Introduction to Computational Thinking and Python Programming

Instructor: Sarom Leang, PhD (Teacher/Professor)

Assistant: Romin Katre

Innovation Hall 203

Pronouns

- Instructor: Sarom Leang, Ph.D. (Teacher/Professor/Instructor)
- Assistant: Romin Katre

Schedule

- 10:00 AM – 10:15 AM (Homeroom)
- 10:15 AM – 11:35 AM (G1)
- 11:40 AM – 12:35 PM (Lunch)
- 12:40 PM – 02:00 PM (G2)

Student Expectations

- **NO FOOD**
- **NO DRINKS** (on the table)
- Be respectful to individuals and property
- Be open to learning
- Be open to not understanding
- Be patient with yourself
- Ask questions
- Explore
- **Embrace failure**

G1

- If we start class 5 minutes early (e.g., at 10:10 AM), then the class will be dismissed to lunch 5 minutes early at 11:30 AM.

G2

- If you arrive in your seat by 12:40 PM then you have the option to finish your lunch in the hallway – if needed.

Course Overview

- This course introduces the fundamental building blocks of computational thinking and computer programming using the Python language.
- Upon successful completion of this course, students will be able to:
  - Improve their problem-solving skills
  - Write, read, and execute Python code using basic data types and operators

Course Overview (cont.)

Exploratory Topics

- ~~Session #1 – October 23, 2023~~
  - ~~Entrance Survey~~
  - ~~How Computers Work~~
- ~~Session #2 – December 2, 2023~~
  - ~~Parallel Computing~~
- ~~Session #3 – February 10, 2024~~
- Session #4 – March 23, 2024
  - Open Lab
- Session #5 – April 20, 2024
  - Generative Artificial Intelligence
  - Exit survey

Topics

- Algorithms and Problem Solving
- Debugging and Troubleshooting
- Loops
- Algorithms and Syntax
- Variables and Conditionals
- Variable Arithmetic
- Conditionals (If/Else)
- Compound Conditionals

G1

- Go to www.ozaria.com and sign-in
- Chapter 1 (Introduction to Coding)
  - 4 out 20 students have completed all modules
- Chapter 2 (Algorithms & Syntax, Debugging, Variables, Conditionals)
  - 2 out of 20 students have completed all modules
- Chapter 3 (Review, For Loops, Nesting, While Loops)
  - 2 out of 20 students have completed all modules
- Chapter 4 (Compound Conditionals, Functions and Data Analysis)
  - 2 out of 20 students have completed all modules

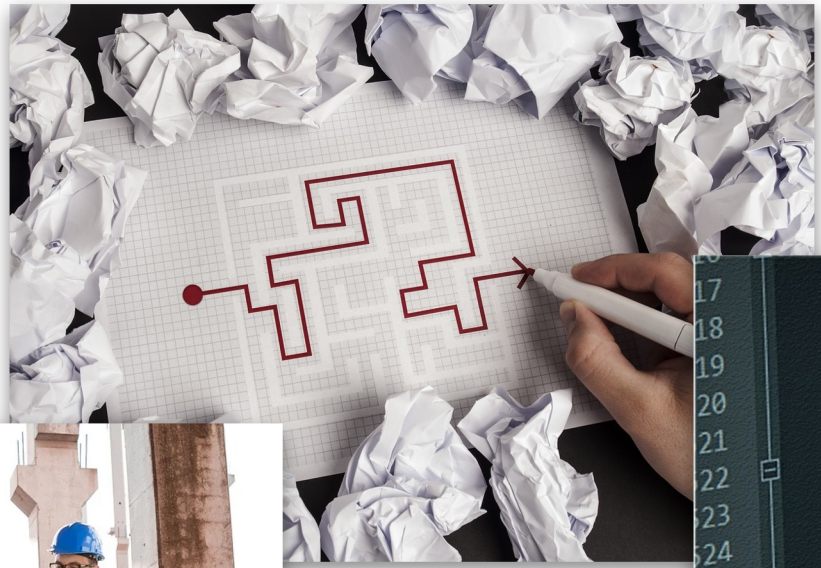
G2

- Go to www.ozaria.com and sign-in
- Chapter 1 (Introduction to Coding)
  - 1 out 17 students have completed all modules
- Chapter 2 (Algorithms & Syntax, Debugging, Variables, Conditionals)
  - 0 out of 20 students have completed all modules
- Chapter 3 (Review, For Loops, Nesting, While Loops)
  - 0 out of 20 students have completed all modules
- Chapter 4 (Compound Conditionals, Functions and Data Analysis)
  - 0 out of 20 students have completed all modules

Algorithms & Syntax

Algorithms

a sequence of instructions that can be used to solve a problem or set of problems



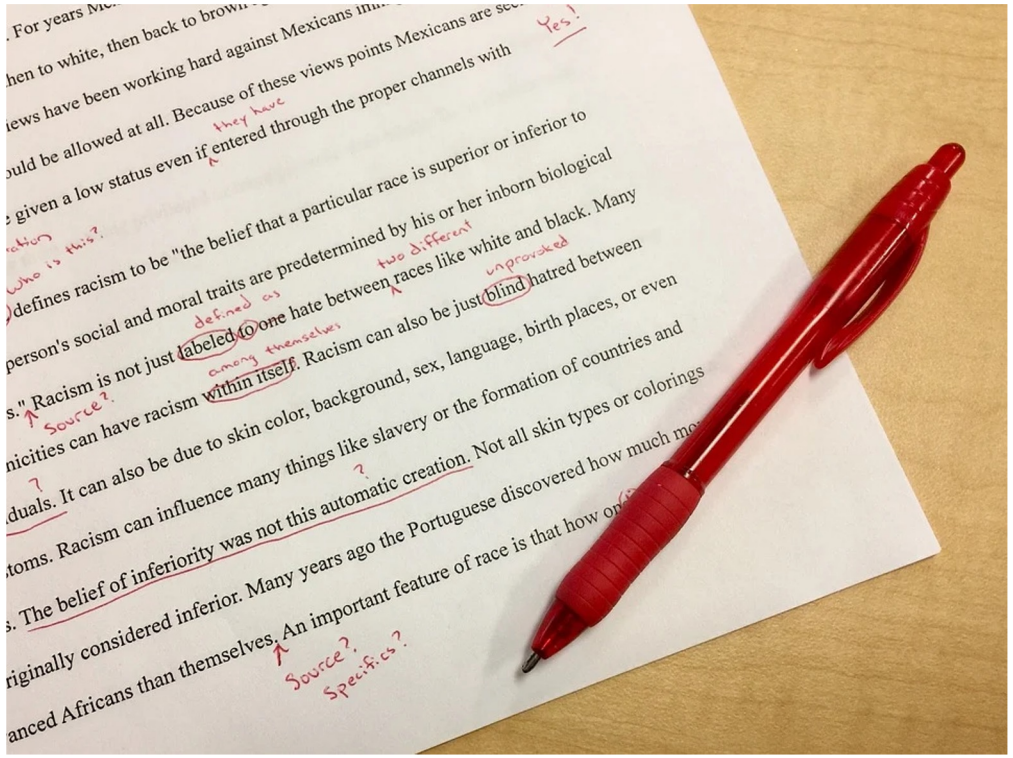
```
string sInput;
int iLength, iN;
double dblTemp;
bool again = true;

while (again) {
    iN = -1;
    again = false;
    getline(cin, sInput);
    system("cls");
    stringstream(sInput) >> dblTemp;
    iLength = sInput.length();
    if (iLength < 4) {
        again = true;
        continue;
    } else if (sInput[iLength - 3] != '.') {
        again = true;
        continue;
    } while (++iN < iLength) {
        if (isdigit(sInput[iN])) {
            continue;
        } else if (iN == (iLength - 3) ) {
```

Syntax

a set of rules for how things are spelled and formatted in code so that the computer can understand it

```
1  hero.moveDown()
2  hero.move()
3  hero.moveRight(2)
4  hero.moveUp(2)
5
```

Object **Method** **Argument**



Concept Check

Let's review the syntax of how objects, methods, and arguments work in our code

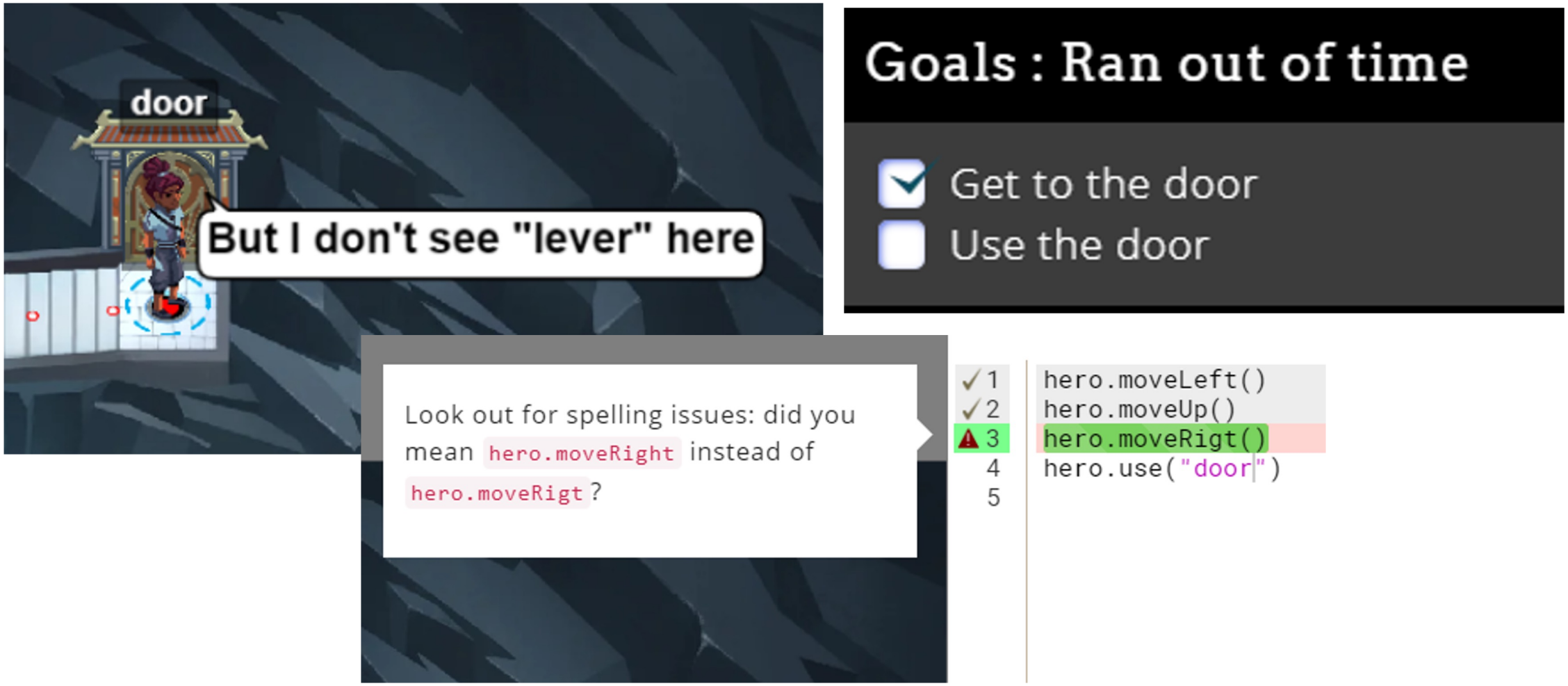
Command	Object	Method	Argument
hero.jumpUp(2)	hero	jumpUp	2
hero.use(“door”)			
helper.build(“rightArrow”)			
helper.moveDown(4)			

Debugging

Debugging

Debugging

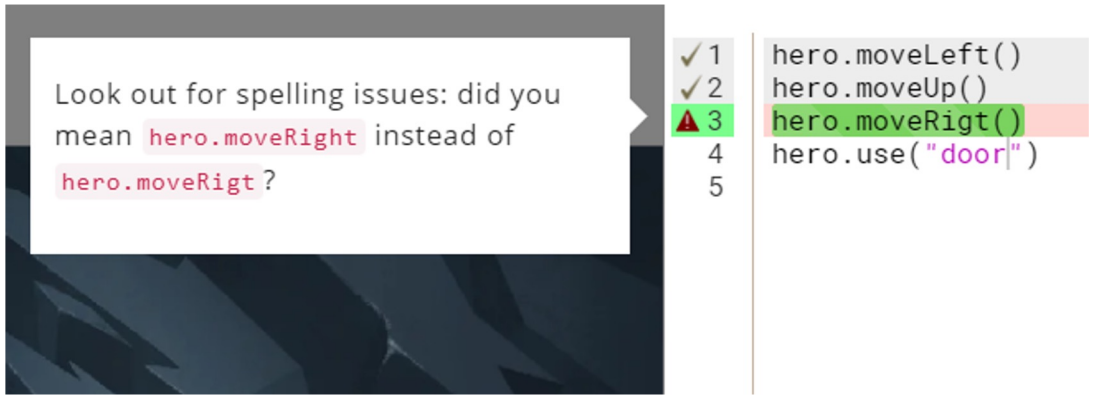
the process of identifying and removing errors from your code so you can meet your goals



Syntax & Logic Bugs

Sometimes you need to try different strategies to fix different kinds of bugs

Syntax Bugs

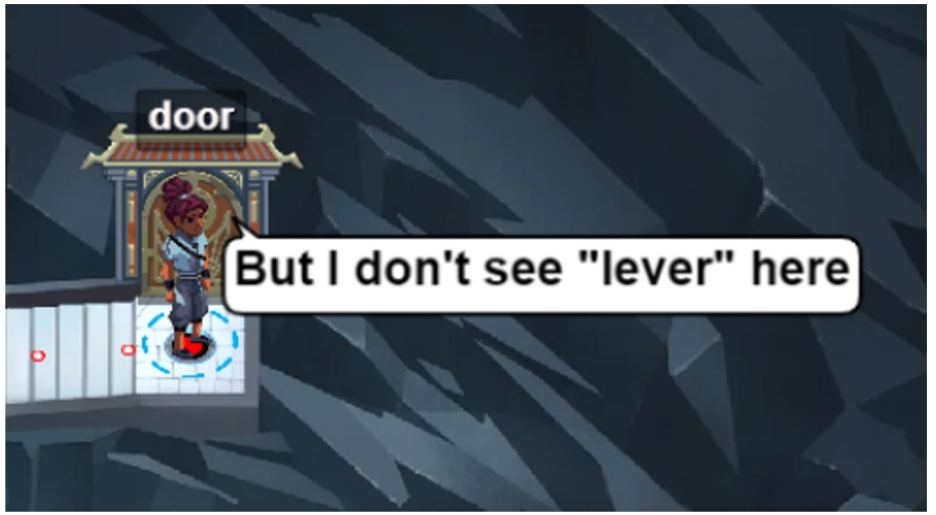


Mistakes with Spelling or Formatting

moveRigt()

hero.jumpUt(“2”):

Logic Bugs



Mistakes with the Algorithm or Logic

hero.use(“lever”)

Introduction to Variables

Variables

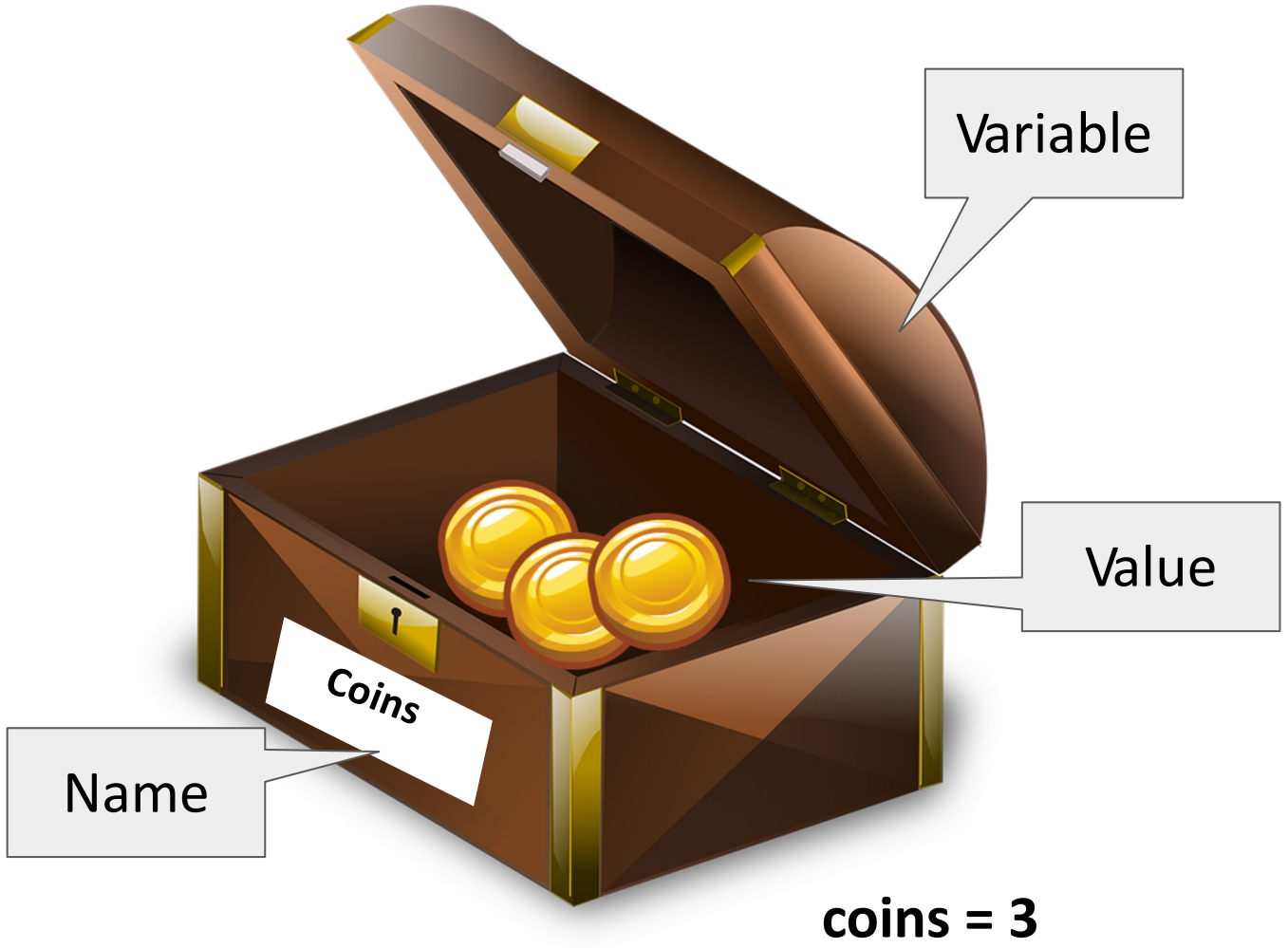
Variables

Variables are a way for us to keep track of information in our code, such as numbers and words.

