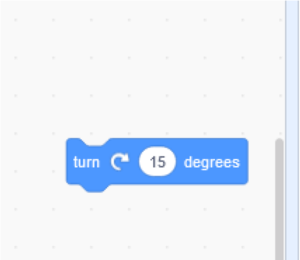
What is a script

July 24, 2022

Effects

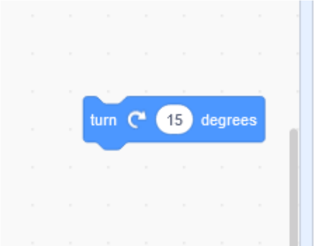
- You have seen commands

A command



turn ↻ 15 degrees

A command



```
turn ↻ 15 degrees
```

Effects

- *Commands*

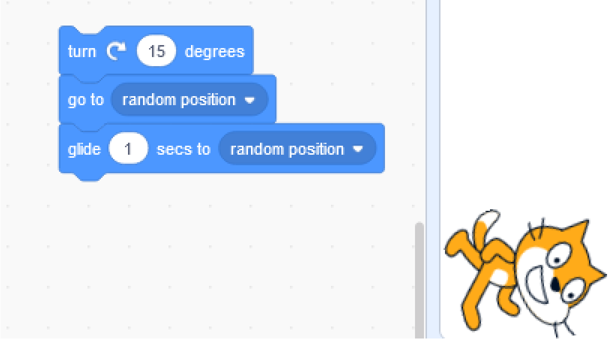
Effects

- *Commands* (typically) cause or request *effects*

Effects

- *Commands* (typically) cause or request *effects*
- *Scripts* are (at a basic level) lists of commands

A script



```
turn ↻ 15 degrees
go to random position ▾
glide 1 secs to random position ▾
```

The end

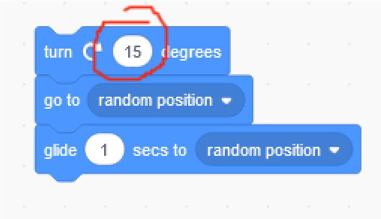
The end

Parameters

- Some commands have different effects for different inputs

Parameters

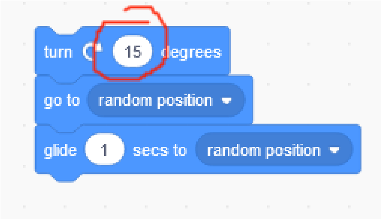
- Some commands have different effects for different inputs



```
turn ↻ 15 degrees
go to random position ▾
glide 1 secs to random position ▾
```

Parameters

- Some commands have different effects for different inputs



```
turn ⟳ 15 degrees
go to random position ▾
glide 1 secs to random position ▾
```

- Such an input is called a *parameter*

Parameters

- Scratch has three *types*:
  - *Integers* (e.g., -2, -1, 0, 1, 2. . .)

Parameters

- Scratch has three *types*:
    - *Integers* (e.g., -2, -1, 0, 1, 2. . .)
    - *Booleans* (True/False)

Parameters

- Scratch has three *types*:
    - *Integers* (e.g., -2, -1, 0, 1, 2. . .)
    - *Booleans* (True/False)
    - *Strings* (“Hello”, “world”, “a line of text”)
- Strings are simply text (lists of characters)

Parameters

- Scratch has three *types*:
    - *Integers* (e.g., -2, -1, 0, 1, 2. . .)
    - *Booleans* (True/False)
    - *Strings* ("Hello", "world", "a line of text")
- Strings are simply text (lists of characters)
- Integers indicate quantities
    - How much, how far, how often?

Parameters

- Scratch has three *types*:
  - *Integers* (e.g., -2, -1, 0, 1, 2...)
  - *Booleans* (True/False)
  - *Strings* ("Hello", "world", "a line of text")
- Strings are simply text (lists of characters)
- Integers indicate quantities
  - How much, how far, how often?
- Booleans indicate truth or success
  - Is $1 + 2 == 3$, is the password correct, should this command be run?

