Este sistema se llama decimal, o **base 10** , ya que hay diez valores diferentes que puede representar un dígito.
- Las computadoras usan un sistema más simple llamado **binario** , o base dos, con solo dos dígitos posibles, 0 y 1.
  - Cada *dígito binario* también se llama **bit** .
- Dado que las computadoras funcionan con electricidad, que se puede encender o apagar, podemos representar convenientemente un bit encendiendo o apagando algún interruptor para representar un 0 o 1.
  - Con una bombilla, por ejemplo, podemos encenderla para contar hasta 1.
- Con tres bombillas, podemos encenderlas en diferentes patrones y contar desde 0 (con las tres apagadas) hasta 7 (con las tres encendidas):



- Dentro de las computadoras modernas, no hay bombillas, sino millones de pequeños interruptores llamados **transistores** que se pueden encender y apagar para representar diferentes valores.
- Por ejemplo, sabemos que el siguiente número en decimal representa ciento veintitrés.

```
1 2 3
```

  - El `3` está en la columna de las unidades, el `2` está en la columna de las decenas y el `1` está en la columna de las centenas.
  - Así `123` es `100×1 + 10×2 + 1×3 = 100 + 20 + 3 = 123`.
  - Cada lugar para un dígito representa una potencia de diez, ya que hay diez dígitos posibles para cada lugar. El lugar más a la

derecha es para 10^0, el del medio 10^1 y el lugar más a la izquierda 10^2:

```
10 2 10 1 10 0
1 2 3
```

- En binario, con solo dos dígitos, tenemos potencias de dos para cada valor posicional:

```
2 2 2 1 2 0
# # #
```

  - Esto es equivalente a:

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

- Con todas las bombillas o interruptores apagados, aún tendríamos un valor de 0:

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

- Ahora, si cambiamos el valor binario a, digamos, `0 1 1` el valor decimal sería 3, ya que sumamos el 2 y el 1:

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

- Si tuviéramos varias bombillas más, podríamos tener un valor binario de `110010`, que tendría el valor decimal equivalente de `50`:

```
32 16 8 4 2 1
 1 1 0 0 1 0
```

  - Note eso `32 + 16 + 2 = 50`.
- Con más bits, podemos contar hasta números aún más altos.

Texto

- Para representar letras, todo lo que tenemos que hacer es decidir cómo se asignan los números a las letras. Algunos humanos, hace muchos años, se decidieron colectivamente por un mapeo estándar de números a letras. La letra "A", por ejemplo, es el número 65, y "B" es 66, y así sucesivamente. Al usar el contexto, como si estamos viendo una hoja de cálculo o un correo electrónico, diferentes programas pueden interpretar y mostrar los mismos bits como números o texto.
- El mapeo estándar, **ASCII** (https://en.wikipedia.org/wiki/ASCII) , también incluye letras minúsculas y puntuación.
- Si recibimos un mensaje de texto con un patrón de bits que tenían los valores decimales `72`, `73` y `33`, esos bits se asignan a las letras `HI!`. Cada letra se representa típicamente con un patrón de ocho bits, o un **byte** , por lo que las secuencias de bits que nos darían son `01001000`, `01001001` y `00100001`.
  - Es posible que ya estemos familiarizados con el uso de bytes como unidad de medida para datos, como en megabytes o gigabytes, para millones o miles de millones de bytes.
- Con ocho bits, o un byte, podemos tener 2^8 o 256 valores diferentes (incluido cero). (El *valor más alto hasta el* que podemos contar sería 255).
- Otros caracteres, como letras con acentos y símbolos en otros idiomas, son parte de un estándar llamado **Unicode** (https://en.wikipedia.org/wiki/Unicode) , que usa más bits que ASCII para acomodar todos estos caracteres.
  - Cuando recibimos un emoji, nuestra computadora en realidad solo recibe un número en binario que luego se asigna a la imagen del emoji según el estándar Unicode.
    - Por ejemplo, el emoji "cara con lágrimas de alegría" son solo los bits `00000001111011000000010`:

Imágenes, video, sonidos

- Una imagen, como la imagen del emoji, está formada por colores.
- Con solo bits, también podemos asignar números a colores. Hay muchos sistemas diferentes para representar colores, pero uno común es el **RGB** , que representa diferentes colores al indicar la cantidad de rojo, verde y azul dentro de cada color.
- Por ejemplo, nuestro patrón de bits anterior, `72`, `73`, y `33` podría indicar la cantidad de rojo, verde y azul en un color. (Y nuestros programas sabrían que esos bits se asignan a un color si abriéramos un archivo de imagen, en lugar de recibirlos en un mensaje de texto)

programas sabrían que esos bits se asignan a un color si abriéramos un archivo de imagen, en lugar de recibirlos en un mensaje de texto).

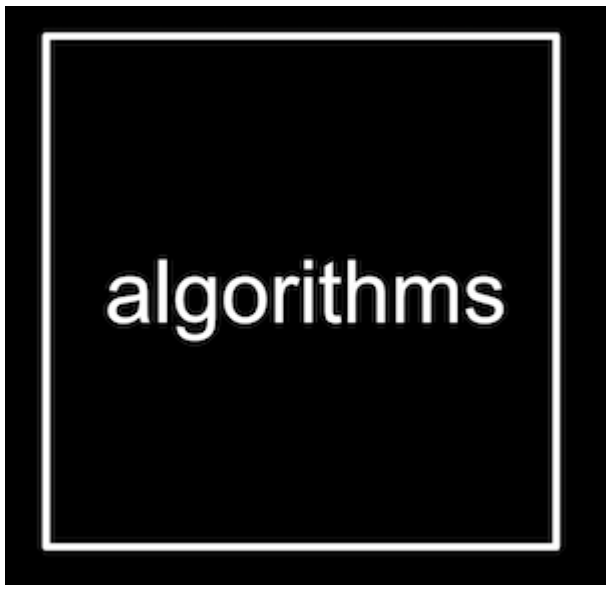
- Cada número puede ser un byte, con 256 valores posibles, por lo que con tres bytes, podemos representar millones de colores. Nuestros tres bytes de arriba representarían un tono oscuro de amarillo:

- Los puntos, o cuadrados, en nuestras pantallas se llaman **píxeles** , y las imágenes también se componen de muchos miles o millones de esos píxeles. Entonces, al usar tres bytes para representar el color de cada píxel, podemos crear imágenes. Podemos ver píxeles en un emoji si hacemos zoom, por ejemplo:

- La **resolución** de una imagen es la cantidad de píxeles que hay, horizontal y verticalmente, por lo que una imagen de alta resolución tendrá más píxeles y requerirá más bytes para almacenarse.
- Los videos se componen de muchas imágenes, que cambian varias veces por segundo para darnos la apariencia de movimiento, como lo haría un libro animado (https://youtu.be/p3q9MM__h-M) antiguo .
- La música también se puede representar con bits, con asignaciones de números a notas y duraciones, o asignaciones más complejas de bits a frecuencias de sonido en cada momento.
- Los formatos de archivo, como JPEG y PNG, o documentos de Word o Excel, también se basan en algún estándar que algunos humanos han acordado para representar la información con bits.