Variable: *The box is like a variable. It can hold or store something that you can take out and use whenever you need it.*

Value: *The number of coins in the box is the value of the variable.*

Variable Name: *You label the box to remember what kind of value it's storing. When you use a lot of variables, you will want to use names to keep track of them.*



Variables

What kind of information do you need to track every day? How can they be stored as variables?

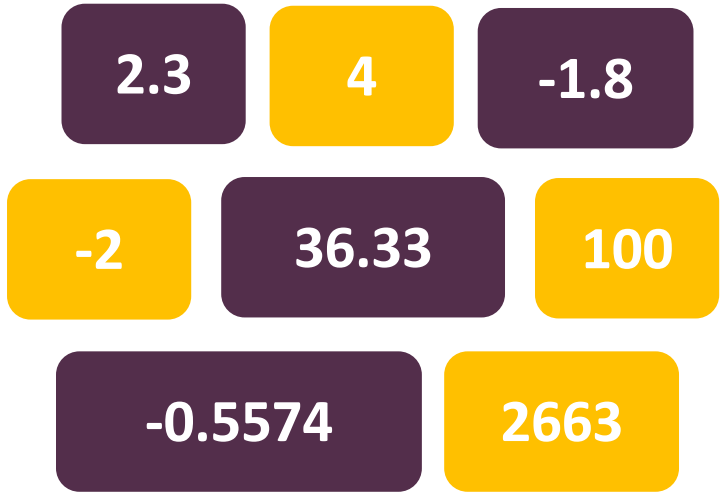
password = “L33tCodez”

teamOneScore = 6
teamTwoScore = 21

OzariaFlightNumber = “E83”

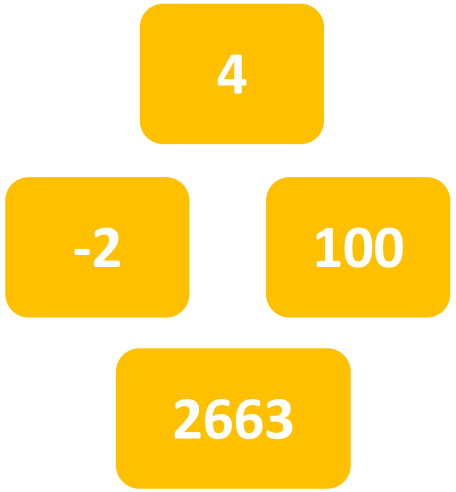
Numbers & Variables

There are different types of numbers that can be stored by variables.



Numbers

Any number



Integers

A whole number that is not a fraction or decimal

Floats

A number that has decimals

Story Builder

Our Variables

- age =
- name =
- vegetable =
- float =
- year =
- integer =

Story Builder

Our Variables

- age =
- name =
- vegetable =
- float =
- year =
- integer =

Once upon a time, there was a **[age]** old astronaut named **[name].**

[name] lived on the planet **[vegetable]**, which was **[float]** light years away from Earth.

In the year **[year]**, **[name]** decided to visit Earth. **[name]** brought **[integer]** bags of space **[vegetable]** chips to share with the Earthlings.

When **[name]** arrived on Earth, the Earthlings loved the space **[vegetable]** chips and built a statue in **[name]**'s honor.

All hail, **[name]**!

Programming Tips

We are always looking for new ways to make our code faster, efficient, and error proof. We achieve this with **structured programming**.

```
steps = 2

hero.moveRight(2)
hero.moveDown()
hero.moveLeft(2)
```

```
steps = 2

hero.moveRight(steps)
hero.moveDown()
hero.moveLeft(steps)
```

Regular

With regular programming, although it requires less planning ahead, a program is prone to errors and hard to update.

Structured

Structured programming uses reusable elements, like variables, to make a program clearer and easy to update.

Concept Check: Variables

In this example, what part of the image is the variable, the name, and the value?

Variable

a way to store information in our program so we can use it later

Name

we give each variable a name so we can keep track of different ones

Value

the information that you store inside a variable that you can change over and over again

Variables with Strings

Variables with Strings

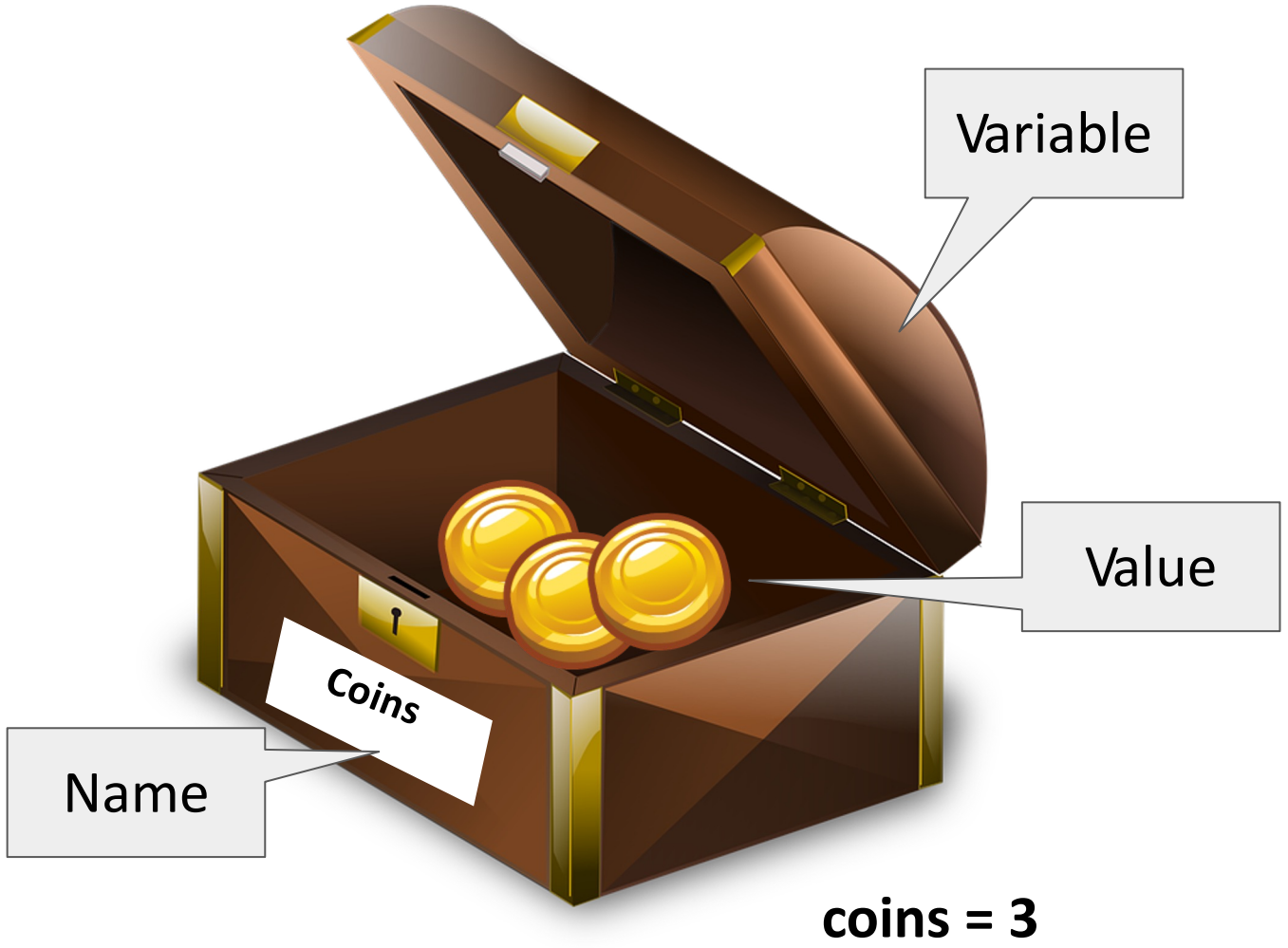
Variables

Variables are a way for us to keep track of information in our code, such as numbers and words.

Variable: *The box is like a variable. It can hold or store something that you can take out and use whenever you need it.*

Value: *The number of coins in the box is the value of the variable.*

Variable Name: *You label the box to remember what kind of value it's storing. When you use a lot of variables, you will want to use names to keep track of them.*



Strings

Strings are a series of characters that can be numbers, letters, and more! Strings are a type of data that can be stored in variables.

password = “St#ngs&Codez”

“Capella”

favoriteFood = “I like popcorn!”

Why Use Strings?

Strings help you keep track of important information such as names, addresses, and more! You can store **strings** inside variables so you can easily access them in your code.

Let's say you have a string that lists the address to your home:

"123 Cabbage Lane,Moon Village,NY 10054 USA"

That's a lot to type each time you want to use that information in your code. Instead, you can **store the string in a variable:**

home = "123 Cabbage Lane,Moon Village,NY 10054 USA"

Now, you only have to **type in the string variable name** (**home**) whenever you need to **use the address in your code**:

hero.goTo(home)

Why Use Strings?

Using string variables also makes it easy to **change the strings**.

You might **create a character** with pink hair and brown eyes whose name is Carmen.

name = “Carmen”
hair = “pink”
eyes = “brown”

name = “Julie”
hair = “blue”
eyes = “brown”

If you store all of this information as string variables in your program, then you can change them at any time by just **reassigning the variable.**

Concept Check: Strings

Sort the data! Which ones are strings? Which ones are integers? Which ones are variable names?

Strings	**Integers**	**Variable Names**
“1246”	3342	name

helloWorld

l33t

1337

“name”

steps

“Hello, world!”

“l33t”

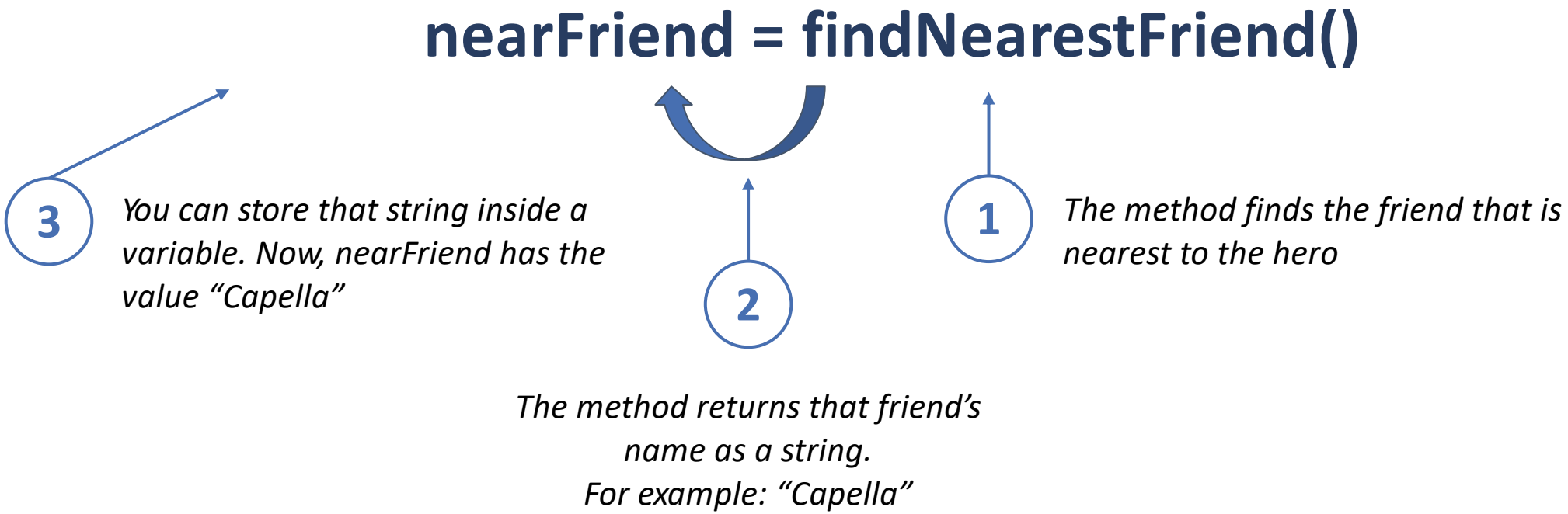
10,423

“steps”

friend

Methods with Information

Some methods return different kinds of information such as strings or numbers. You can store that information into a variable.



Concept Check: Story Builder

Our Variables

- age = myAge()
- name = myName()
- song =
- food = myFavoriteFood()
- number =

Story Builder

Our Variables

- age = myAge()
- name = myName()
- song =
- food = myFavoriteFood()
- number =

Once upon a time, there was a **[age]** year old coder named **[name]**.

[name] liked to hum the song **[song]** while coding. It was so annoying that their teammates would throw **[food]** until **[name]** would stop singing.

Still, **[name]** was the best coder on the team and could write [number] lines of code every day. Maybe **[song]** was **[name]**’s secret power?

No one will ever know.

Introduction to Conditionals

Conditionals

Conditionals

We use conditionals to help us plan and make decisions. We check a **condition** to see if it’s true or false. If the condition is true, then we perform an **action**.

If

It is raining.

condition

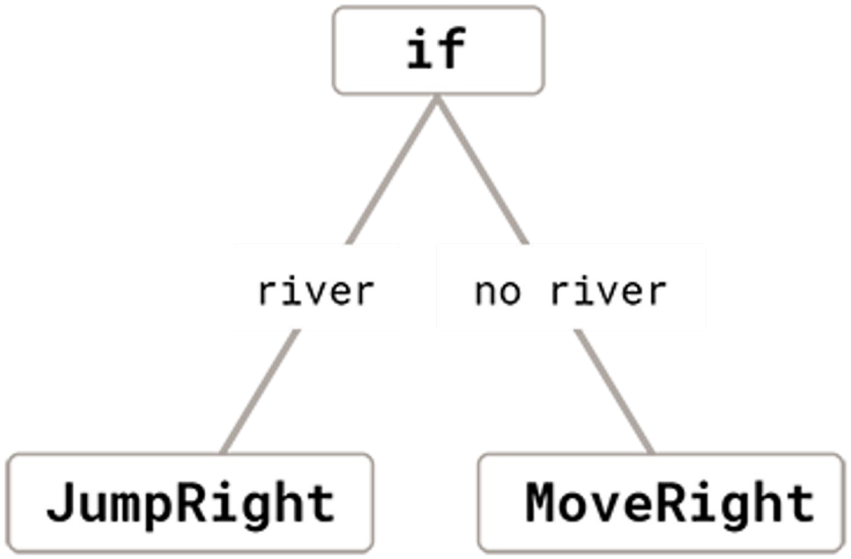
Then

Bring an umbrella!

action

Conditionals

Programs use conditionals to make decisions. This allows you to use the same program even if conditions change.



condition

indented action

IF there is a **RIVER**
    hero **jumpRight**
IF there is **NO RIVER**
    hero **moveRight**

Concept Check: Conditionals

Fill out the table using pseudocode

	Condition		**Action**
	There is a door		Use the door
IF	There is a pit	**THEN**	Jump over the pit

IF there is a **DOOR**
hero **USES DOOR**

IF **[condition]**
[action]

Comparators & Conditionals

Comparators & Conditionals

Comparators

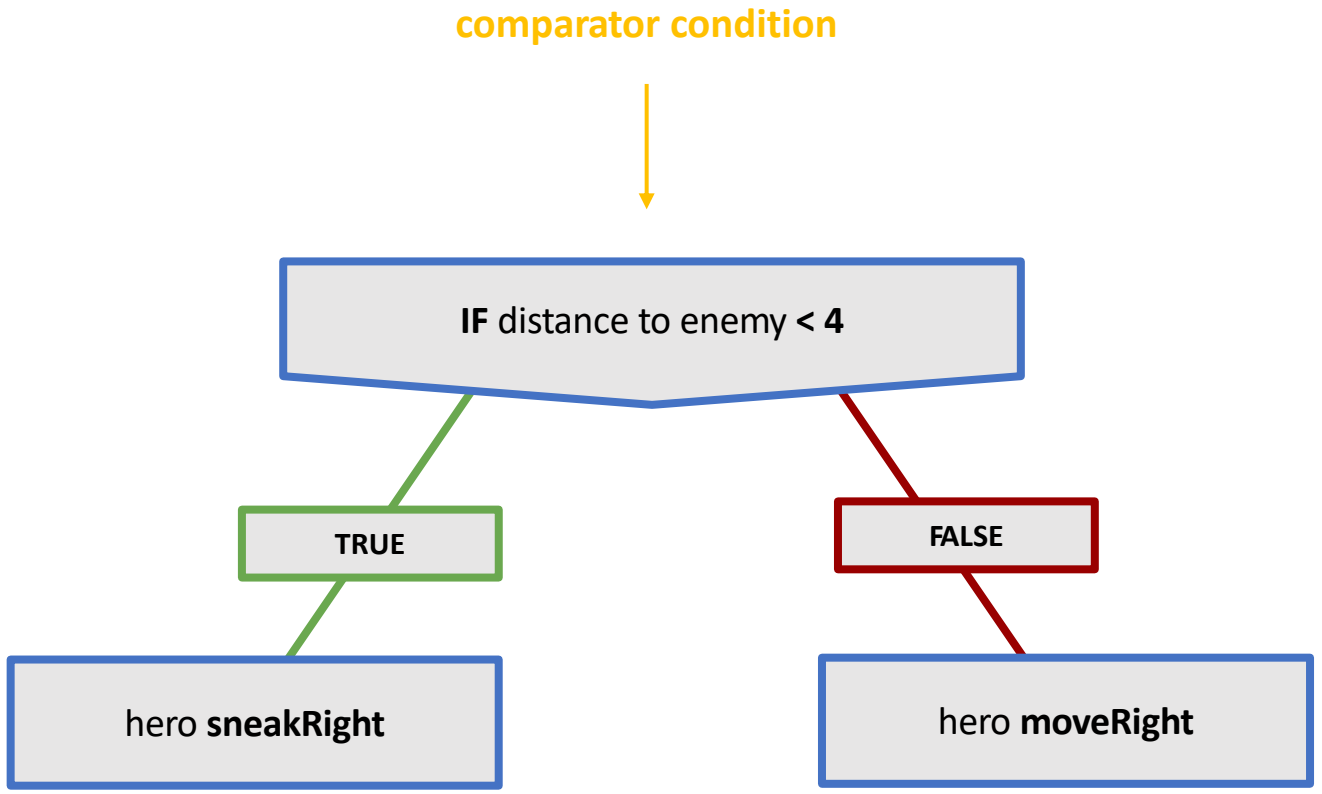
How do we use comparators in math?

Is 3 < 5?

Is 3 > 5?