Operators

- *Values* can be combined with *operators*

Operators

- *Values* can be combined with *operators*
- For example, integer operators are $+, -, *, / \ldots$

Operators

- *Values* can be combined with *operators*
- For example, integer operators are $+, -, *, /\dots$
  - Operators can go between different types

Operators

- *Values* can be combined with *operators*
- For example, integer operators are $+, -, *, / \ldots$
  - Operators can go between different types
- The *comparison* operators are $==, <, =<, >, >=$

Operators

- *Values* can be combined with *operators*
- For example, integer operators are $+, -, *, /\ldots$
  - Operators can go between different types
- The *comparison* operators are $==, <, =<, >, >=$
  - e.g., $10 < 20$ is True.

Operators

- *Values* can be combined with *operators*
- For example, integer operators are $+, -, *, / \ldots$
  - Operators can go between different types
- The *comparison* operators are $==, <, =<, >, >=$
  - e.g., $10 < 20$ is True.
- Boolean operators can combine results of multiple tests

Operators

- *Values* can be combined with *operators*
- For example, integer operators are $+, -, *, / \ldots$
  - Operators can go between different types
- The *comparison* operators are $==, <, =<, >, >=$
  - e.g., $10 < 20$ is True.

- Boolean operators can combine results of multiple tests
  - does this person fit in a rollercoaster?

Operators

- *Values* can be combined with *operators*
- For example, integer operators are $+, -, *, / \dots$
  - Operators can go between different types
- The *comparison* operators are $==, <, =<, >, >=$
  - e.g., $10 < 20$ is True.
- Boolean operators can combine results of multiple tests
  - does this person fit in a rollercoaster?
    - `(150 < length) and (length < 210)`

Operators

- *Values* can be combined with *operators*
- For example, integer operators are $+, -, *, / \ldots$
  - Operators can go between different types
- The *comparison* operators are $==, <, =<, >, >=$
  - e.g., $10 < 20$ is True.

- Boolean operators can combine results of multiple tests
  - does this person fit in a rollercoaster?
    - `(150 < length) and (length < 210)`
  - is this person eligible for a discount at the museum?

Operators

- *Values* can be combined with *operators*
- For example, integer operators are $+, -, *, / \ldots$
  - Operators can go between different types
- The *comparison* operators are $==, <, =<, >, >=$
  - e.g., $10 < 20$ is True.

- Boolean operators can combine results of multiple tests
  - does this person fit in a rollercoaster?
    - `(150 < length) and (length < 210)`
  - is this person eligible for a discount at the museum?
    - `(age < 17) or (65 < age)`

Operators

- *Values* can be combined with *operators*
- For example, integer operators are $+, -, *, / \ldots$
  - Operators can go between different types
- The *comparison* operators are $==, <, =<, >, >=$
  - e.g., $10 < 20$ is True.
- Boolean operators can combine results of multiple tests
  - does this person fit in a rollercoaster?
    - `(150 < length) and (length < 210)`
  - is this person eligible for a discount at the museum?
    - `(age < 17) or (65 < age)`
- `and`/`or` are written in many ways. . .

Operators

- *Values* can be combined with *operators*
- For example, integer operators are $+, -, *, / \ldots$
  - Operators can go between different types
- The *comparison* operators are $==, <, =<, >, >=$
  - e.g., $10 < 20$ is True.
- Boolean operators can combine results of multiple tests
  - does this person fit in a rollercoaster?
    - `(150 < length) and (length < 210)`
  - is this person eligible for a discount at the museum?
    - `(age < 17) or (65 < age)`
- `and`/`or` are written in many ways...
- Math: $\wedge$, $\vee$. Python: `and`, `or`. Java: `&&`, `||`.