Algoritmos

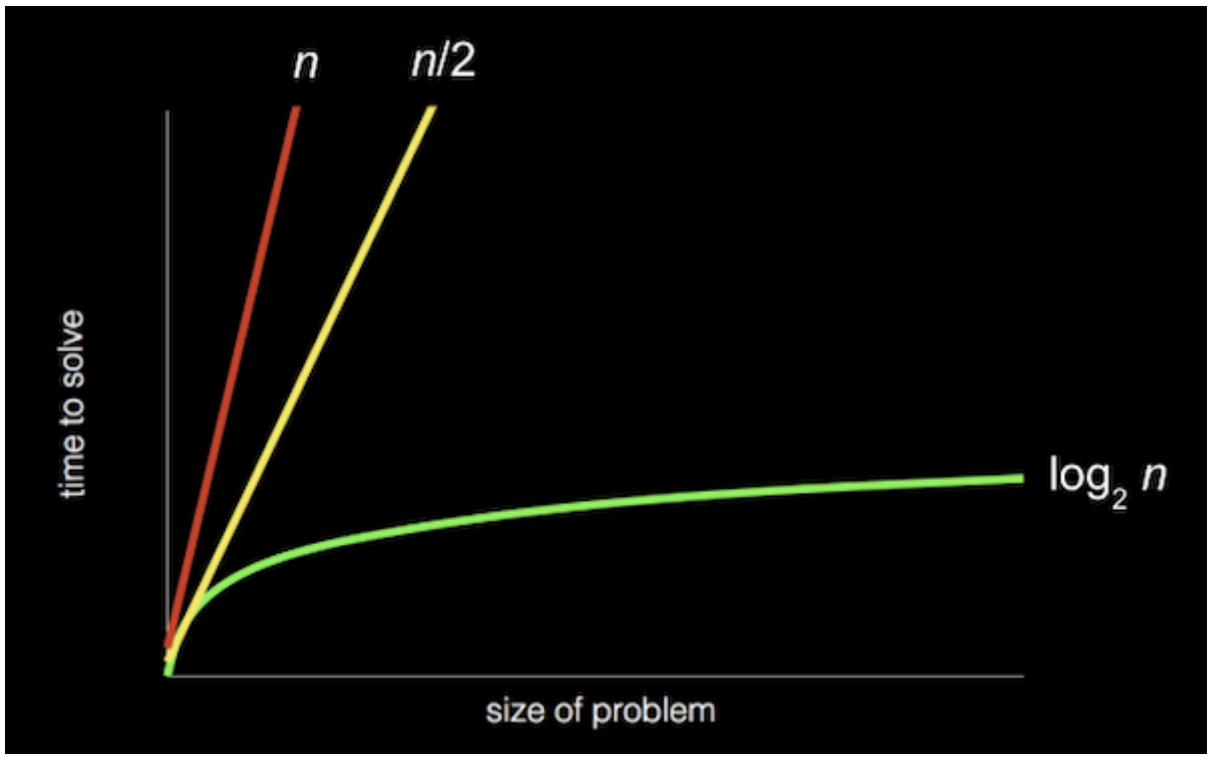
- Ahora que podemos representar entradas y salidas, podemos trabajar en la resolución de problemas. La caja negra anterior contendrá **algoritmos** , instrucciones paso a paso para resolver problemas:



- Los humanos también pueden seguir algoritmos, como recetas para cocinar. Al programar una computadora, debemos ser más precisos con nuestros algoritmos para que nuestras instrucciones no sean ambiguas o mal interpretadas.

con nuestros algoritmos para que nuestras instrucciones no sean ambiguas o mal interpretadas.

- Es posible que tengamos una aplicación en nuestros teléfonos que almacene nuestros contactos, con sus nombres y números de teléfono ordenados alfabéticamente. El equivalente de la vieja escuela podría ser una guía telefónica, una copia impresa de nombres y números de teléfono.
- Nuestra aportación al problema de encontrar el número de alguien sería la guía telefónica y un nombre a buscar. Podríamos abrir el libro y comenzar desde la primera página, buscando un nombre una página a la vez. Este algoritmo sería **correcto** , ya que eventualmente encontraremos el nombre si está en el libro.
- Podríamos hojear el libro dos páginas a la vez, pero este algoritmo no será correcto ya que podríamos omitir la página con nuestro nombre. Podemos corregir este **error** , o error, retrocediendo una página si pasamos demasiado, ya que sabemos que la guía telefónica está ordenada alfabéticamente.
- Otro algoritmo sería abrir la guía telefónica por la mitad, decidir si nuestro nombre estará en la mitad izquierda o derecha del libro (porque el libro está en orden alfabético) y reducir el tamaño de nuestro problema a la mitad. Podemos repetir esto hasta que encontremos nuestro nombre, dividiendo el problema por la mitad cada vez. Con 1024 páginas para comenzar, solo necesitaríamos 10 pasos para dividir por la mitad antes de que solo nos quede una página para verificar. Podemos ver esto visualizado en una animación de dividir una guía telefónica por la mitad repetidamente (https://youtu.be/F5LZhsekEBc) , en comparación con la animación de buscar una página a la vez (https://youtu.be/-yTRajiUi5s) .
- De hecho, podemos representar la eficiencia de cada uno de esos algoritmos con un gráfico:



- Nuestra primera solución, buscar una página a la vez, se puede representar con la línea roja: nuestro tiempo para resolver aumenta linealmente a medida que aumenta el tamaño del problema. *n* es un número que representa el tamaño del problema, por lo que con *n* páginas en nuestra guía telefónica, tenemos que realizar hasta *n* pasos para encontrar un nombre.
- La segunda solución, buscar dos páginas a la vez, se puede representar con la línea amarilla: nuestra pendiente es menos empinada, pero sigue siendo lineal. Ahora, solo necesitamos (aproximadamente) *n* / 2 pasos, ya que pasamos dos páginas a la vez.
- Nuestra solución final, dividiendo la guía telefónica por la mitad cada vez, puede ser representada por la línea verde, con una relación fundamentalmente diferente entre el tamaño del problema y el tiempo para resolverlo: **logarítmica** (https://en.wikipedia.org/wiki/Logarithm) , ya que nuestro tiempo para resolver aumenta cada vez más. lentamente a medida que aumenta el tamaño del problema. En otras palabras, si la guía telefónica pasara de 1000 a 2000 páginas, solo necesitaríamos un paso más para encontrar nuestro nombre. Si el tamaño se duplicara nuevamente de 2000 a 4000 páginas, solo necesitaríamos un paso más. La línea verde está etiquetada como $\log_2 n$, o log base 2 de *n* , ya que dividimos el problema por dos con cada paso.
- Cuando escribimos programas usando algoritmos, generalmente nos preocupamos no solo de cuán correctos son, sino de cuán **bien diseñados** están, considerando factores como la eficiencia.

Pseudocódigo

- Podemos escribir **pseudocódigo** , que es una representación de nuestro algoritmo en inglés preciso (o en algún otro lenguaje humano):

```
1 Recoger directorio telefónico
2 Abrir hasta la mitad de la guía telefónica
3 Mira la página
4 Si la persona está en la página
5 Llamar a la persona
6 De lo contrario, si la persona está antes en el libro
7 Abierto hasta la mitad de la mitad izquierda del libro
8 Regrese a la línea 3
9 De lo contrario, si la persona está más tarde en el libro
10 Abierto hasta la mitad de la mitad derecha del libro
11 Regrese a la línea 3
12 más
```

```
13 Salir
```

- Con estos pasos, revisamos la página del medio, decidimos qué hacer y repetimos. Si la persona no está en la página y no quedan más páginas en el libro, entonces nos detenemos. Y ese último caso es particularmente importante de recordar. Cuando otros programas en nuestras computadoras olvidaron ese caso final, es posible que parezca que se congelan o dejan de responder, ya que se han encontrado con un caso que no se tuvo en cuenta, o continúan repitiendo el mismo trabajo una y otra vez entre bastidores sin hacer nada. Progreso.

- Algunas de estas líneas comienzan con verbos o acciones. Comenzaremos a llamar a estas *funciones* :

```
1   Recoger directorio telefónico
2   Abrir hasta la mitad de la guía telefónica
3   Mira la página
4 Si la persona está en la página
5       Llamar a la persona
6 De lo contrario, si la persona está antes en el libro
7       Abierto hasta la mitad de la mitad izquierda del libro
8 Regrese a la línea 3
9 De lo contrario, si la persona está más tarde en el libro
10      Abierto hasta la mitad de la mitad derecha del libro
11 Regrese a la línea 3
12 más
13      Salir
```

- También tenemos ramas que conducen a diferentes caminos, como bifurcaciones en la carretera, a las que llamaremos *condiciones* :

```
1 Recoger directorio telefónico
2 Abrir hasta la mitad de la guía telefónica
3 Mira la página
4   Si la persona está en la página
5 Llamar a la persona
6 De lo   contrario, si la persona está antes en el libro
7 Abierto hasta la mitad de la mitad izquierda del libro
8 Regrese a la línea 3
9 De lo   contrario, si la persona está más tarde en el libro
10 Abierto hasta la mitad de la mitad derecha del libro
11 Regrese a la línea 3
12 más
13 Salir
```

- Y las preguntas que deciden hacia dónde vamos se llaman *expresiones booleanas* , que eventualmente dan como resultado un valor de sí o no, o verdadero o falso:

```
1 Recoger directorio telefónico
2 Abrir hasta la mitad de la guía telefónica
3 Mira la página
4 Si la persona está en la página
5 Llamar a la persona
6 De lo contrario, si la persona está antes en el libro
7 Abierto hasta la mitad de la mitad izquierda del libro
8 Regrese a la línea 3
9 De lo contrario, si la persona está más tarde en el libro
10 Abierto hasta la mitad de la mitad derecha del libro
11 Regrese a la línea 3
12 más
13 Salir
```

- Por último, tenemos palabras que crean ciclos, donde podemos repetir partes de nuestro programa, llamadas *bucles* :

```
1 Recoger directorio telefónico
2 Abrir hasta la mitad de la guía telefónica
3 Mira la página
4 Si la persona está en la página
5 Llamar a la persona
6 De lo contrario, si la persona está antes en el libro
7 Abierto hasta la mitad de la mitad izquierda del libro
```

```
7 Abierto hasta la mitad de la mitad izquierda del libro
8       Regrese a la línea 3
9 De lo contrario, si la persona está más tarde en el libro
10 Abierto hasta la mitad de la mitad derecha del libro
11      Regrese a la línea 3
12 más
13 Salir
```