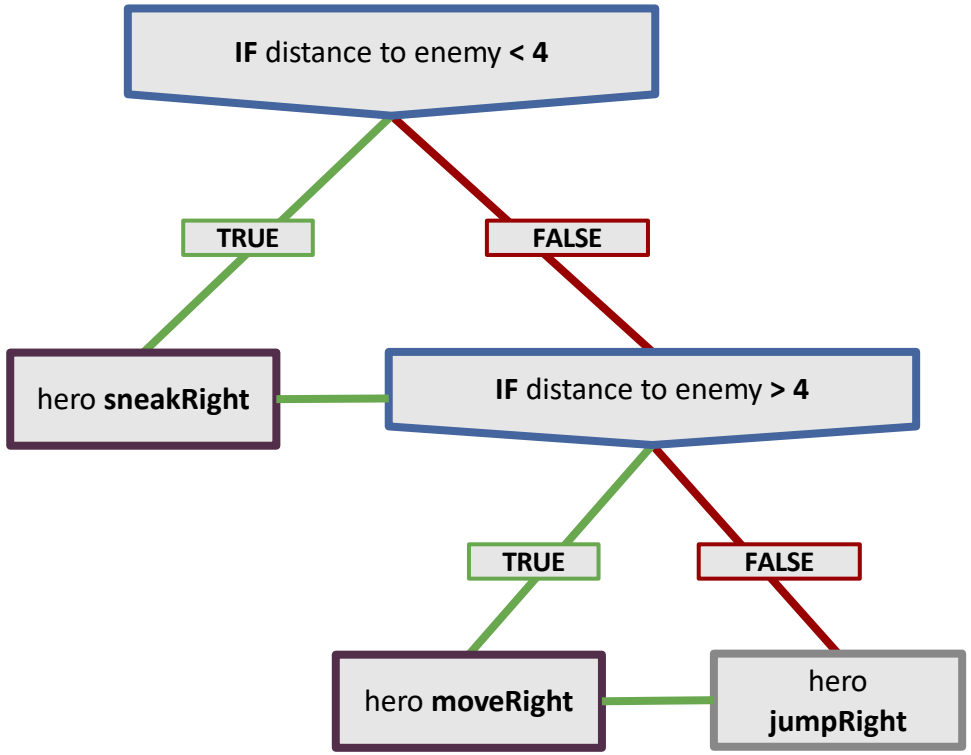
Comparators & Conditionals

Comparators can be used in conditionals too



Comparators & Conditionals

The way conditionals work is a little more complicated...




```
IF distance < 4
        hero sneakRight
IF distance > 4
        hero moveRight
hero jumpRight
```

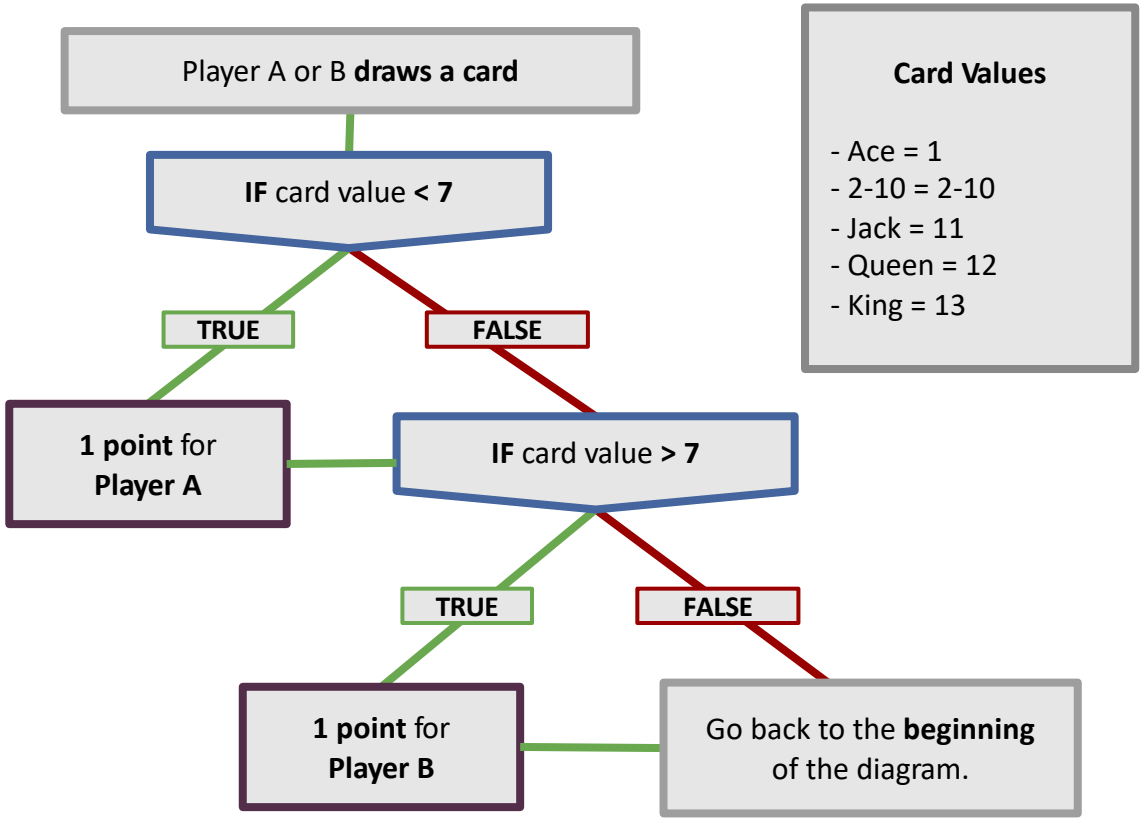
Only happens if **distance < 4**

Only happens if **distance > 4**

Will **always happen** after the hero sneaks right or moves right

Concept Check: Comparators

Let's play a card game using comparators and conditionals!



Card Values

- Ace = 1
- 2-10 = 2-10
- Jack = 11
- Queen = 12
- King = 13

PSEUDOCODE!

Draw a Card

card = get VALUE of CARD

IF card **< 7**
    Player A gets 1 POINT

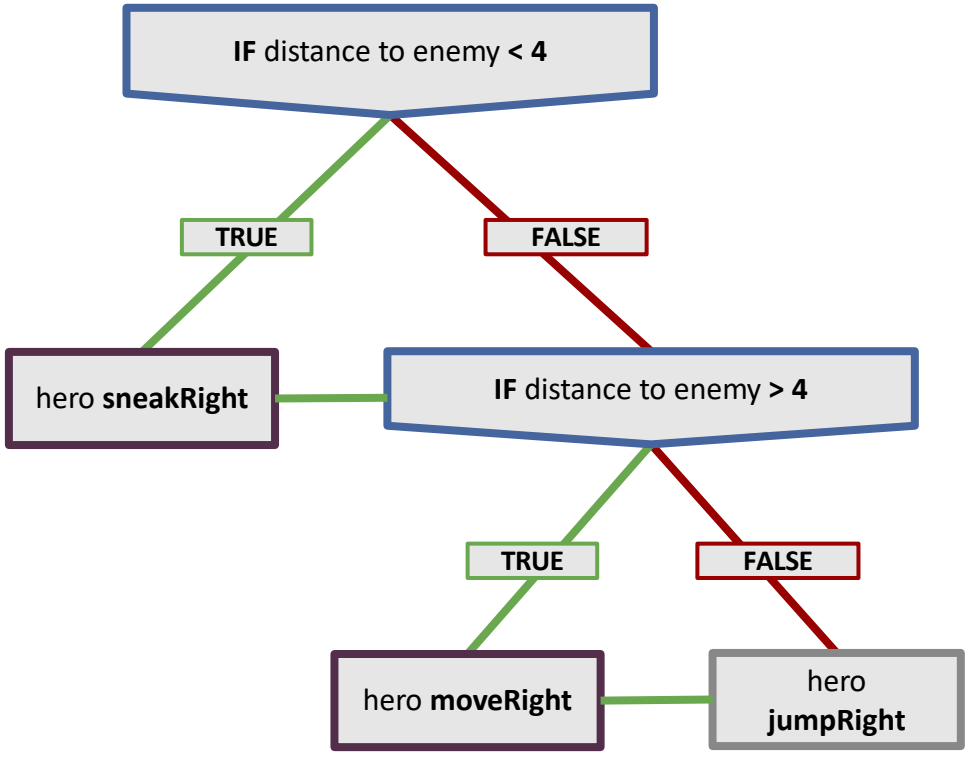
IF **[condition]**
    [action]

If/Else Conditionals

If/Else Conditionals

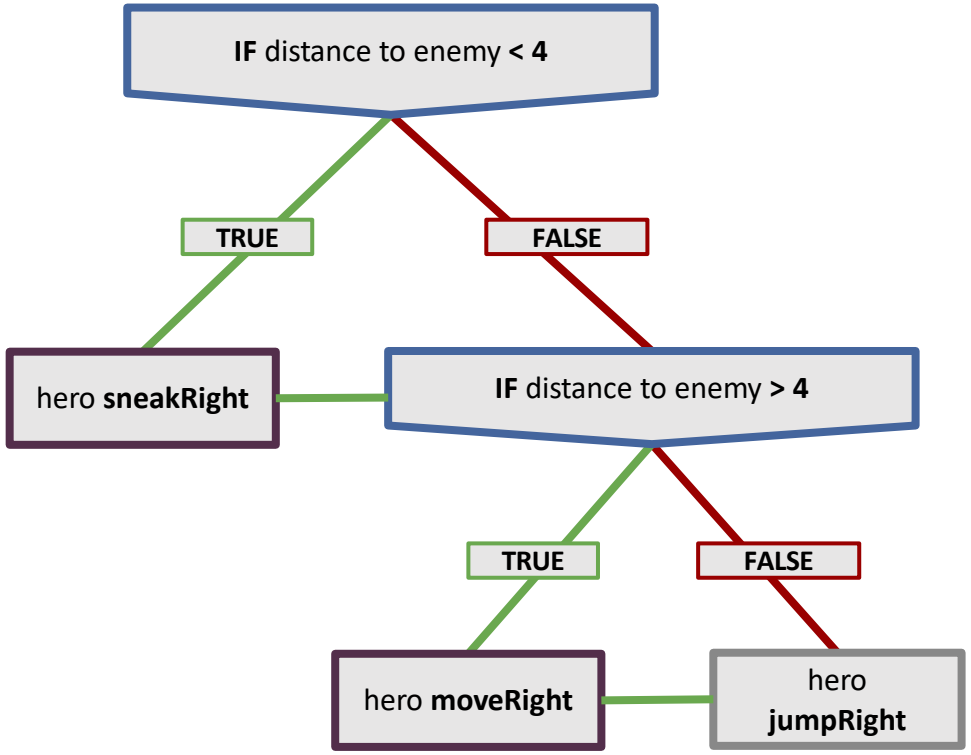
If/Then Conditionals

How did we use a flowchart to map these conditionals?

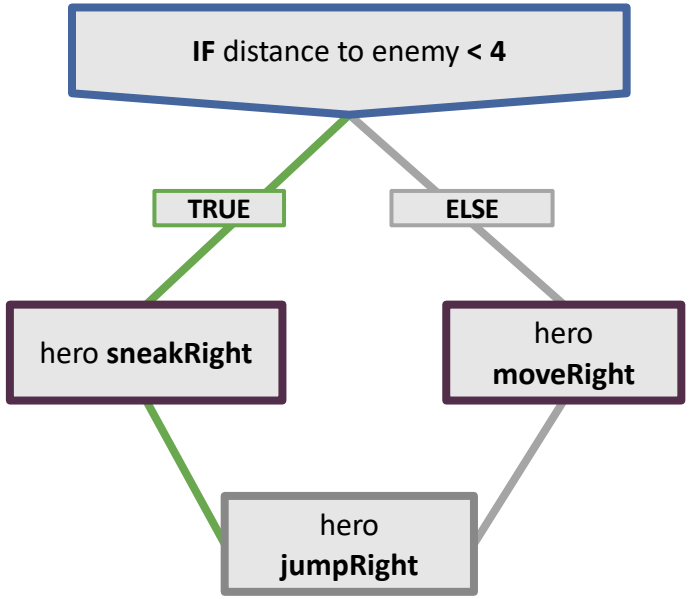


If/Else Conditionals

What do you notice is different between the two flowcharts?



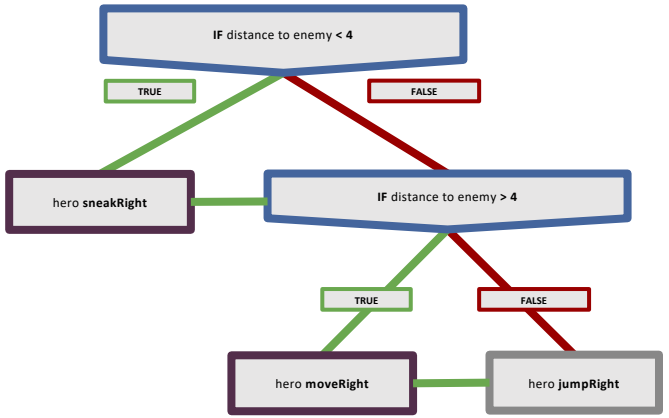
If/Then



If/Else

Programming Tips

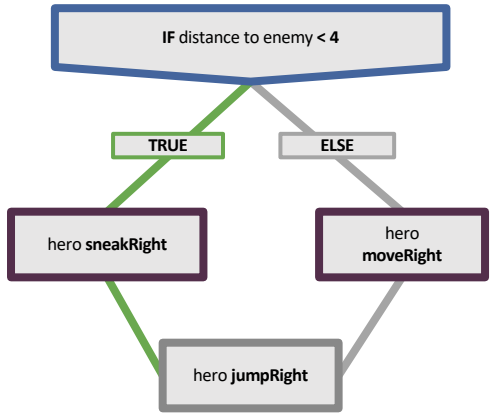
We are always looking for new ways to make our code faster, efficient, and error proof. We achieve this with **structured programming**.



If/Then

Regular

With regular programming, although it requires less planning ahead, a program is prone to errors and hard to update.



If/Else

Structured

Structured programming uses reusable elements, like variables, to make a program clearer and easy to update.

If/Else Conditionals

Let's see how if/else conditionals work in pseudocode


```
IF distance < 4
        hero sneakRight
IF distance > 4
        hero moveRight
hero jumpRight
```

If/Then

```
IF distance < 4
        hero sneakRight
ELSE
        hero moveRight
hero jumpRight
```

Only happens if **distance < 4**

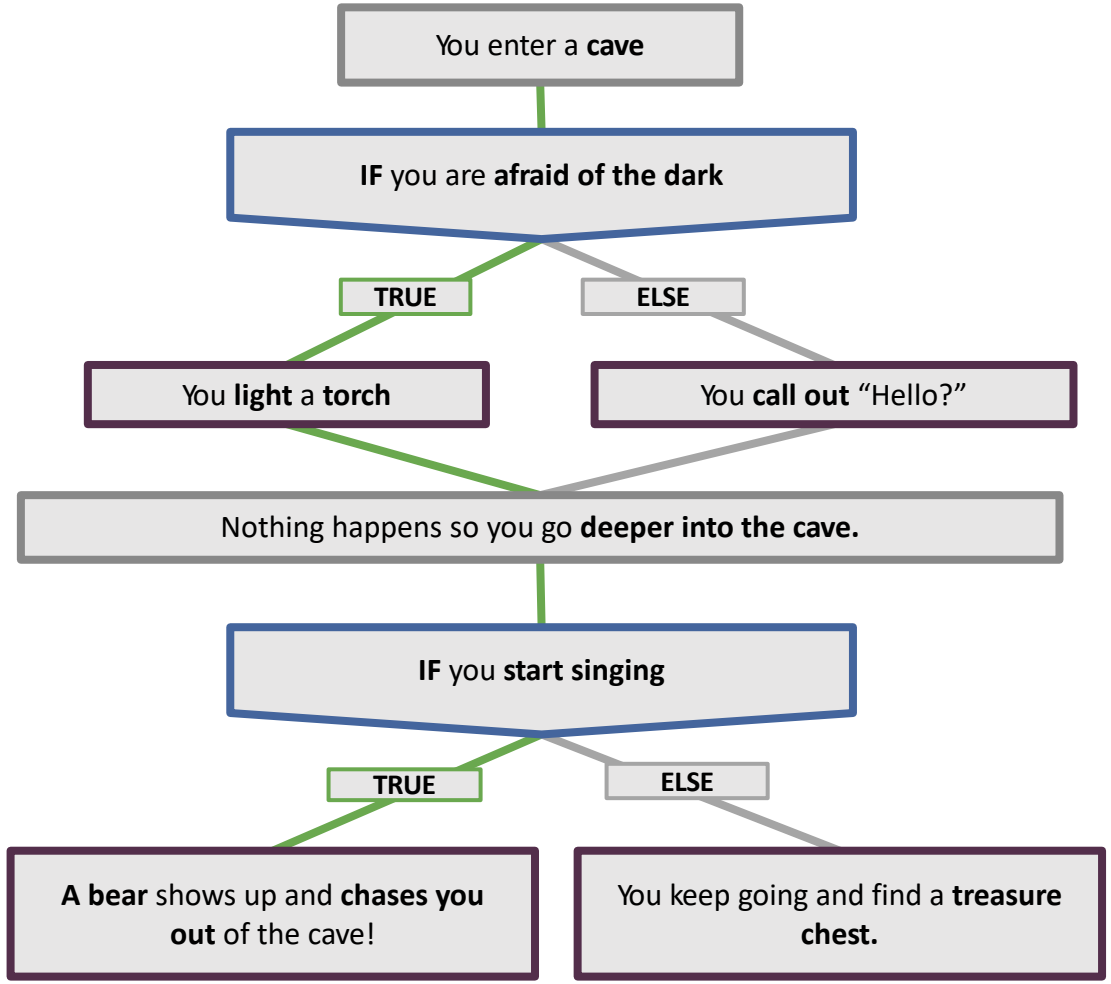
ELSE this happens if the condition is **false**

Will **always happen** after the hero sneaks right or moves right

If/Else

Concept Check: If/Else Conditionals

Let's choose our own adventure using if/else conditionals!