Variables

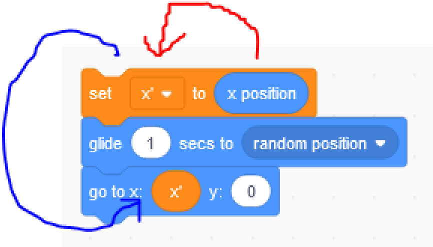
- Need to store a value for later?

Variables

- Need to store a value for later?
- You can assign values to *variables*, and reference those later

Variables

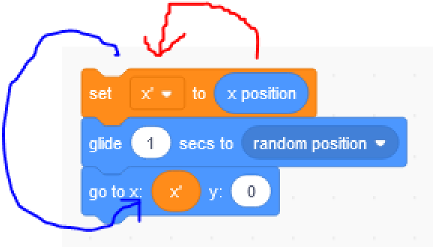
- Need to store a value for later?
- You can assign values to *variables*, and reference those later



```
set x' to x position
glide 1 secs to random position
go to x: x' y: 0
```

Variables

- Need to store a value for later?
- You can assign values to *variables*, and reference those later

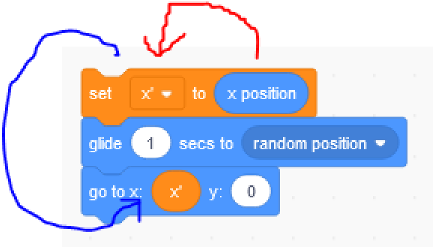


```
set x' to x position
glide 1 secs to random position
go to x: x' y: 0
```

- Variables can also be used as parameters

Variables

- Need to store a value for later?
- You can assign values to *variables*, and reference those later

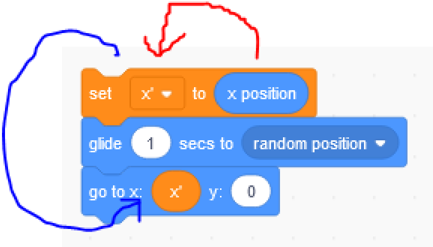


```
set x' to x position
glide 1 secs to random position
go to x: x' y: 0
```

- Variables can also be used as parameters
- Or in any place a value fits

Variables

- Need to store a value for later?
- You can assign values to *variables*, and reference those later



```
set x' to x position
glide 1 secs to random position
go to x: x' y: 0
```

- Variables can also be used as parameters
- Or in any place a value fits
- What does `x := 0; x := x + 1` do?

Control flow

- Sometimes, a script does something different based on a *condition*

Control flow

- Sometimes, a script does something different based on a *condition*
  - Apply a discount if `(age < 17) or (65 < age)`

Control flow

- Sometimes, a script does something different based on a *condition*
    - Apply a discount if `(age < 17) or (65 < age)`
- We say that the *control flow* depends on age

Control flow

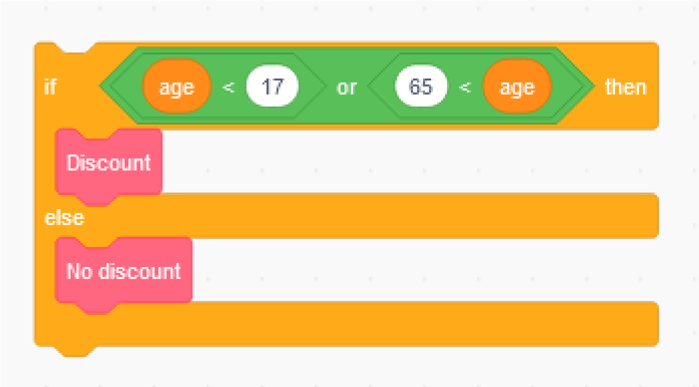
- Sometimes, a script does something different based on a *condition*
    - Apply a discount if `(age < 17) or (65 < age)`
- We say that the *control flow* depends on age
- A statement of the form `if x then y else z` is (clearly) called an *if-then-else statement*

Control flow

- Sometimes, a script does something different based on a *condition*
  - Apply a discount if `(age < 17) or (65 < age)`
- We say that the *control flow* depends on age