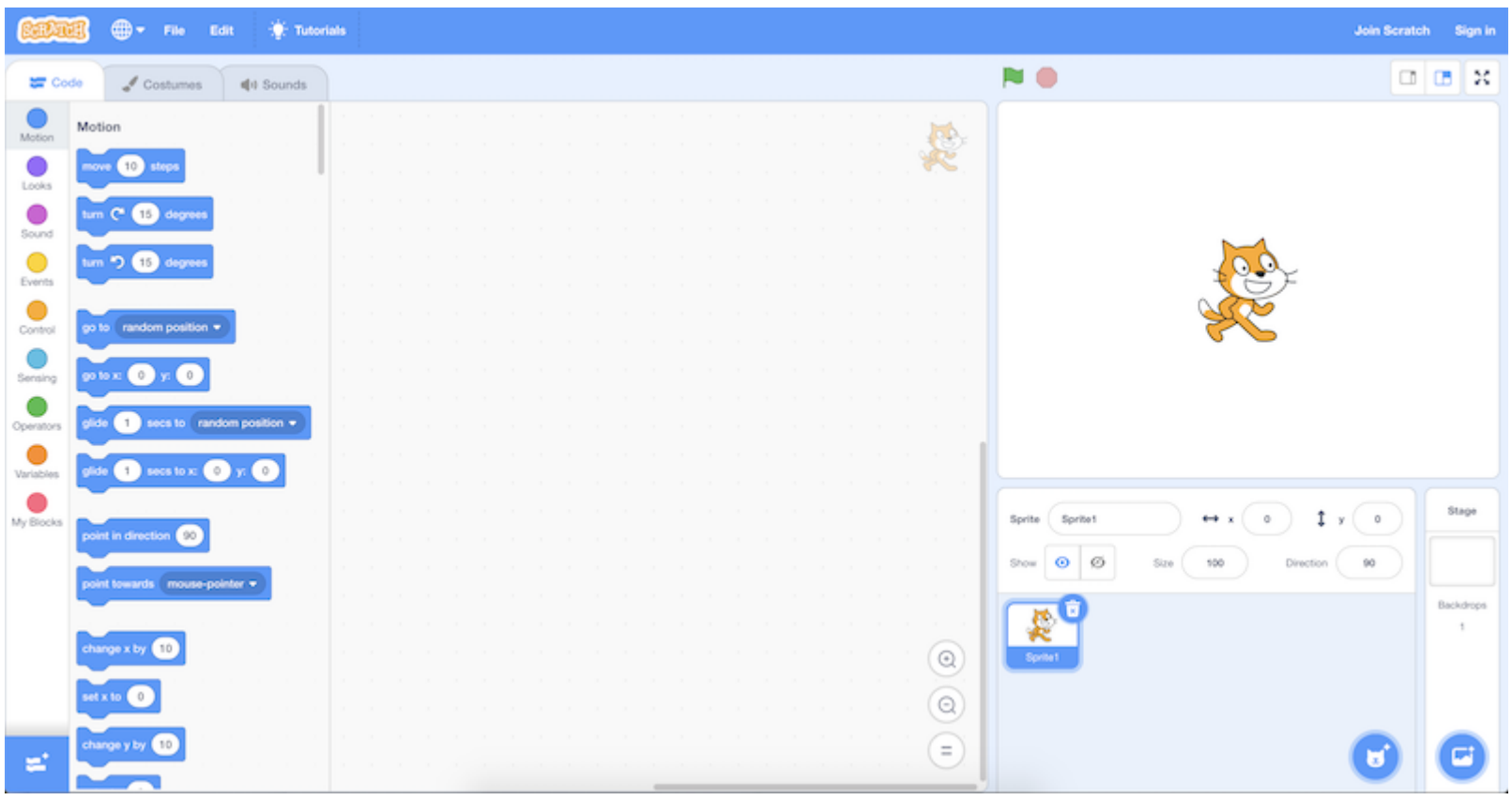
Rasguño

- Podemos escribir programas con los bloques de construcción que acabamos de descubrir:
  - funciones
  - condiciones
  - Expresiones booleanas
  - bucles
- Y descubriremos características adicionales que incluyen:
  - variables
  - hilos
  - eventos
  - ...
- Antes de aprender a usar un lenguaje de programación basado en texto llamado C, usaremos un lenguaje de programación gráfico llamado Scratch (https://scratch.mit.edu/) , donde arrastraremos y soltaremos bloques que contienen instrucciones.
- Un programa simple en C que imprima "hola, mundo", se vería así:

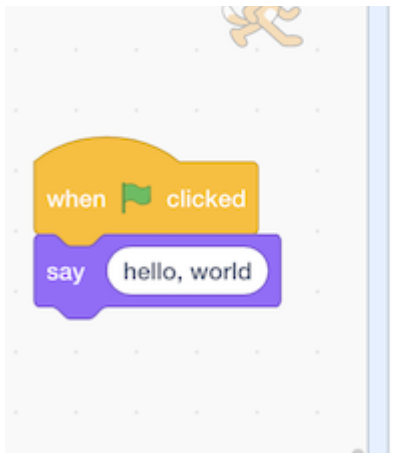
```c
#include <stdio.h>

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

  - Hay muchos símbolos y sintaxis, o disposición de estos símbolos, que tendríamos que averiguar.
- El entorno de programación de Scratch es un poco más amigable:



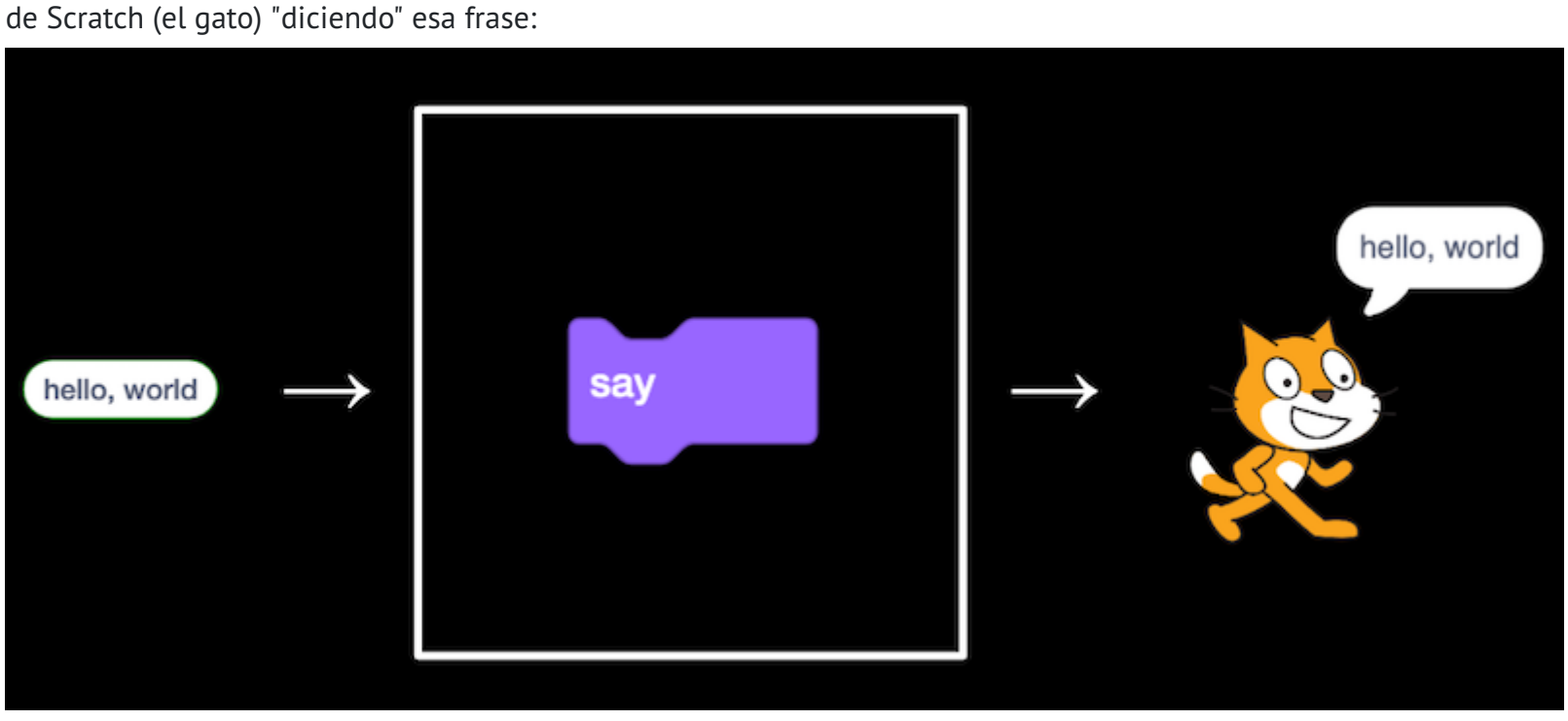
  - En la parte superior derecha, tenemos un escenario que será mostrado por nuestro programa, donde podemos agregar o cambiar fondos, personajes (llamados sprites en Scratch), y más.
  - A la izquierda, tenemos piezas de rompecabezas que representan funciones o variables, u otros conceptos, que podemos arrastrar y soltar en nuestra área de instrucción en el centro.
  - En la parte inferior derecha, podemos agregar más caracteres para que los use nuestro programa.
- Podemos arrastrar algunos bloques para que Scratch diga "hola, mundo":



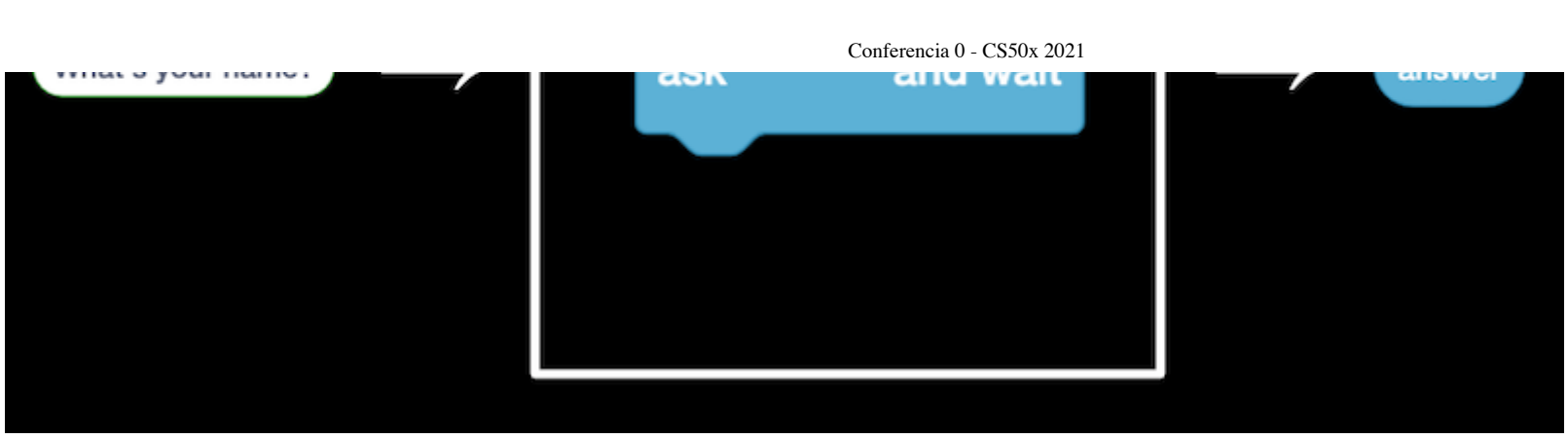
  - El bloque "cuando se hace clic en la bandera verde" se refiere al inicio de nuestro programa (ya que hay una bandera verde sobre el escenario que podemos usar para iniciarlo), y debajo de él hemos insertado un bloque "decir" y escrito "Hola Mundo". Y podemos averiguar qué hacen estos bloques explorando la interfaz y experimentando.
- También podemos arrastrar el bloque "preguntar y esperar", con una pregunta como "¿Cómo te llamas?", Y combinarlo con un bloque "decir" para la respuesta:

  - El bloque "respuesta" es una variable, o valor, que almacena lo que escribe el usuario del programa, y podemos colocarlo en un bloque "decir" arrastrando y soltando también.
- Pero no esperamos después de decir "Hola" con el primer bloque, por lo que podemos usar el bloque "unirse" para combinar dos frases para que nuestro gato pueda decir "hola, David":

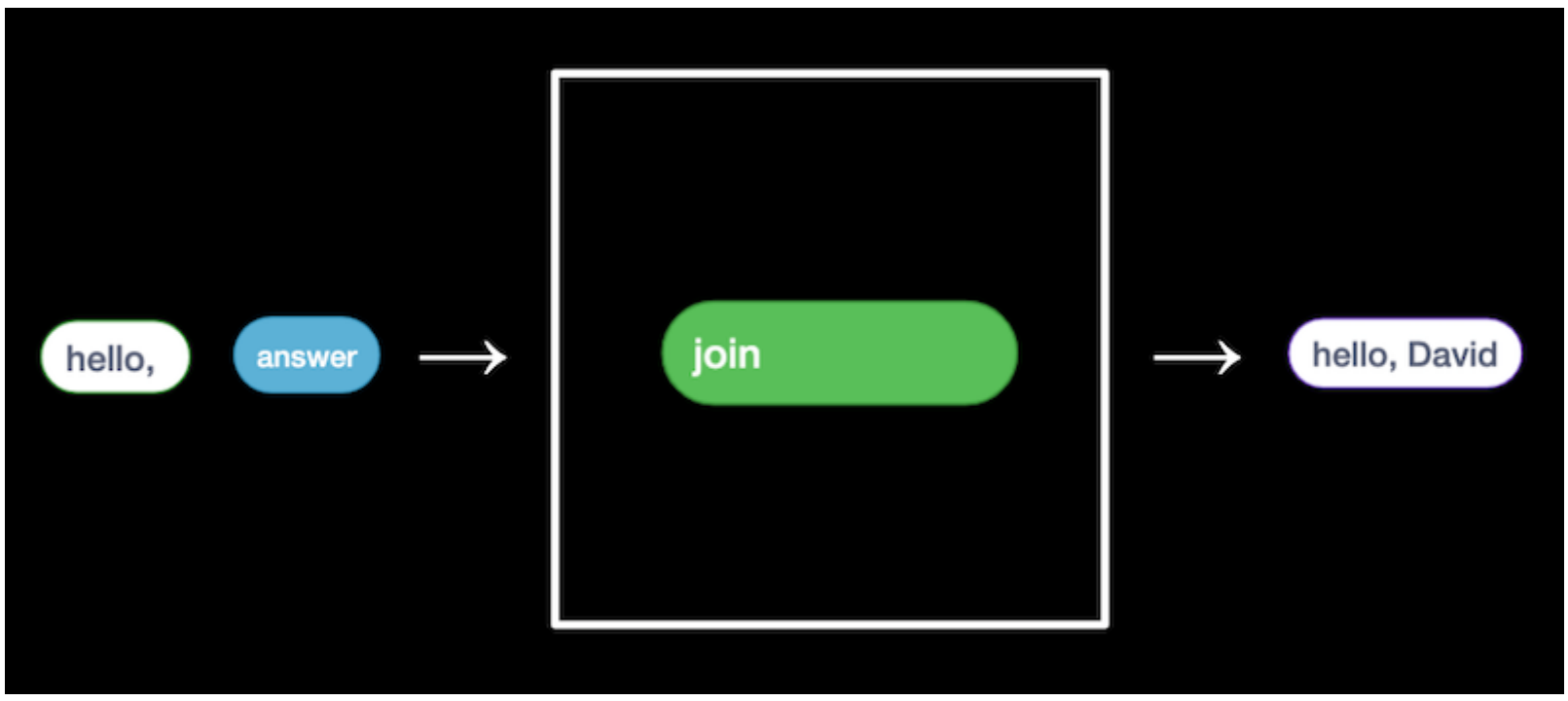
  - Cuando intentamos anidar bloques, o colocarlos uno dentro del otro, Scratch nos ayudará expandiendo los lugares donde se pueden usar.
- De hecho, el bloque "decir" en sí mismo es como un algoritmo, donde proporcionamos una entrada de "hola, mundo" y produjo la salida de Scratch (el gato) "diciendo" esa frase:



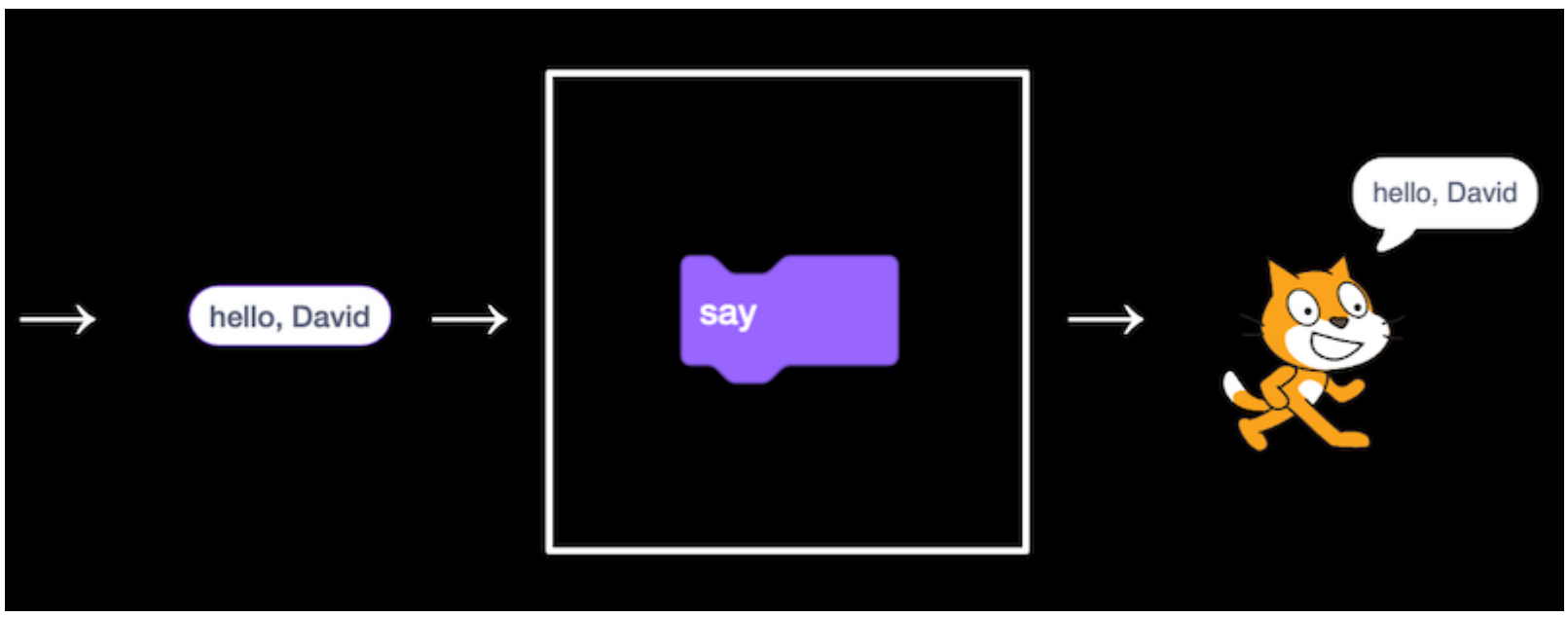