PSEUDOCODE!

hero **enters cave**

IF afraid of the dark
    hero **lightsTorch**
ELSE
    [action]

If/Else Conditionals

The syntax of if/else conditionals is quite similar to if/then conditionals. Remember where to use indentations and colons/braces!

```
if distance < 4:
        hero sneakRight()
else:
        hero moveRight()

hero jumpRight()
```

Python

```
if (distance < 4){
        hero sneakRight();
}else{
        hero moveRight;
}

hero jumpRight;
```

JavaScript

Variable Arithmetic

Today's Journey

- **Warm-Up**: Variable Values
- **Variable Arithmetic**
- **Play Ozaria**: Use Variable Arithmetic with Mouse
- **Optional Extension**: Design a Variable Puzzle Tournament

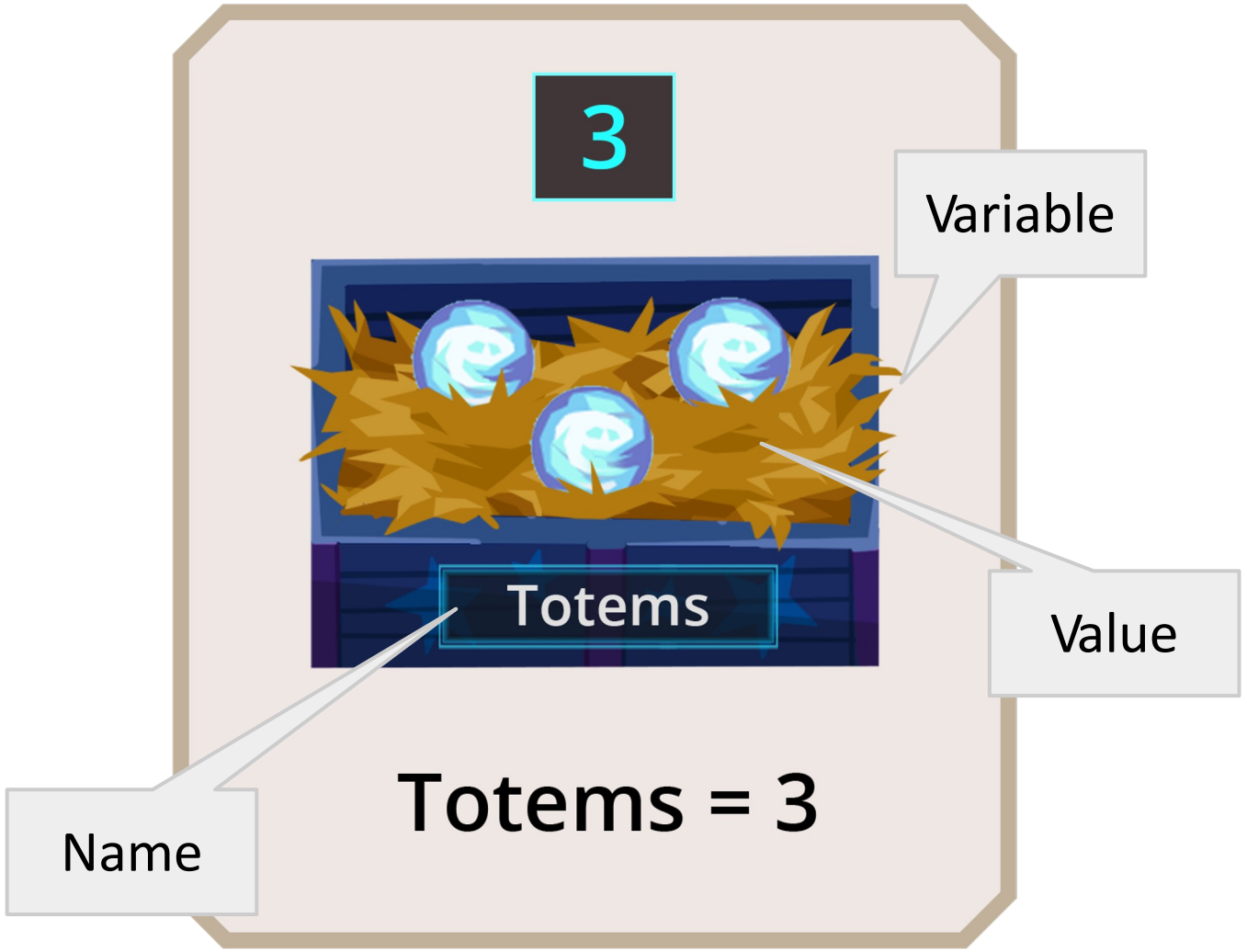
Variables Review

Variables are a way for us to keep track of information in our code, such as numbers and words.

Variable: *The box is like a variable. It can hold or store something that you can take out and use whenever you need it.*

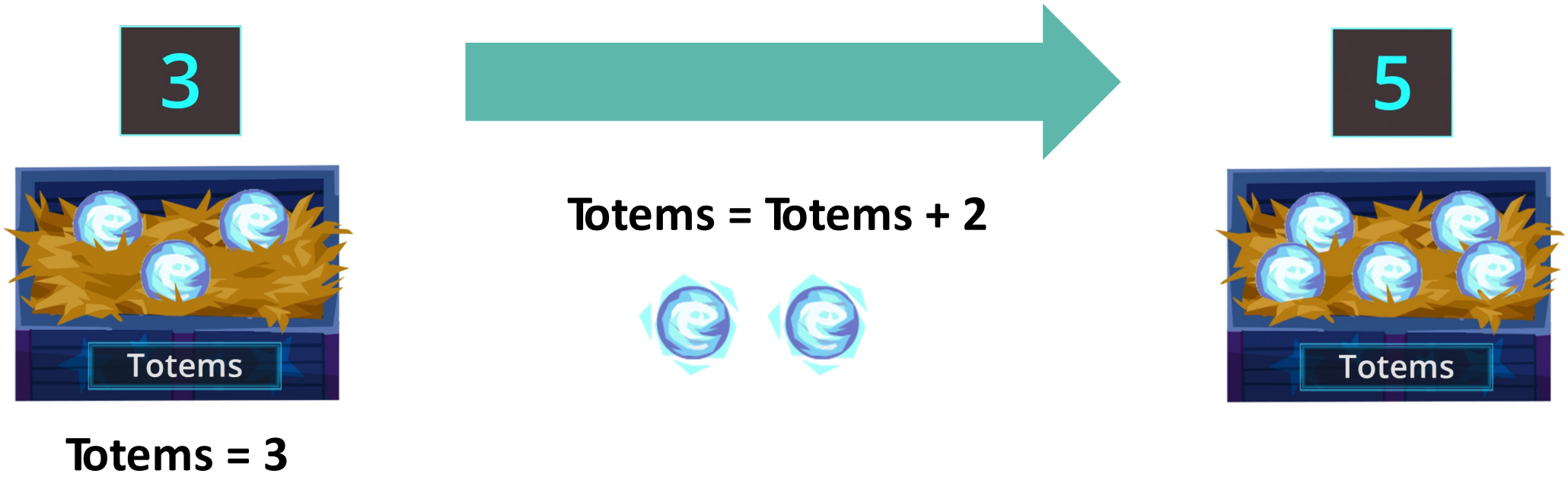
Value: *The number of totems in the box is the value of the variable.*

Variable Name: *You label the box to remember what kind of value it's storing. When you use a lot of variables, you will want to use names to keep track of them.*



Arithmetic with Variables

Powering up the variables tool!



Totems - 2 Totems x 2 Totems/3

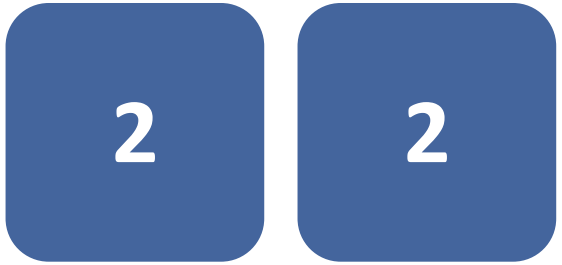
Concept Check: Variable Arithmetic

Figure out what the resulting variable value is for each of the arithmetic expressions

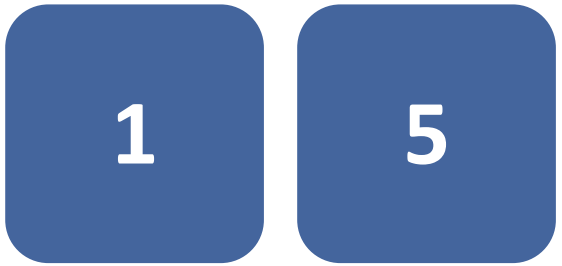
Initial Variable Value	**Arithmetic**	**The variable value is now. . .**
Totems = 14	Totems = Totems - 2	12 Totems
Totems = 4	Totems = Totems * 2	
steps = 9	steps = steps / 3	
health = 38	health = health + 2.2	
friends = 8	friends = friends * 8	

Variable Puzzles

Use the 4 numbers to create a list of arithmetic expressions that will get to the **number 24!** See who can solve each puzzle first.



number = 4 + 2
number = number + 6
number = number * 2



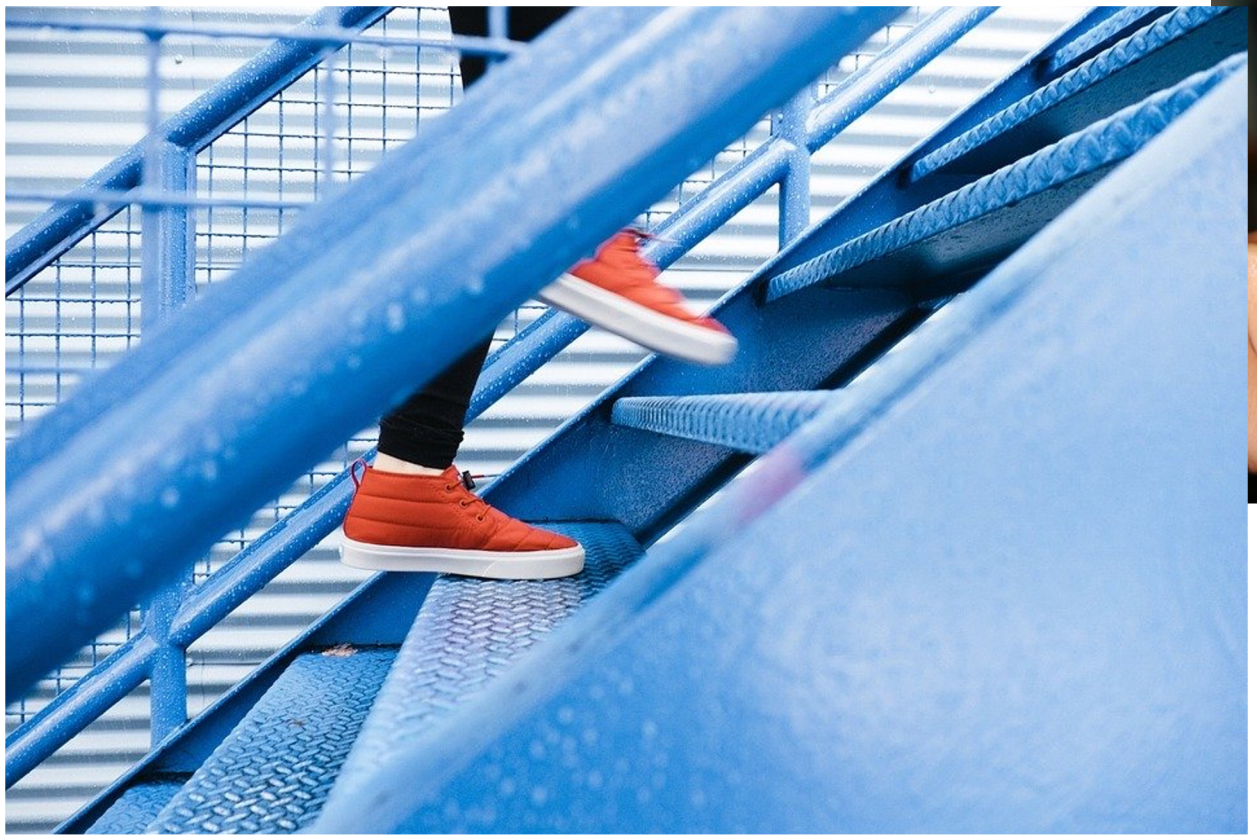
number =

number =

For Loops

Review: For Loops

a sequence of instructions that repeats a specified number of times



Concept Check: For Loops

Create a Dance

- Write down a repeating sequence of steps or commands for your dance (examples: clap, step left, etc.)
- Indicate repeating steps using loops
- Have other groups perform your dance
- Example steps:
  - Repeat 2x: clap
  - Step left
  - Step right

Use Correct Syntax with For Loops

Python

```
for i in range (3):
    hero.moveRight()
    hero.moveUp()
hero.moveLeft()
```

Concept Check: Syntax with Loops

Let's Debug Python Code

1.
```
for i in range(2):
hero.moveUp(2)
hero.movedown(3)
```

2.
```
for i in range(4)
    hero.moveUp(2)
    hero.movedown(3)
```

3.
```
    For i in range(2):
hero.moveUp(2)
hero.moveDown(2)
```

4.
```
for i in range 2:
    hero.moveUp(4)
    hero.movedown(3)
```

Lesson 1

Nested Loops

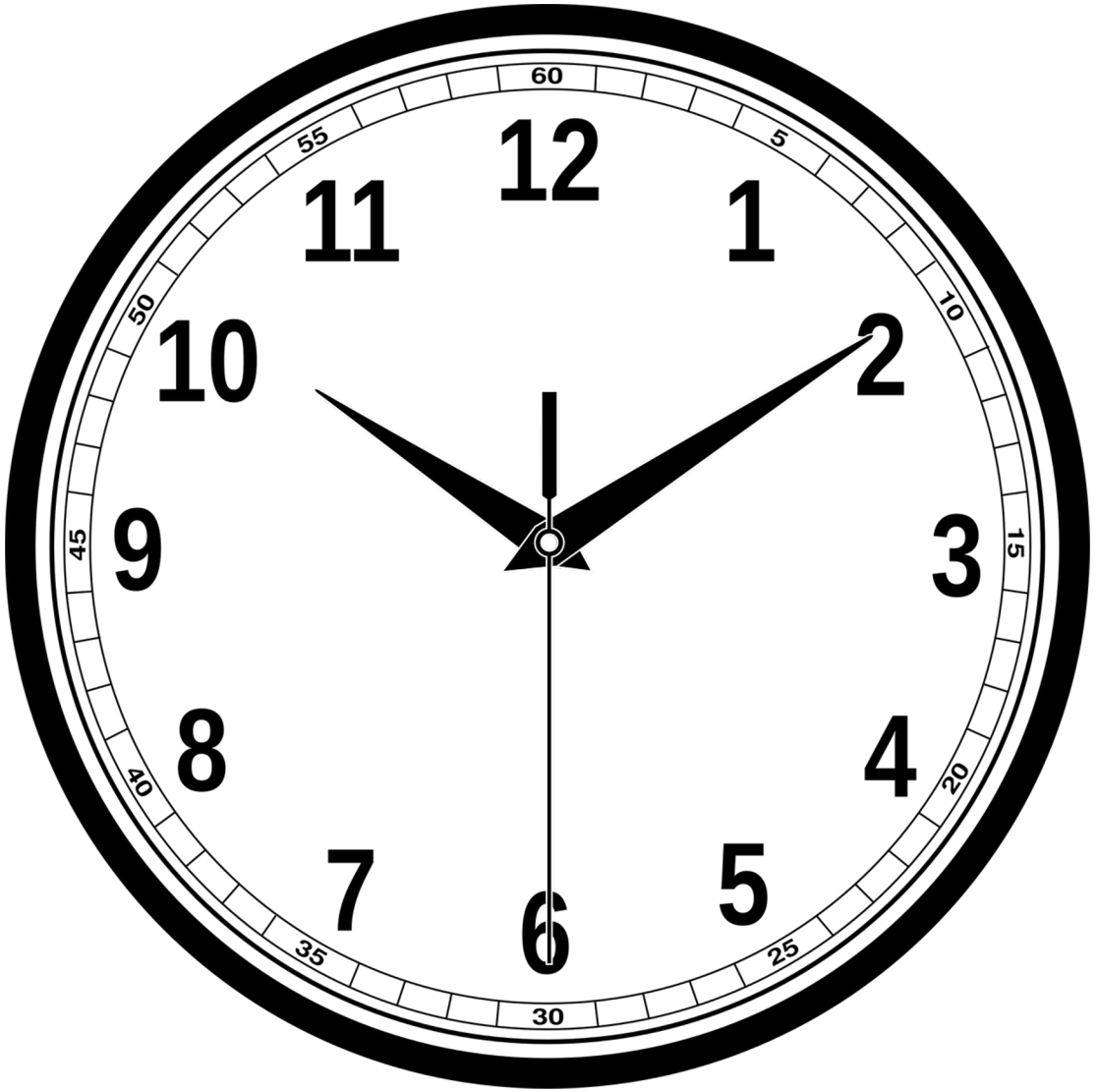
Review: For Loops

a sequence of instructions that repeats a specified number of times

Nested Loops

A loop within another loop is called a **nested loop.**

Describe the loop action that occurs with a clock.



Now think about what other actions occur during that loop (think about the second hand, the minute hand, and the hour hand).

- Is that action also a loop?
- Is it a **nested** loop?

Concept Check: Nested Loops

Write the directions in pseudocode to make a clock tell time across the span of one day. Include outer and inner loops.

```
for...
    for...
        for...

            [#do something]
            [#do something]
        [#do something]
```

Think about:

- Which is the first action? (This is your outer loop. It only repeats after all of your inner loops are complete.)
- What actions should be in your inner loops? (Those actions repeat multiple times before the outer loop action repeats.)

Lesson 2

Nested Structures

Review: Conditionals

Conditional statements tell your program to execute actions depending on whether a condition is true or false.