- A statement of the form `if x then y else z` is (clearly) called an *if-then-else statement*
- alternative: *conditional*

A conditional



```
if age < 17 or 65 < age then
  Discount
else
  No discount
```

In practice

- A program needs to *interact* with something external

In practice

- A program needs to *interact* with something external
  - otherwise, you would not notice when it executes

In practice

- A program needs to *interact* with something external
  - otherwise, you would not notice when it executes
- Such interactions are known as *IO* (input/output)

In practice

- A program needs to *interact* with something external
  - otherwise, you would not notice when it executes
- Such interactions are known as *IO* (input/output)
  - Ask the user for text input

In practice

- A program needs to *interact* with something external
  - otherwise, you would not notice when it executes
- Such interactions are known as *IO* (input/output)
  - Ask the user for text input
  - Write text to the screen

In practice

- A program needs to *interact* with something external
  - otherwise, you would not notice when it executes
- Such interactions are known as *IO* (input/output)
  - Ask the user for text input
  - Write text to the screen
  - Modify a file

In practice

- A program needs to *interact* with something external
  - otherwise, you would not notice when it executes
- Such interactions are known as *IO* (input/output)
  - Ask the user for text input
  - Write text to the screen
  - Modify a file
- IO also includes some other things

In practice

- A program needs to *interact* with something external
  - otherwise, you would not notice when it executes
- Such interactions are known as *IO* (input/output)
  - Ask the user for text input
  - Write text to the screen
  - Modify a file
- IO also includes some other things
  - Generate a random number

In practice

- A program needs to *interact* with something external
  - otherwise, you would not notice when it executes
- Such interactions are known as *IO* (input/output)
  - Ask the user for text input
  - Write text to the screen
  - Modify a file
- IO also includes some other things
  - Generate a random number
  - Get the measurements of the screen

Practical

- You are now ready to
  - test codewords (insecurely)
  - use bounding boxes
- in Scratch

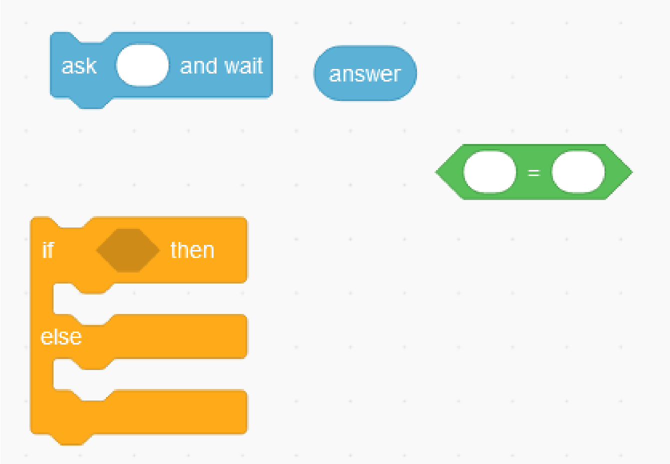
Codewords

Think of a (not very) secret word

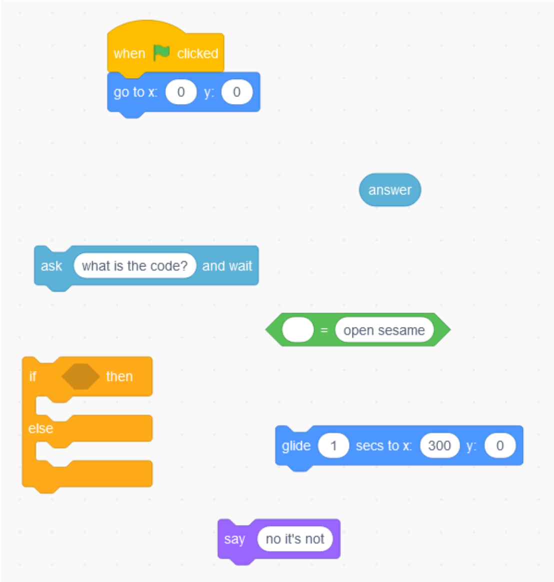
Write a program that

- asks the user for the codeword (“what is the codeword?”)
- tests if the input is equal to the the secret word
- if it is
  - perform an effect to indicate success
- otherwise
  - perform an effect to indicate failure