- El bloque "preguntar" también toma una entrada (la pregunta que queremos hacer) y produce la salida del bloque "respuesta":



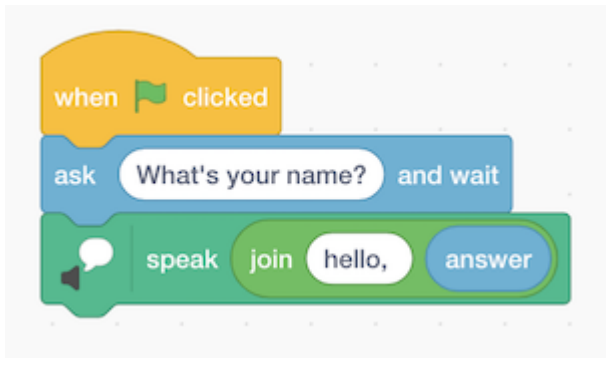
- Luego podemos usar el bloque "respuesta" junto con nuestro propio texto, "hola", como dos entradas para el algoritmo de unión ...



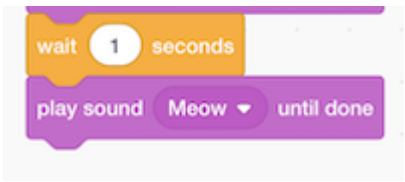
- ... cuya salida pasamos can como entrada al bloque "decir":



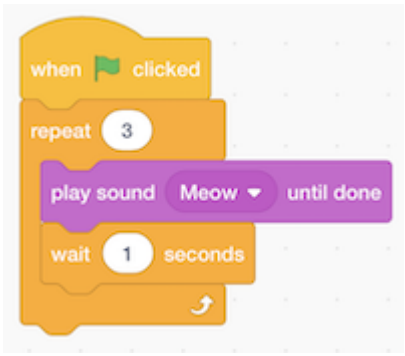
- En la parte inferior izquierda de la pantalla, vemos un icono de extensiones, y uno de ellos se llama Text to Speech. Después de agregarlo, podemos usar el bloque "hablar" para escuchar a nuestro gato hablar:



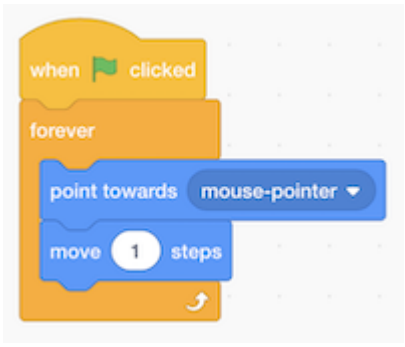
- La extensión Text to Speech, gracias a la nube, o servidores informáticos en Internet, está convirtiendo nuestro texto en audio.
- Podemos intentar hacer que el gato diga miau:



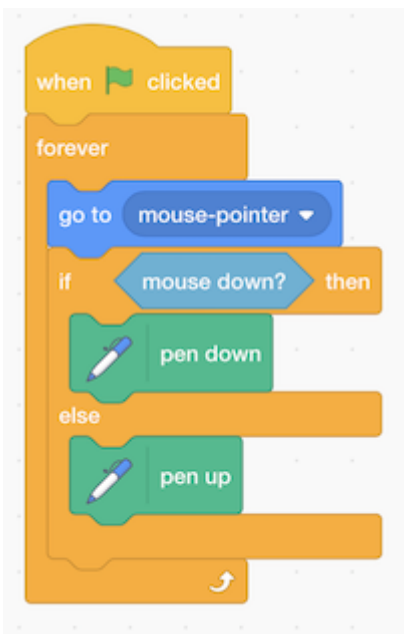
  - Podemos hacer que diga miau tres veces, pero ahora estamos repitiendo bloques una y otra vez.
- Usemos un bucle o un bloque de "repetición":



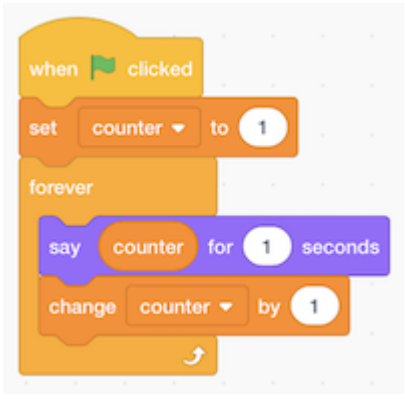
  - Ahora nuestro programa logra los mismos resultados, pero con menos bloques. Podemos considerar que tiene un mejor diseño: si hay algo que quisiéramos cambiar, solo necesitaríamos cambiarlo en un lugar en lugar de tres.
- Podemos hacer que el gato apunte hacia el mouse y se mueva hacia él:



- Probamos la extensión Pen, usando el bloque "pen down" con una condición:



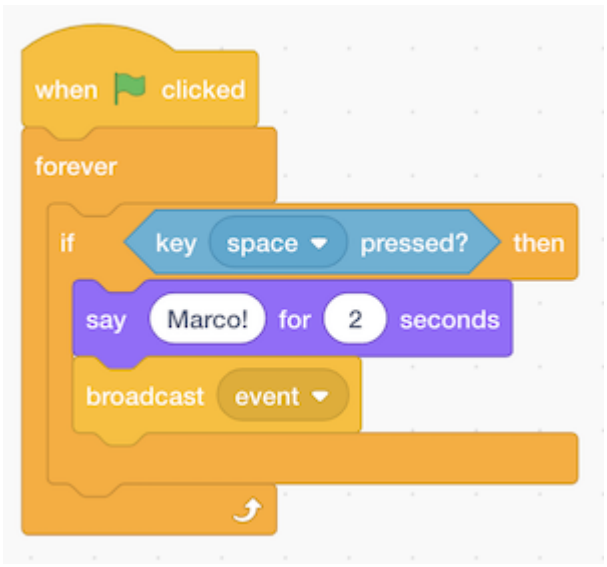
  - Aquí, movemos el gato al puntero del mouse, y si se hace clic en el mouse, o hacia abajo, colocamos el “bolígrafo”, que dibuja. De lo contrario, levantamos el bolígrafo. Repetimos esto muy rápido, una y otra vez, por lo que terminamos con el efecto de dibujar cada vez que mantenemos presionado el mouse.
- Scratch también tiene diferentes disfraces, o imágenes, que podemos usar para nuestros personajes.
- Haremos un programa que pueda contar:



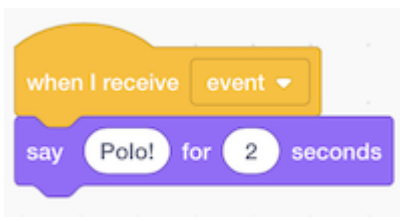
  - Aquí, `counter` hay una variable, cuyo valor podemos establecer, usar y cambiar.