Condition	[true or false]	Action
If it's cold,	→	**then** I'll wear a coat.
If...	→	**then...**

Review: If/Else and If/Then Conditionals

if/else and if/then conditionals are used when we want an action to occur only when a certain condition is met

IF distance **< 4**
    hero **sneakRight**
IF distance **> 4**
    hero **moveRight**
hero **jumpRight**

If/Then

IF distance **< 4**
    hero **sneakRight**
ELSE
    hero **moveRight**
hero **jumpRight**

If/Else

Only happens if **distance < 4**

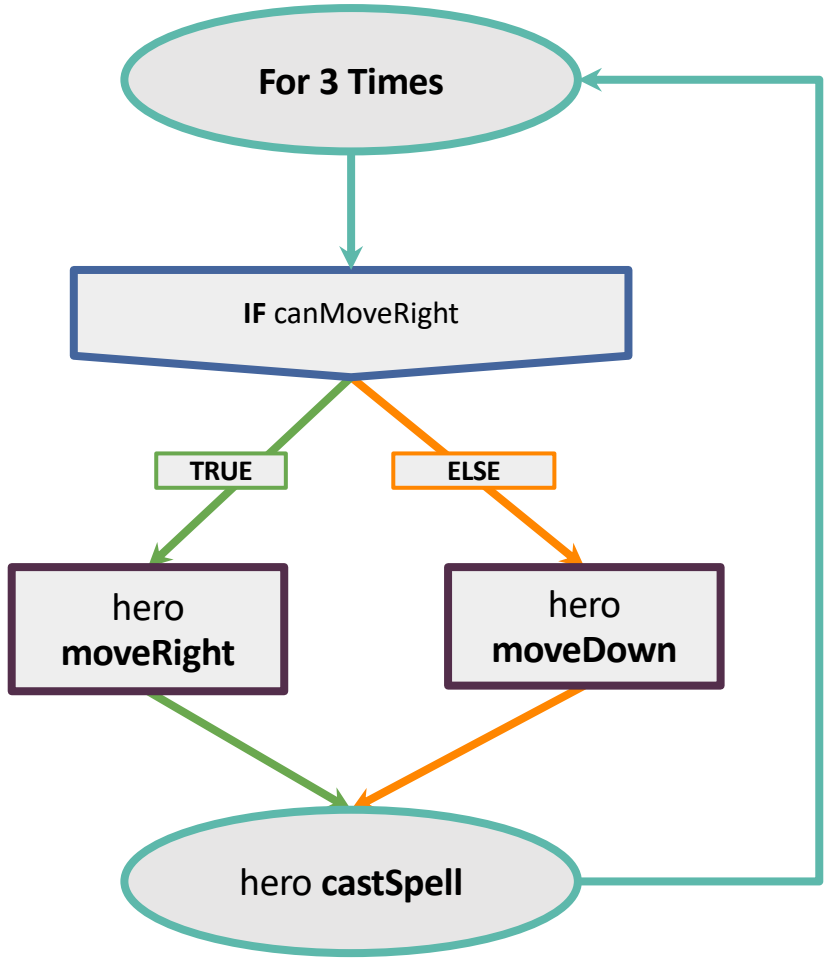
ELSE this happens if the condition is **false**

Will **always happen** after the hero sneaks right or moves right

Nested Structures

if/else and if/then conditionals can be used inside loops too! This lets you check the same conditional multiple times.

```
FOR 3 times
    IF canMoveRight
        hero moveRight
    ELSE
        hero moveDown
    hero castSpell
```

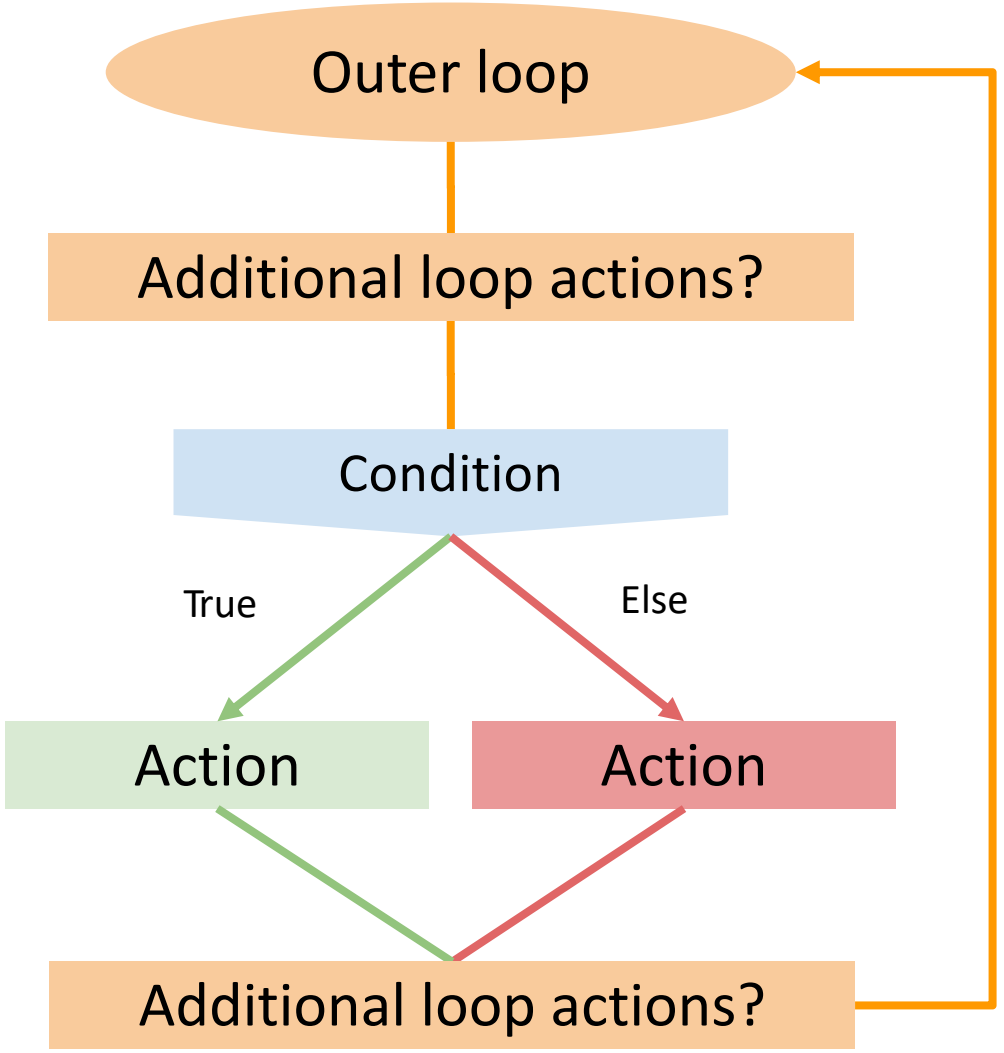


Concept Check: Nested Structures

Think of an example of a day to day activity that involves an **if/else conditional nested inside a loop**.

Use this chart to describe:

- The outer loop that repeats
- The condition that needs to be met
- What action happens if the condition is true
- What happens if the condition is false
- Any additional actions that happen in the loop



While Loops

While Loops

Warm Up

While Loops

a way of repeating a sequence of code **while** a **condition** is true

while my savings are less than the bike costs:
    save allowance
buy the **bike**

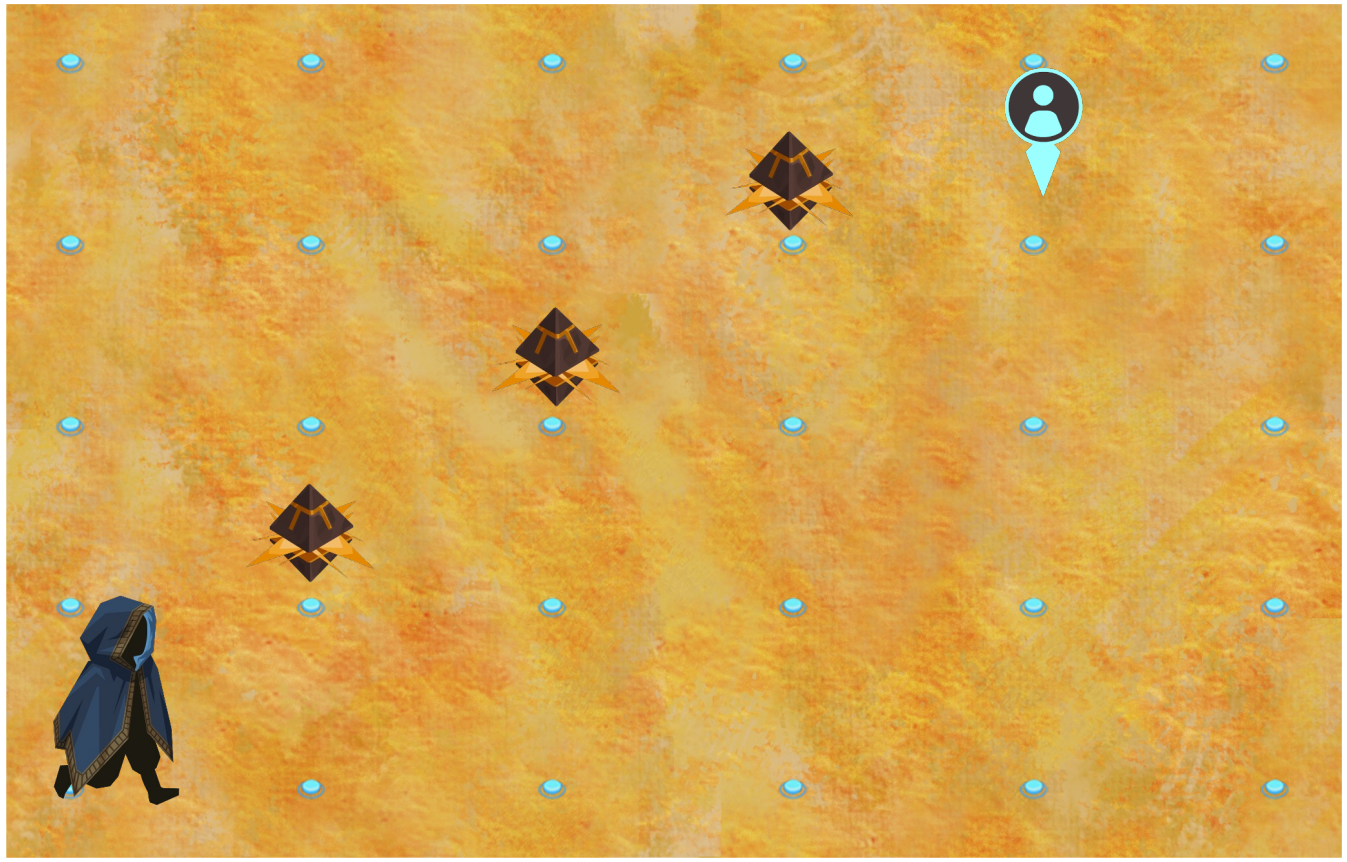
What are some other examples of actions you'd repeat while something is true?

While Loops

Loops can be based on **conditions** - statements that can be either true or false.

An example of a **condition** might be: there are still totems on the map

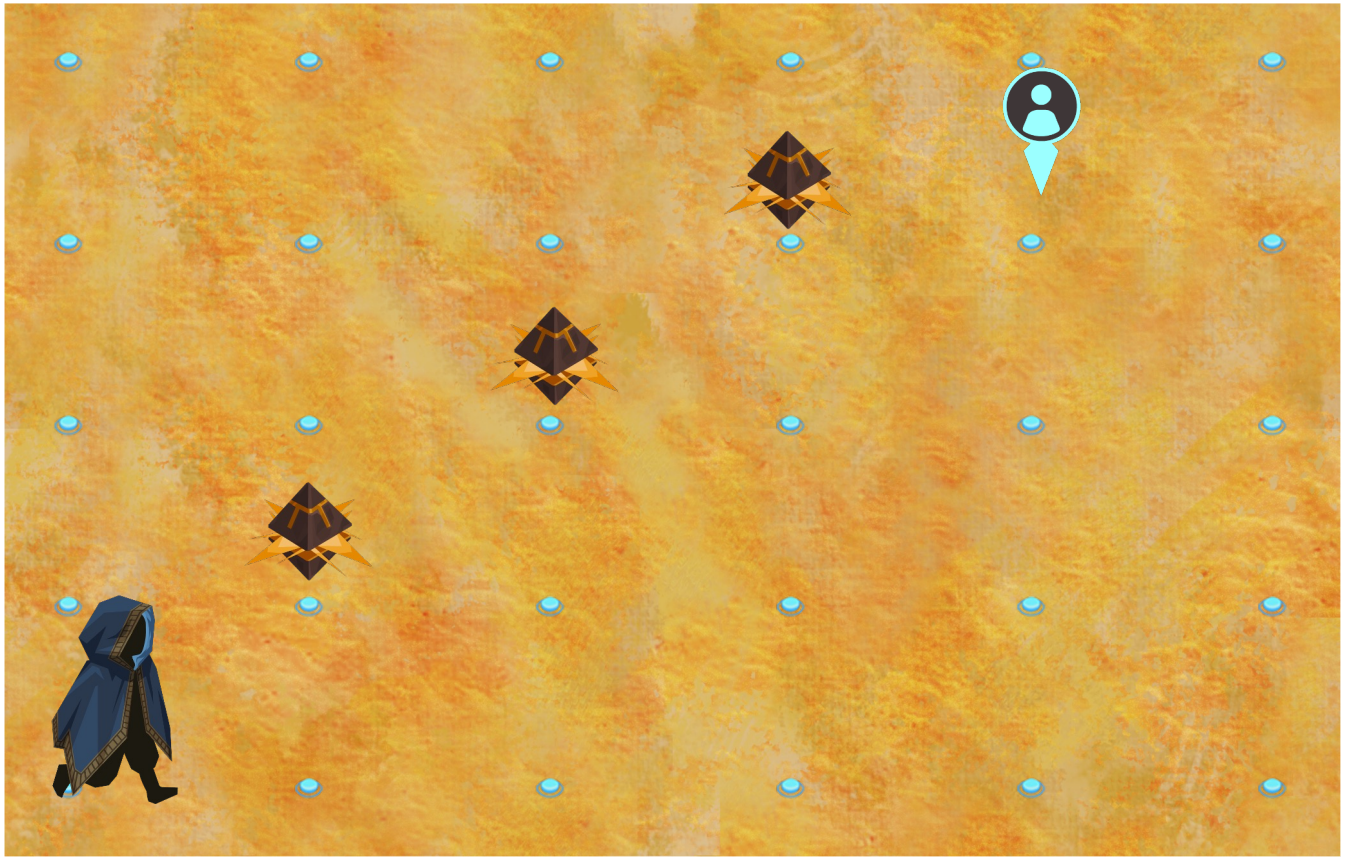
```
while totems < 3:
        hero moveRight
        hero moveUp
hero moveRight
```



While Loops Syntax: Python

Parts of a while loop:

1. **while** keyword
2. condition (check to see if it's true)
3. body (indented actions that repeat while the condition is true)



Don't forget:

1. The **colon** after the condition!
2. To **indent repeated statements** within the while loop

```
while totems < 3:
        hero.moveRight()
    hero.moveUp()
hero.moveRight()
```

Concept Check: While Loops

Write the code needed for the hero to get to the exit marker with all three totems.

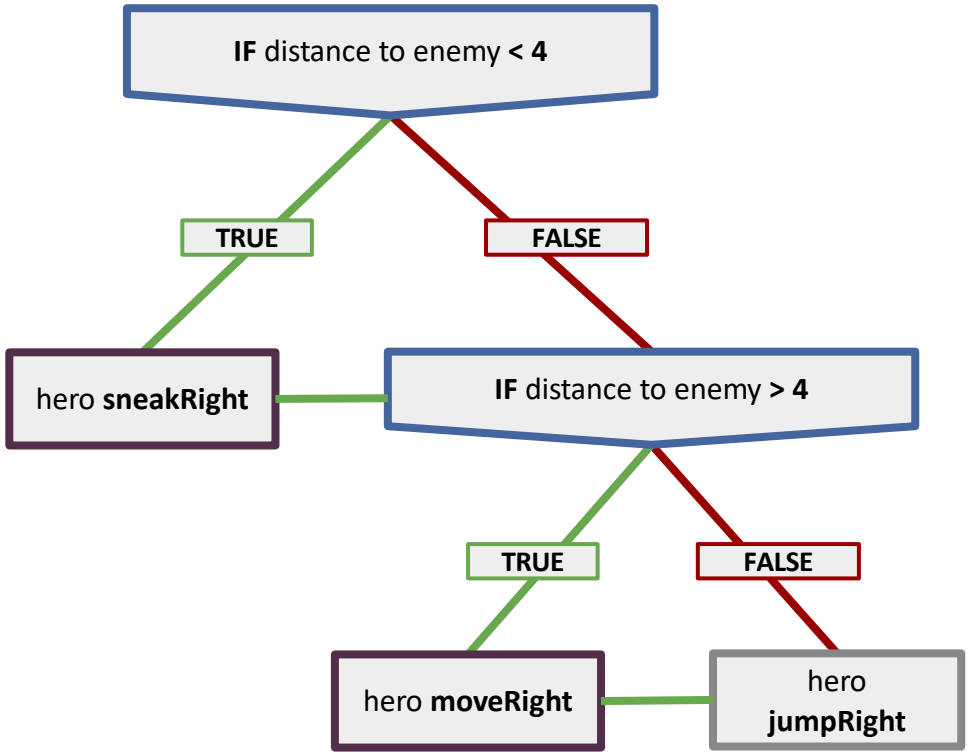
Include a while loop to write the sequence of steps for the hero to collect all the totems.

```
while condition
```

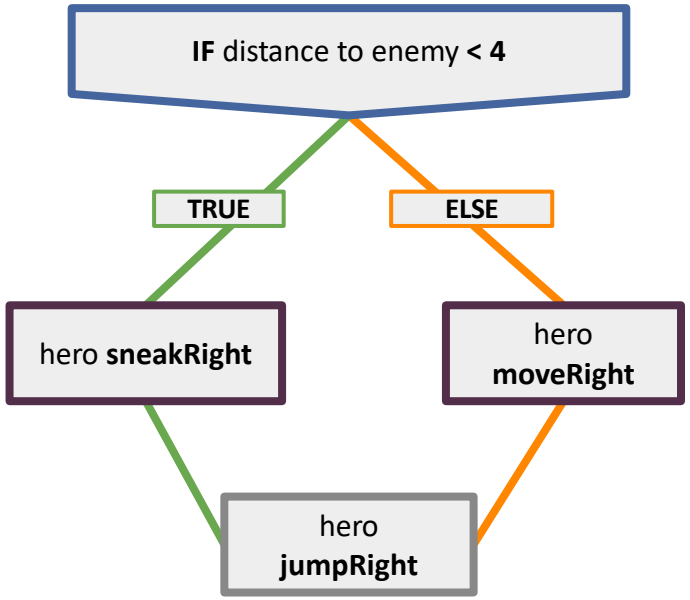
Else/If Conditionals

Review: Conditionals

How have you used if/then and if/else conditionals?



If/Then



If/Else

Else/If Conditionals

Else/if conditionals let you check multiple conditions in a specific order and guarantee that only one of these actions will happen.

1. **IF** doorOne = UnLocked
   hero **use "doorOne"**
2. **ELSE IF** doorTwo = UnLocked
   hero **use "doorTwo"**
3. **ELSE IF** doorThree = UnLocked
   hero **use "doorThree"**
4. **ELSE IF** doorFour = Unlocked
   hero **use "doorFour"**

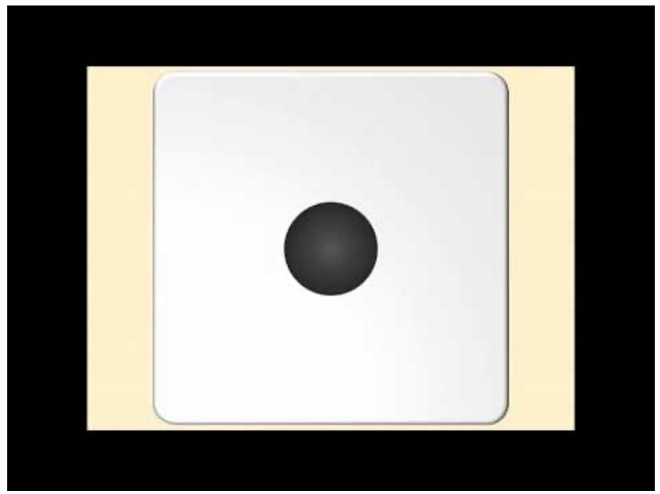
Only happens if **Door 1 is unlocked**

Only happens if **Door 1 is locked** and **Door 2 is unlocked**

Conditional Fortune Telling

1. **Ask a question about your future.** Example: “Should I have pizza for dinner?”