Hint 1



Hint 2



when 🏁 clicked

go to x: 0 y: 0

answer

ask what is the code? and wait

( ) = open sesame

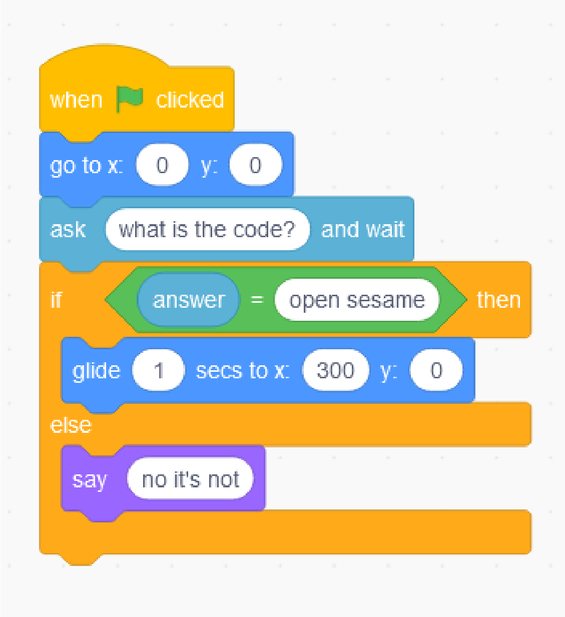
if ◆ then

else

glide 1 secs to x: 300 y: 0

say no it's not

Sample solution



```
when 🏁 clicked
go to x: 0 y: 0
ask what is the code? and wait
if answer = open sesame then
    glide 1 secs to x: 300 y: 0
else
    say no it's not
```

Codewords

Think of a (not very) secret word
Write a program that

- asks the user for the codeword ("what is the codeword?")
- tests if the input is equal to the the secret word
- if it is
  - perform an effect to indicate success
- otherwise
  - perform an effect to indicate failure

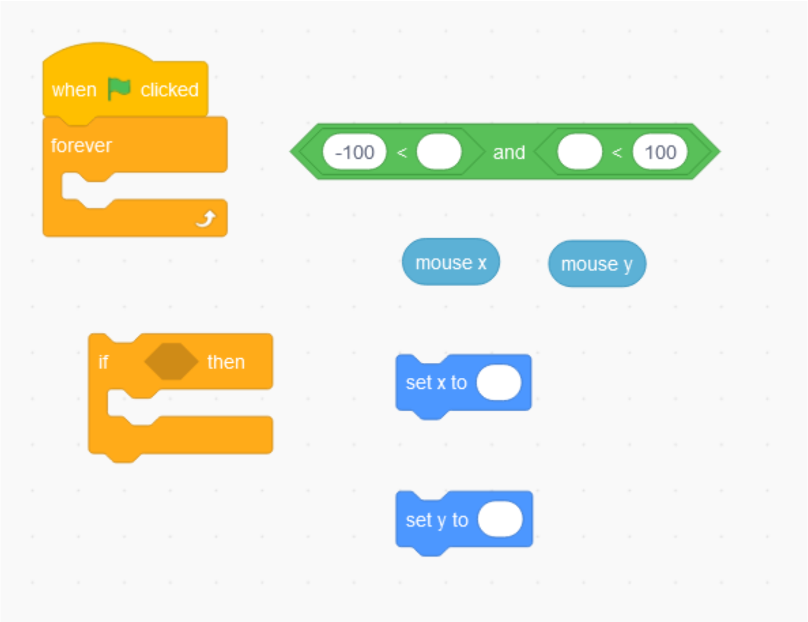
Why should you never do this with an actual secret?