- Observamos algunos programas más, como rebote (https://scratch.mit.edu/projects/277536611/editor/) , donde el gato se mueve hacia adelante y hacia atrás en la pantalla para siempre, girando cada vez que estamos en el borde de la pantalla.
  - Podemos mejorar la animación haciendo que el gato cambie a un disfraz (https://scratch.mit.edu/projects/277536630/editor/) diferente después de cada 10 pasos en bounce1 (https://scratch.mit.edu/projects/277536630/editor/) . Ahora cuando hacemos clic en la bandera verde para ejecutar nuestro programa, vemos que el gato alterna el movimiento de sus patas.
- Incluso podemos grabar nuestros propios sonidos con el micrófono de nuestra computadora y reproducirlos en nuestro programa.
- Para crear programas cada vez más complejos, comenzamos con cada una de estas características más simples y las colocamos una encima de la otra.

- También podemos tener Scratch miau si lo tocamos con el puntero del mouse, en pet0 (https://scratch.mit.edu/projects/277537223/editor/) .
- En bark (https://scratch.mit.edu/projects/326130490/editor/) , no tenemos uno, sino dos programas en el mismo proyecto Scratch. Ambos programas se ejecutarán al mismo tiempo después de hacer clic en la bandera verde. Uno de ellos reproducirá un sonido de león marino si la `muted` variable está configurada en `false` , y el otro configurará la `muted` variable de `true` a `false` , o `false` a `true` , si se presiona la tecla de espacio.
- Otra extensión mira el video capturado por la cámara web de nuestra computadora y reproduce el sonido del maullido si el video tiene movimiento por encima de algún umbral.

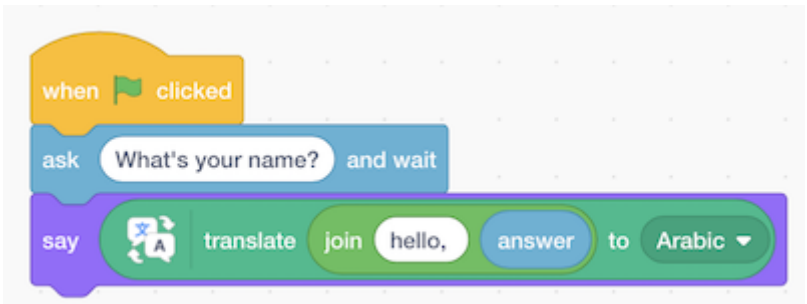
- Con múltiples sprites, o personajes, podemos tener diferentes conjuntos de bloques para cada uno de ellos:



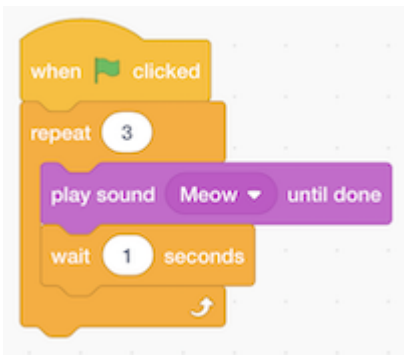
  - Para una marioneta, tenemos estos bloques que dicen "¡Marco!", Y luego un bloque de "evento de transmisión". Este "evento" se utiliza para que nuestros dos sprites se comuniquen entre sí, como enviar un mensaje detrás de escena. Así que nuestro otro títere puede esperar a que este "evento" diga "¡Polo!":



- También podemos usar la extensión Translate para decir algo en otros idiomas:



  - Aquí, la salida del bloque "join" se utiliza como entrada al bloque "translate", cuya salida se pasa como entrada al bloque "say".
- Ahora que conocemos algunos conceptos básicos, podemos pensar en el diseño o la calidad de nuestros programas. Por ejemplo, podríamos querer que el gato maulle tres veces con el bloque "repetir":



- Podemos usar la **abstracción** , lo que simplifica un concepto más complejo. En este caso, podemos definir nuestro propio bloque "miau" en Scratch, y reutilizarlo en otro lugar de nuestro programa, como se ve en meow3 (https://scratch.mit.edu/projects/421542702/editor/) . La ventaja es que no necesitamos saber cómo se implementa el maullido o cómo se escribe en código, sino más bien usarlo en nuestro programa, haciéndolo más legible.
- Incluso podemos definir un bloque con una entrada en miau4 (https://scratch.mit.edu/projects/421543064/editor/) , donde tenemos un bloque que hace que el gato maúlle un cierto número de veces. Ahora podemos reutilizar ese bloque en nuestro programa para maullar cualquier cantidad de veces, muy parecido a cómo podemos usar los bloques "traducir" o "hablar", sin conocer los **detalles de implementación** , o cómo funciona realmente el bloque.
- Echamos (https://scratch.mit.edu/projects/277537196/) un vistazo a algunas demostraciones más, incluida la remezcla de cuentos de

- Echamos (https://scratch.mit.edu/projects/277537196/) un vistazo a algunas demostraciones más, incluida la remezcla de cuentos de jengibre (https://scratch.mit.edu/projects/277536784/) y Oscartime (https://scratch.mit.edu/projects/277537196/) , que combinan bucles, condiciones y movimiento para crear un juego interactivo.
- Oscartime fue creado por David hace muchos años, y comenzó agregando un sprite, luego una característica a la vez, y así sucesivamente, hasta que se sumaron al programa más complicado.
- Un ex alumno, Andrew, creó Raining Men (https://scratch.mit.edu/projects/37412/) . Aunque Andrew finalmente terminó sin dedicarse a la informática como profesión, las habilidades para la resolución de problemas, los algoritmos y las ideas que aprenderemos en el curso son aplicables en todas partes.
- ¡Hasta la proxima vez!