2. **Roll a die**

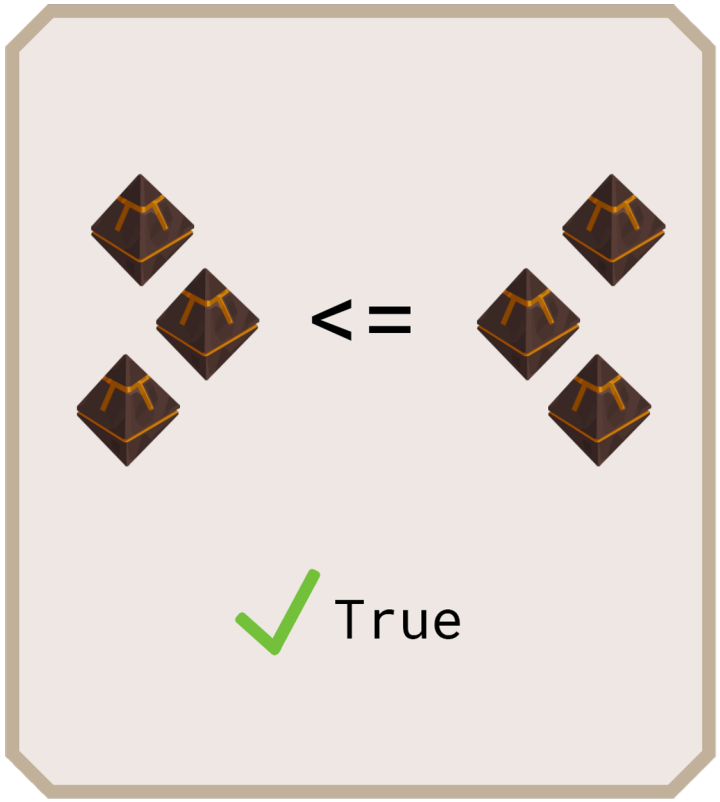


3. **Use the else/if conditional to find your fortune!**

```
IF number < 3:
        hero says “Yes!”
ELSE IF number < 5:
        hero says “Maybe.”
ELSE
        hero says “Try again.”
```

Else by itself works just like if/else conditionals. The else action will happen for **any number** that is **not** < 3 or < 5.

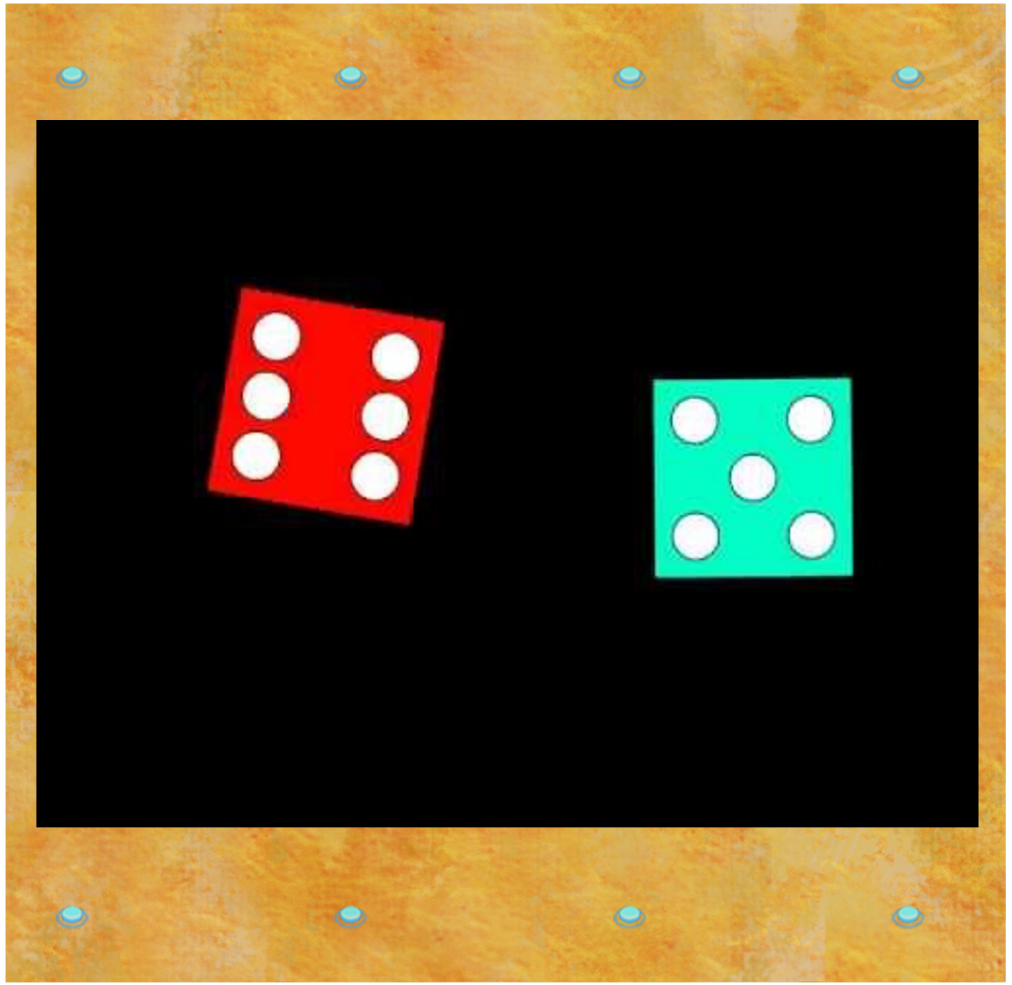
New Comparators



Comparator	Meaning
<=	Less than **or equal**
>=	Greater than **or equal**
<	Less than
>	Greater than
=== or ==	Equal

Concept Check: Else/If Conditionals

Finish the else/if conditional so that there's a fortune each time you roll both dice (1-12).



PSEUDOCODE!

```
IF number <= 2:
        hero say "Yes!"
ELSE IF number <= 4:
        hero say "Maybe."
ELSE IF number <= 6:
        hero say "Check back tomorrow."
ELSE IF  :
```

Lesson 2

Compound Conditionals

NOT Conditions

Put a **NOT** in front of a condition to check if it's **not true**.

IF NOT sunny:
Vega **use "umbrella"**

What are some **NOT conditionals** that you use in your **everyday life**?

IF NOT doing homework:
hero

IF NOT:
hero

Compound Conditionals: AND

Compound conditionals can check **multiple conditions at once**. Use **AND** to combine two or more conditions and check if they're **all** true.

IF raining AND outside:
Vega **use "umbrella"**

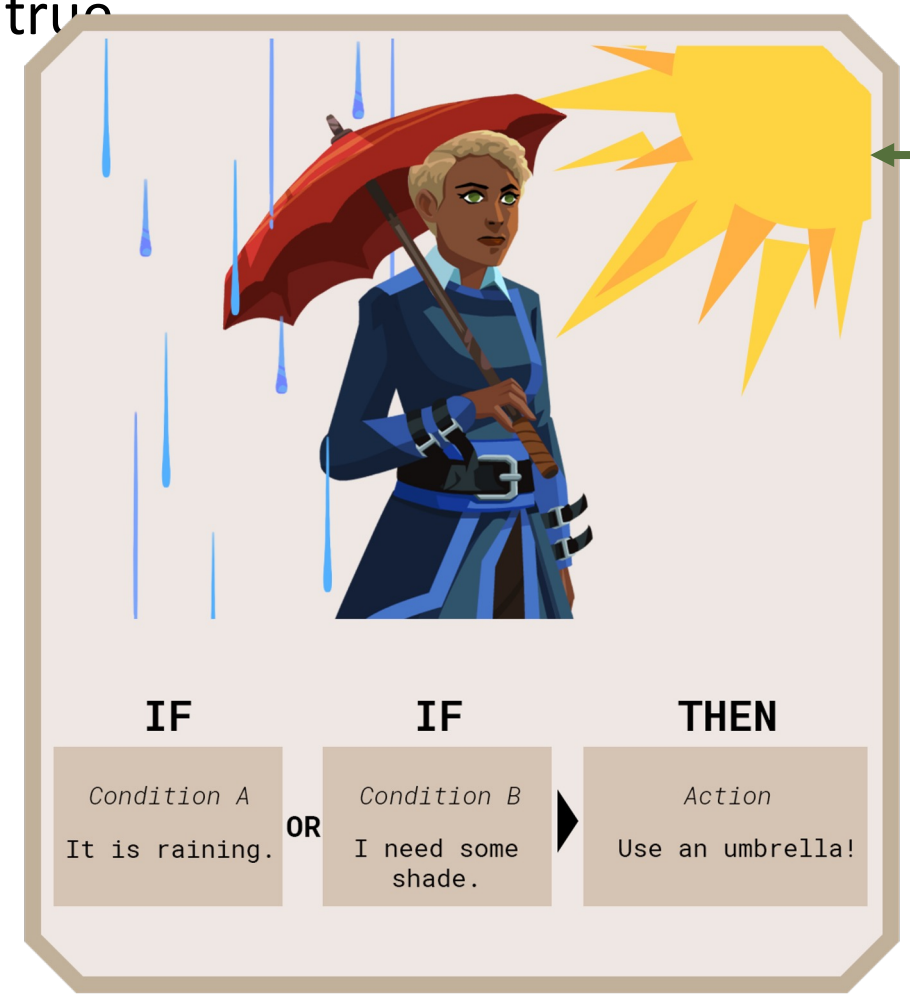
When do you use **compound conditionals** that have **AND** in your **everyday life**?

IF hungry AND _____:
hero

IF _____ AND _____:
hero

Compound Conditionals: OR

Use OR to combine two or more conditions and check if **any** of the conditions are true.



IF raining OR need shade:
Vega **use "umbrella"**

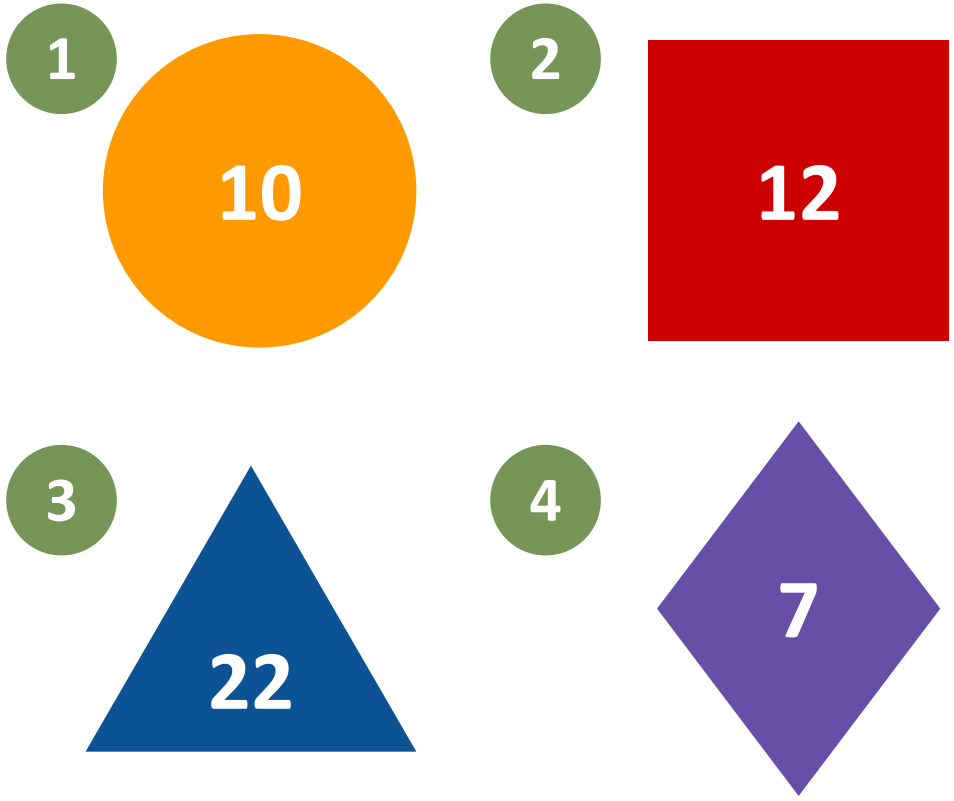
When do you use **compound conditionals** with **OR** your **everyday life**?

IF have online class OR _____:
hero

IF _____ AND _____:
hero

Human Program

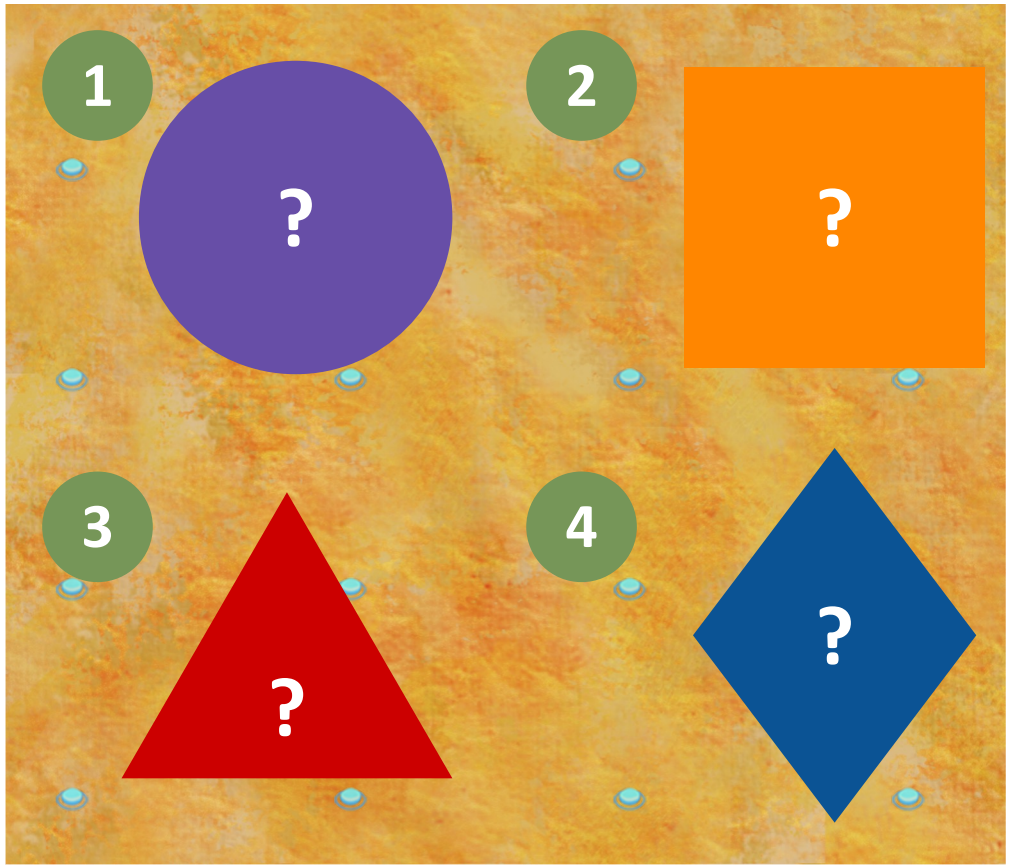
Look at the compound conditional. Then have a volunteer be the human in this program. What should the human do for each of the shapes?




```
IF number > 20 AND shapeColor NOT Red:
        human says “Kaboom!”
ELSE IF number > 9 AND shape = Circle:
        human barks
ELSE IF number <= 6 OR shape = Square:
        human patsHead
ELSE:
        human hopsBackwards
```

Concept Check: Human Program Design

Change the color and number of these 4 shapes. Then finish writing the conditional and have a volunteer act out what the human should do for each shape.

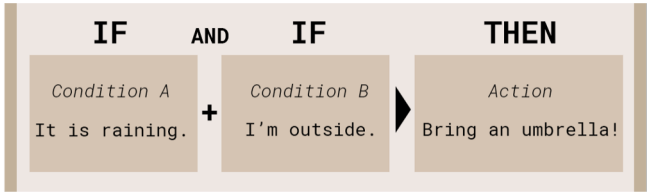


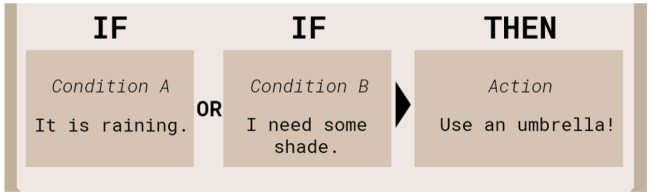
IF ???? **AND** ????:
    human ????
ELSE IF ???? **OR** ????:
    human ????
ELSE:
    human ????

PSEUDOCODE!

Syntax

Use OR to combine two or more conditions and check if **any** of the conditions are true.