Bounding boxes

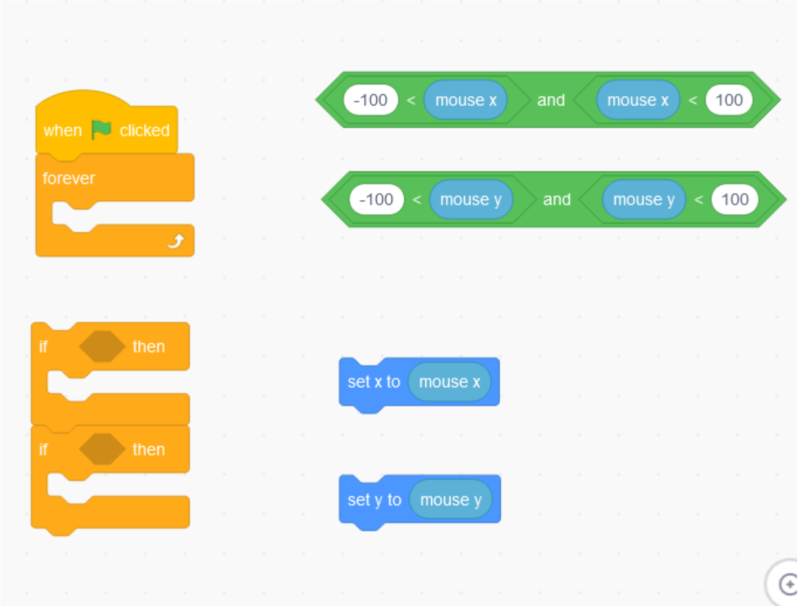
Write a program that

- moves the sprite to the mouse of the user
- except if the mouse is outside of a smaller rectangle on the screen

Hint 1

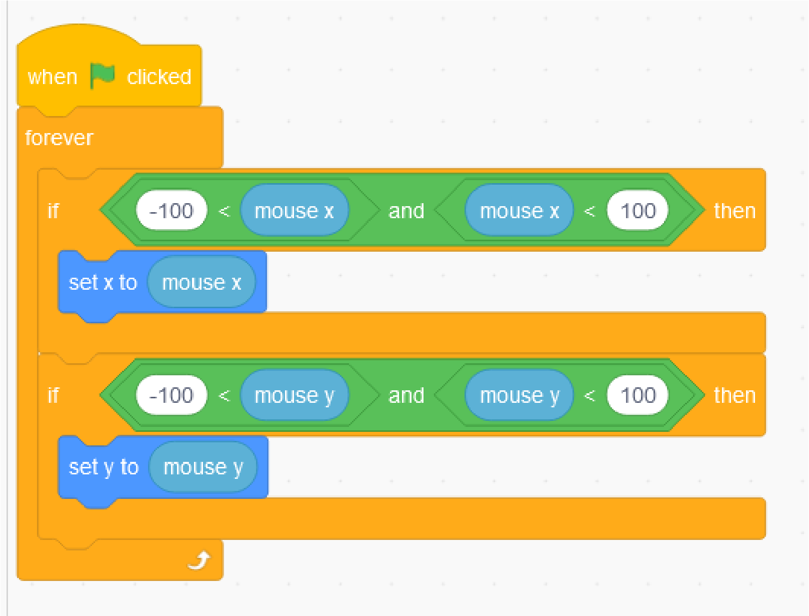

when clicked
forever
-100 < and < 100
mouse x
mouse y
if then
set x to
set y to

Hint 2



```
when ⚑ clicked
forever

if  then

if  then

-100 < mouse x and mouse x < 100

-100 < mouse y and mouse y < 100

set x to mouse x

set y to mouse y
```

Sample solution



```
when 🏁 clicked
forever
    if (-100) < (mouse x) and (mouse x) < (100) then
        set x to (mouse x)
    if (-100) < (mouse y) and (mouse y) < (100) then
        set y to (mouse y)
    ↺
```

A bug

Can you find the “bug”?