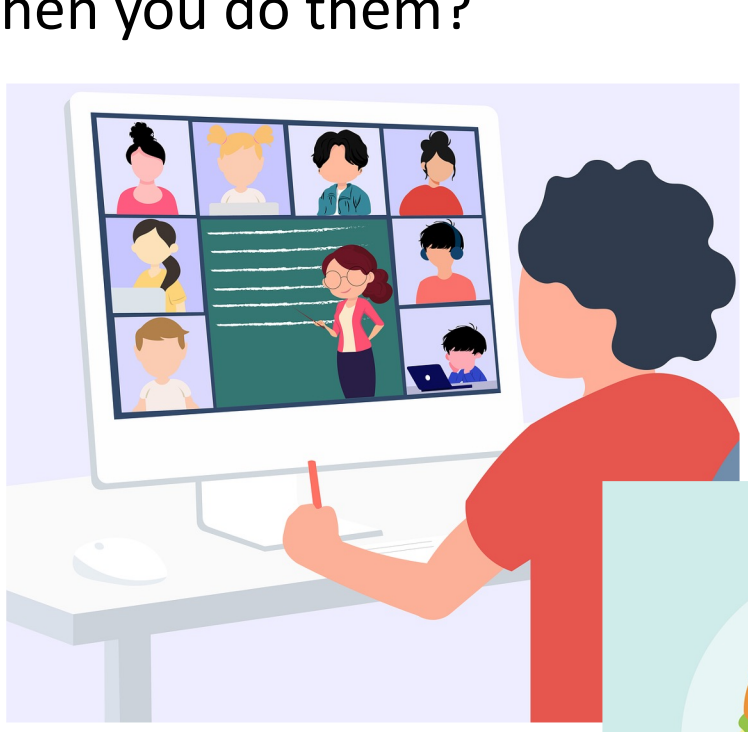


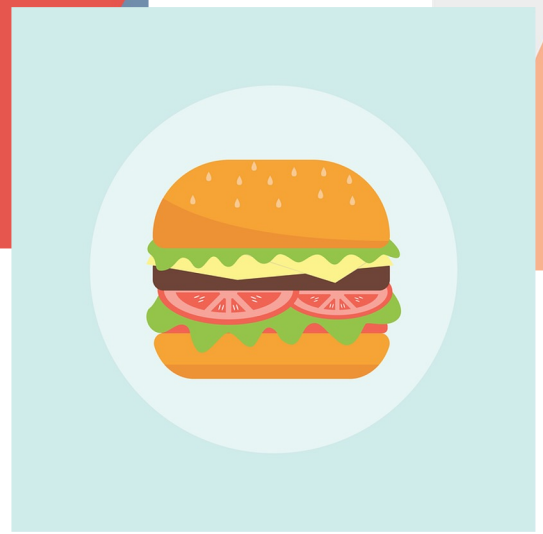
	Python	JavaScript
NOT	**not** sunny	**!**sunny
AND	raining **and** outside	(raining **&&** outside)
OR	raining **or** needShade	(raining **||** needShade)

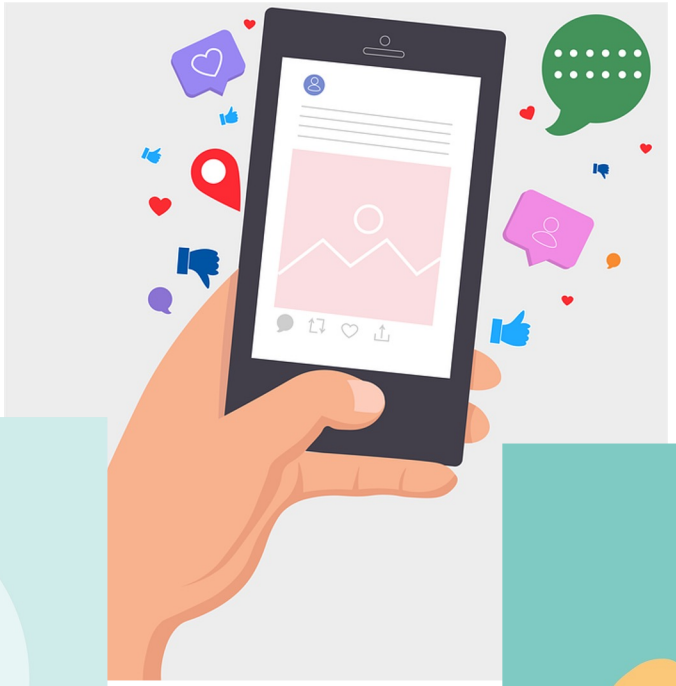
Introduction to Functions

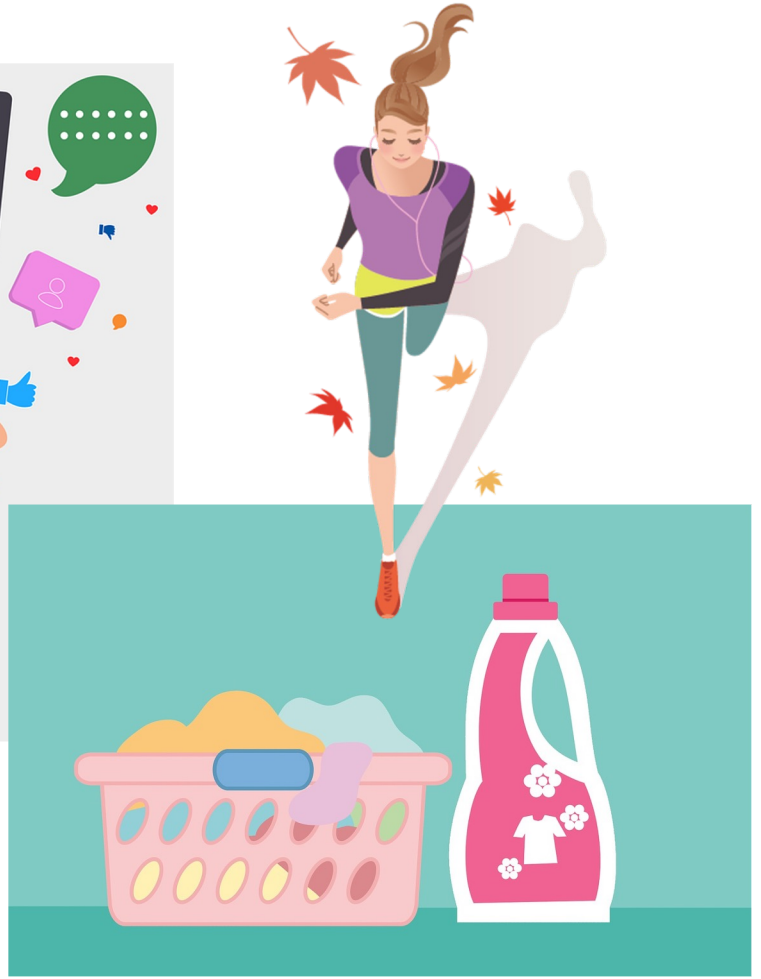
What are some of your everyday activities?

What are the steps to each of those activities? Do you have to think about each step when you do them?









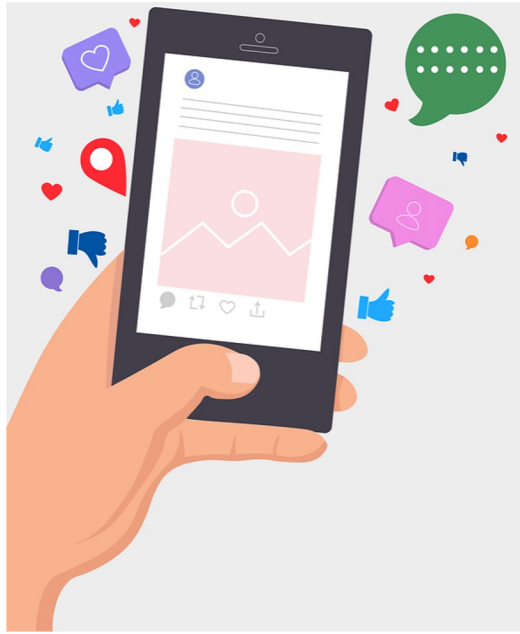
Functions in Life

Functions are shortcuts that let you simplify steps and instructions. You can put repetitive steps inside a function. Then, you can use the function instead of listing each of those steps.

FUNCTION makePost
- **OPEN** app
- **START** a new post
- **ADD** photo or video
  - **WRITE** description
  - **ADD** hashtags

Today's Social Media Activity

TAKE a photo
makePost
MESSAGE with friends
FIND funny cat video
makePost
SCROLL feed
ADD comments
SEE dog chases tail
TAKE a video
makePost

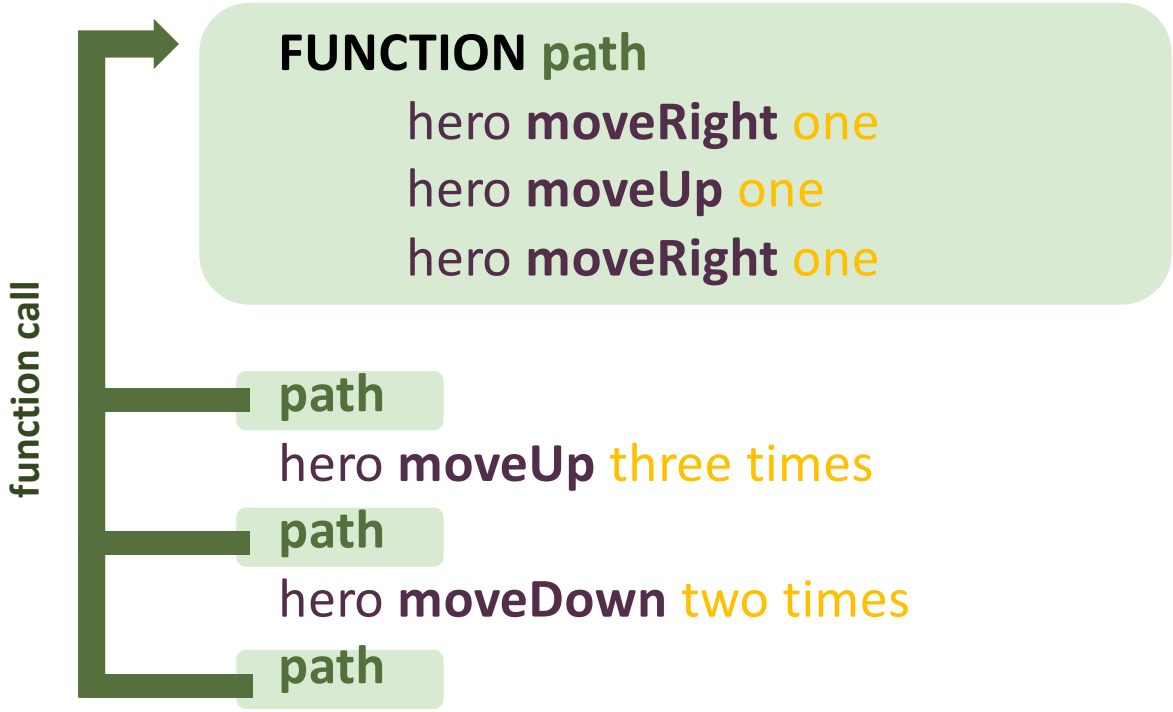


Functions in Code

A **function** contains a piece of code that you can use over and over again in your program. It provides a shortcut that helps your code stay organized and easier to read.

```
hero moveRight one
hero moveUp one
hero moveRight one
hero moveUp three times
hero moveRight one
hero moveUp one
hero moveRight one
hero moveDown two times
hero moveRight one
hero moveUp one
hero moveRight one
```

Original Program

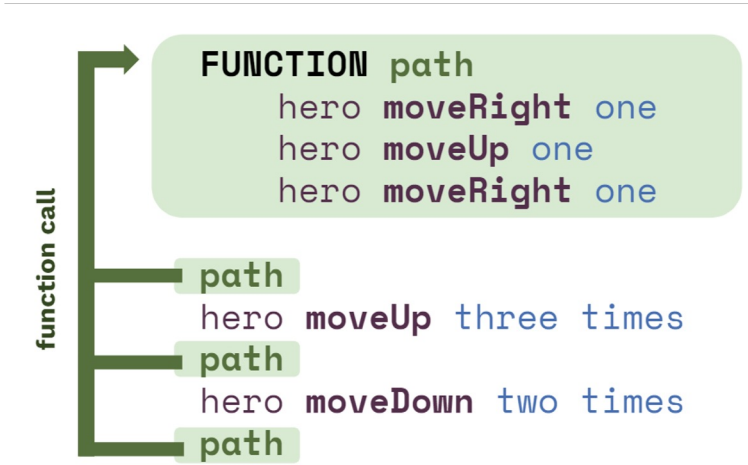


Program with a Function

Structured Programming

We are always looking for new ways to make our code faster, efficient, and error proof. We achieve this with **structured programming**.

```
hero moveRight one
hero moveUp one
hero moveRight one
hero moveUp three times
hero moveRight one
hero moveUp one
hero moveRight one
hero moveDown two times
hero moveRight one
hero moveUp one
hero moveRight one
```



Regular

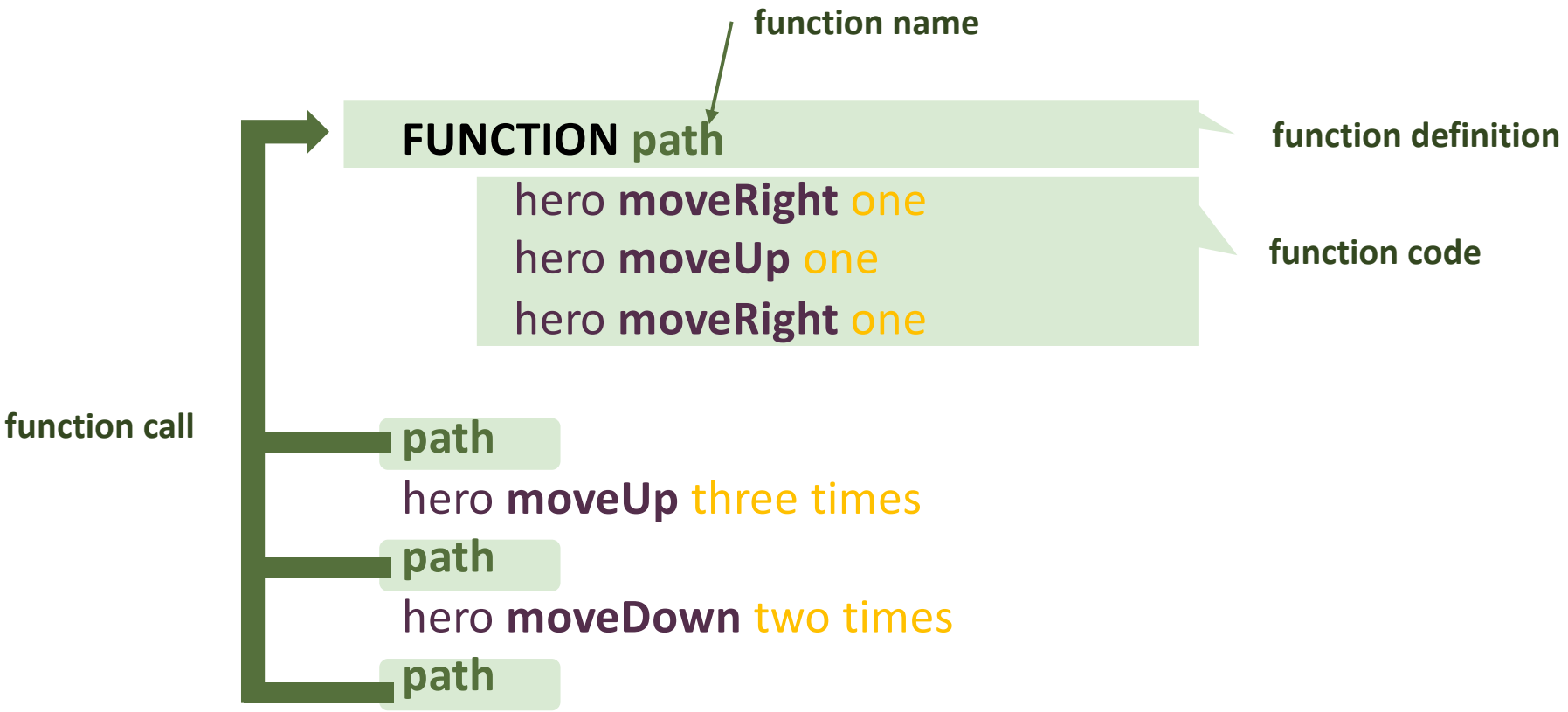
With regular programming, although it requires less planning ahead, a program is prone to errors and hard to update.

Structured

Structured programming uses reusable elements, like variables, to make a program clearer and easy to update.

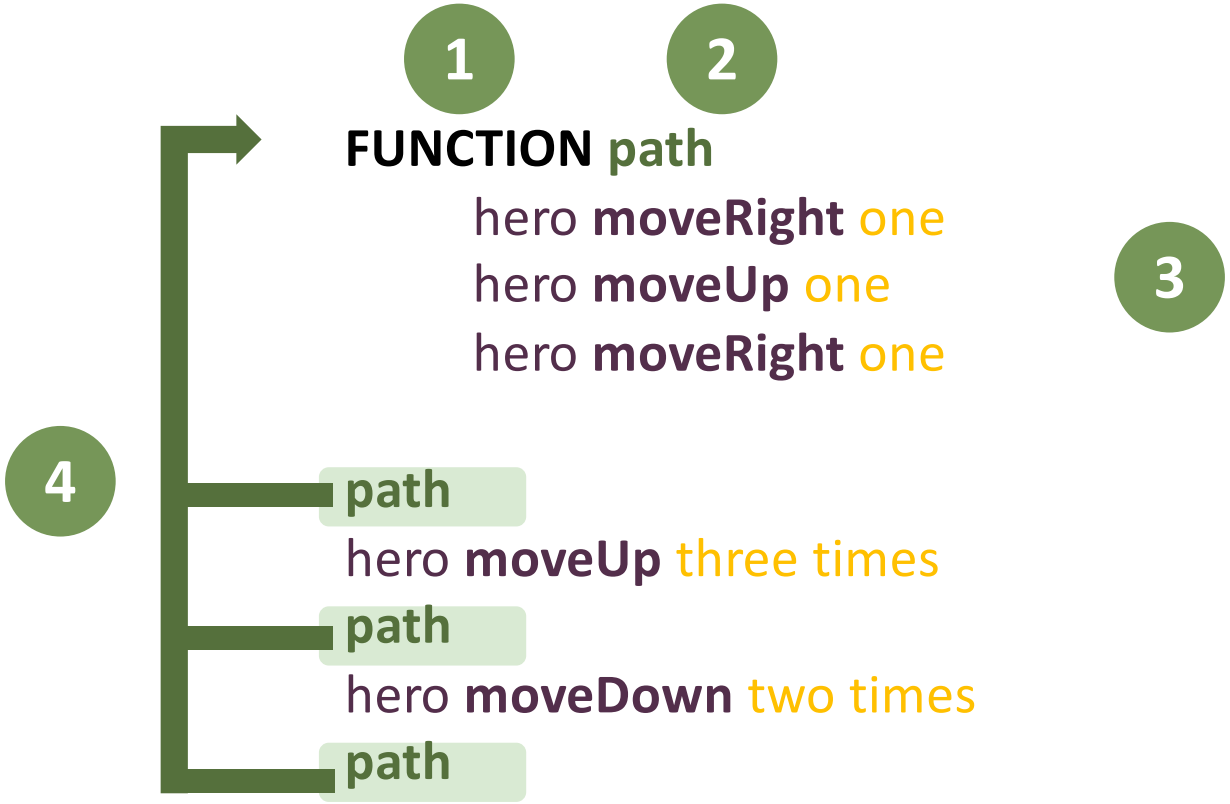
Function Parts

A **function** contains a piece of code that you can use over and over again in your program. It provides a shortcut that helps your code stay organized and easier to read.



Concept Check: Function Parts

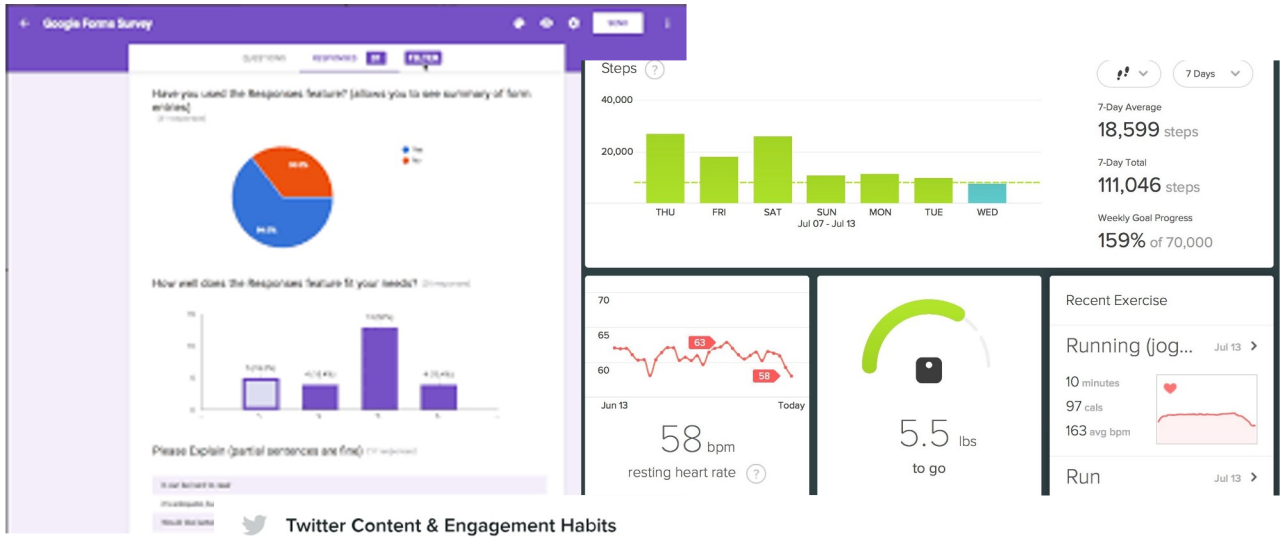
Identify the different parts of this program

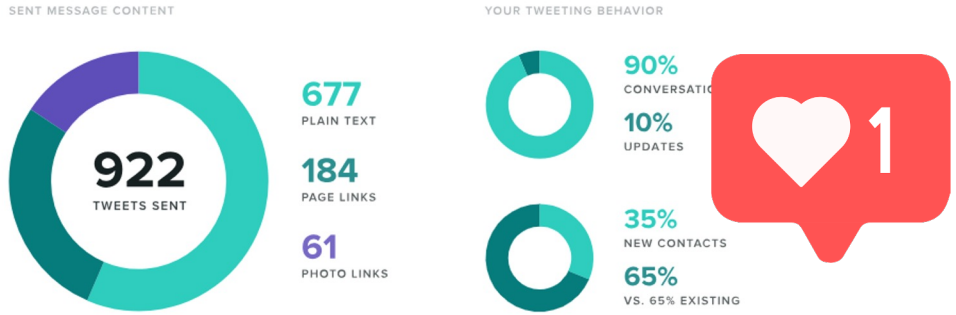


Data & Analysis

What kind of data have you collected in the past?

Have you collected data through surveys? Spreadsheets? Online tools?





How does data help you make decisions?

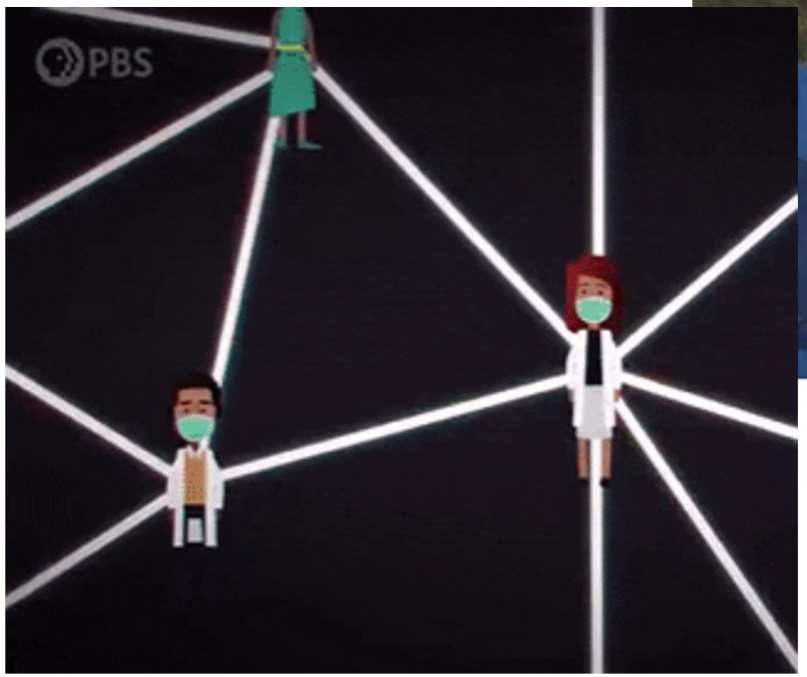
How have you used data to help you make predictions and decisions in your life or in past projects?

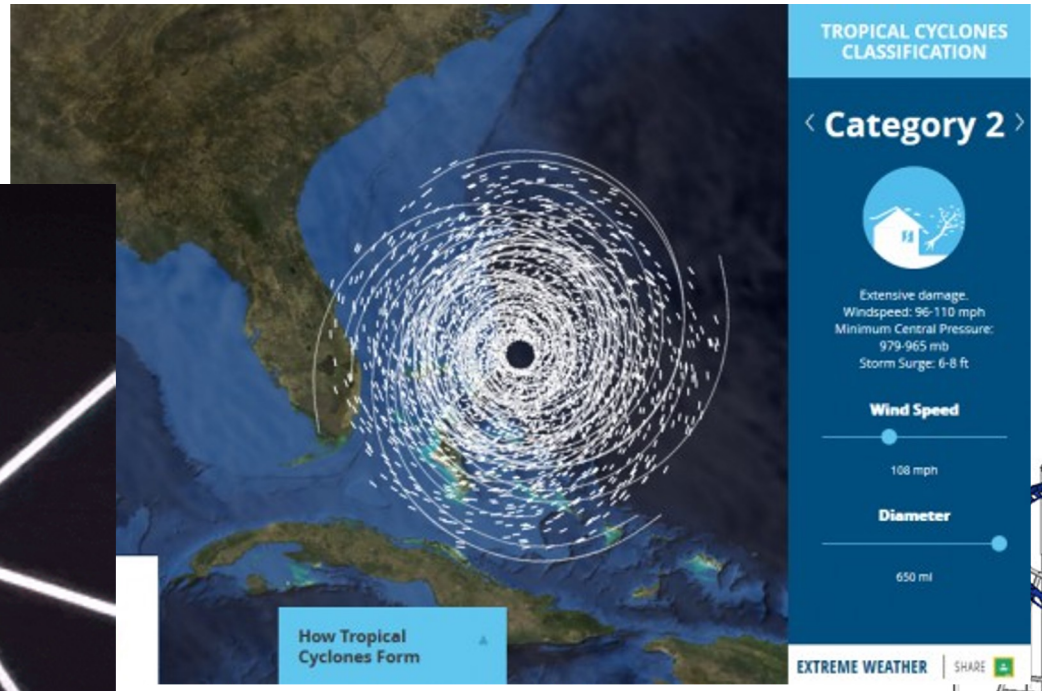
Data can help you:

- Predict if it’s going to rain
- Plan out a marketing plan
- Make design decisions for your game
- ?????
- ?????
- ????

Computational Models

Computational models use **mathematics** and **computer science** to **simulate complex systems**. You can change the **settings** of a model and then see how they cause **different outcomes.** Then you can use that **data** to make **predictions** and **decisions**.



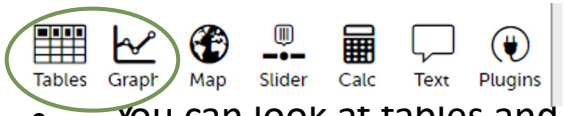


Rock, Paper Scissors!

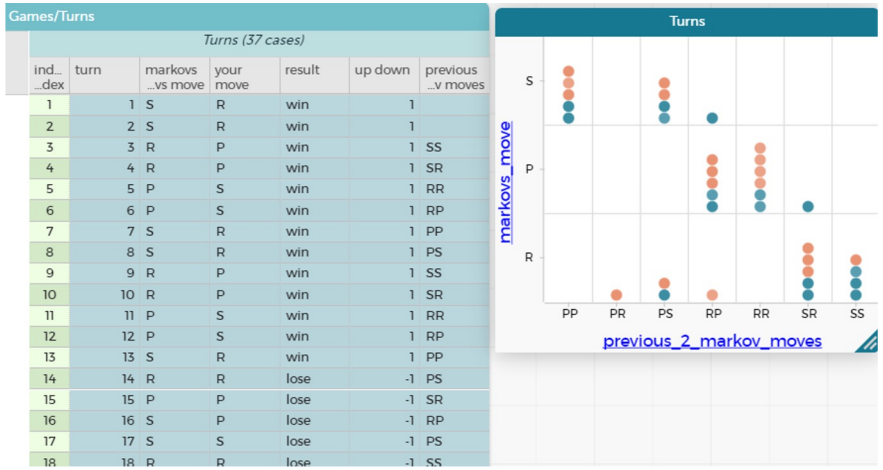
Try out a computational model and review the data that you collect.

- Watch the tutorial and learn how to play Rock, Paper, Scissors against Markov. Then start a game.

's previous moves to figure out a pattern in his game strategy

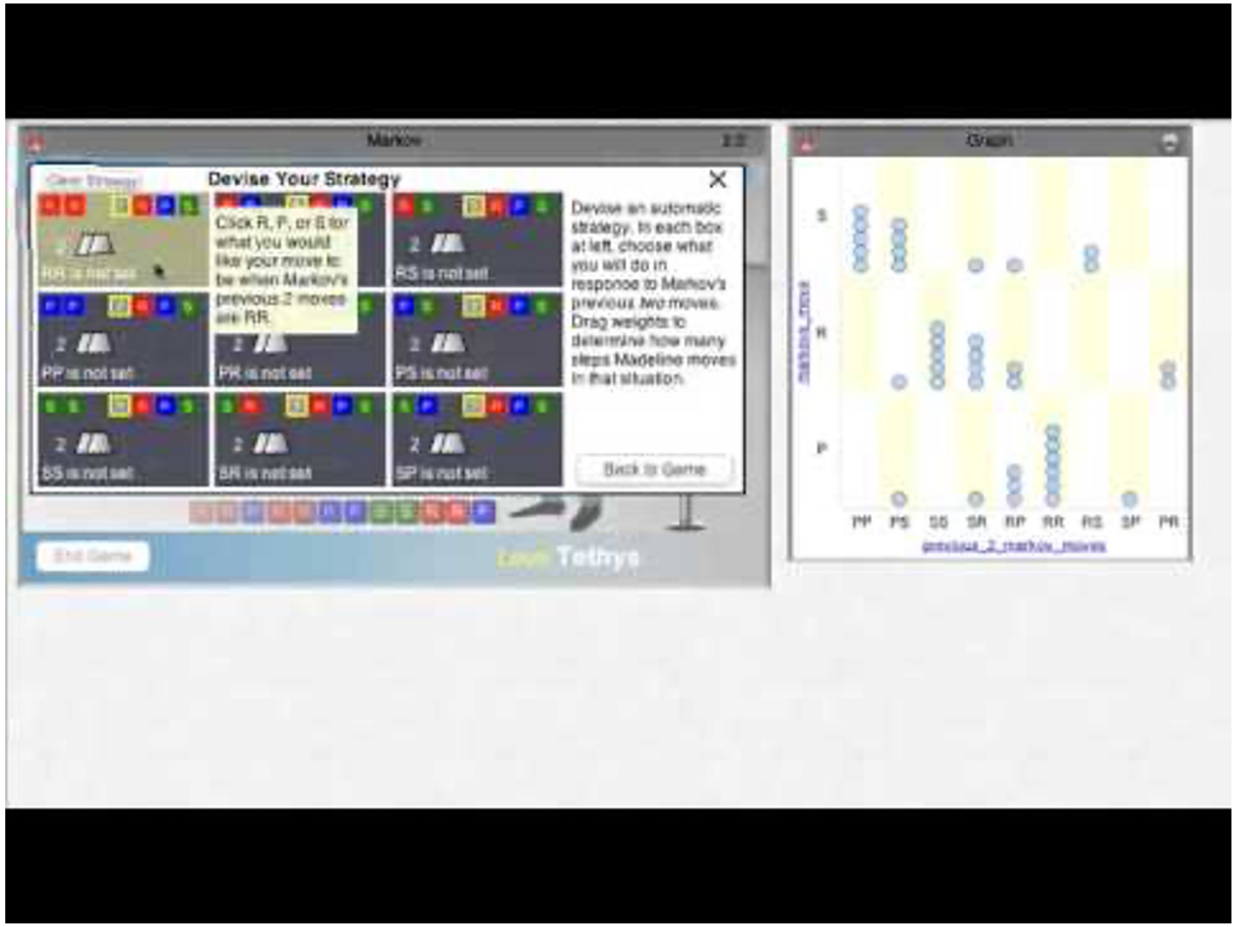


- You can look at tables and graphs to help you find the pattern

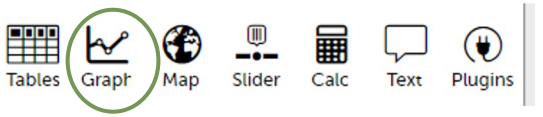


Concept Check: Advanced Rock, Paper Scissors

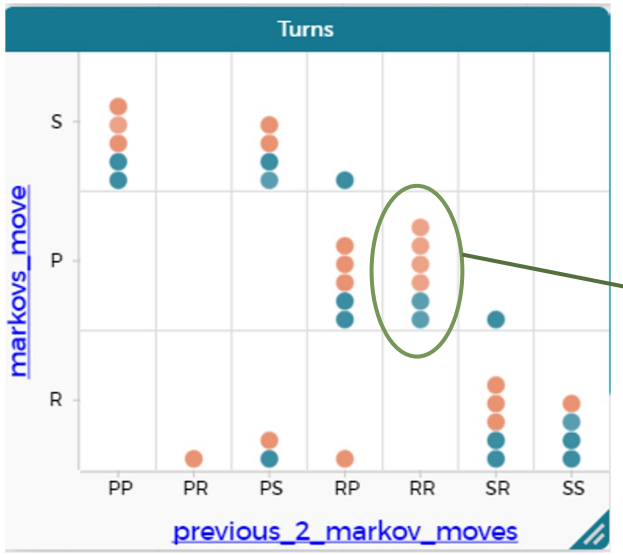
Use data to set up a **strategy** and improve your odds of winning!



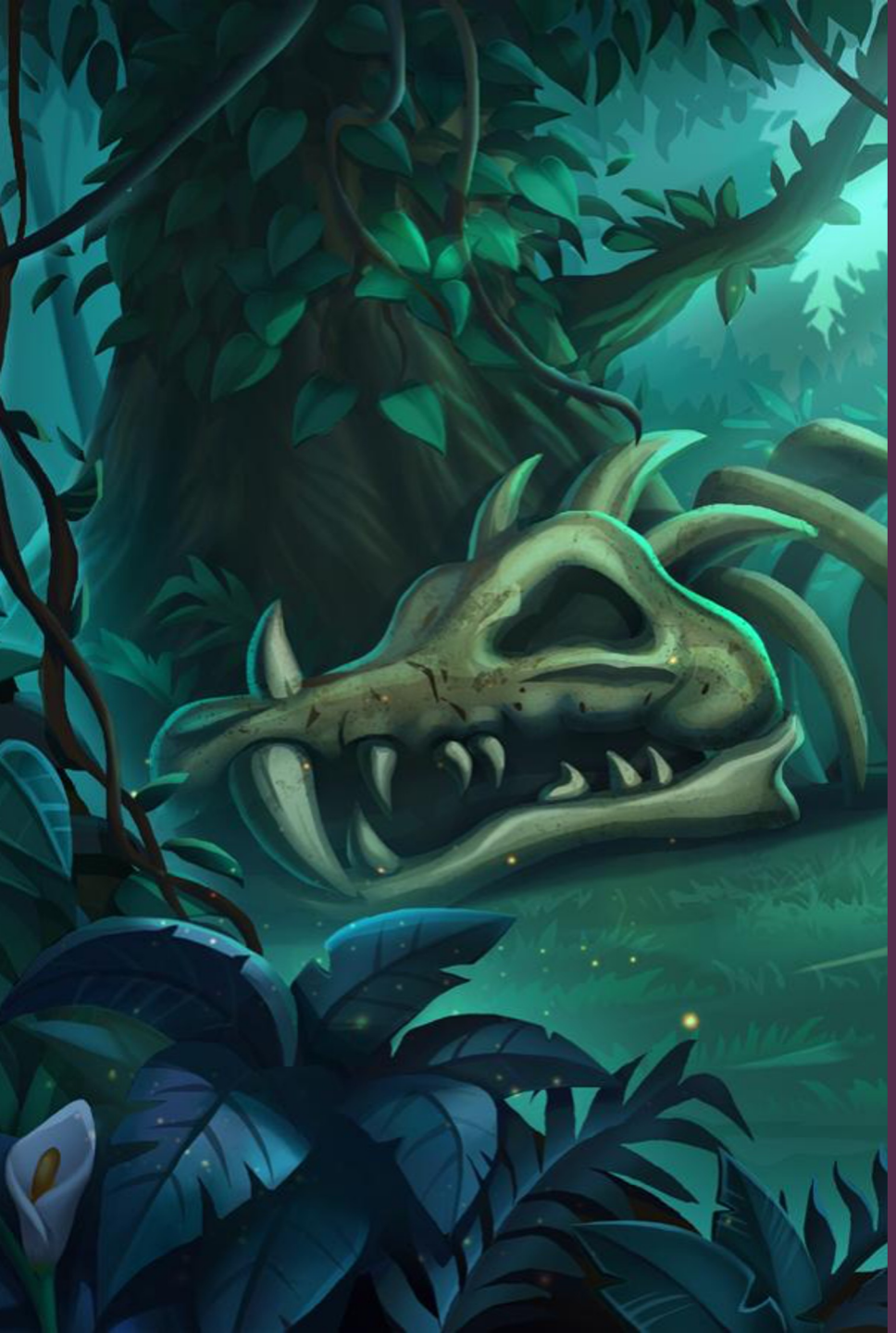
- Watch the advanced tutorial and learn how to set your own strategy.



- Open the graph and make sure the table is set with the settings below. Then see if you can notice a **pattern.**



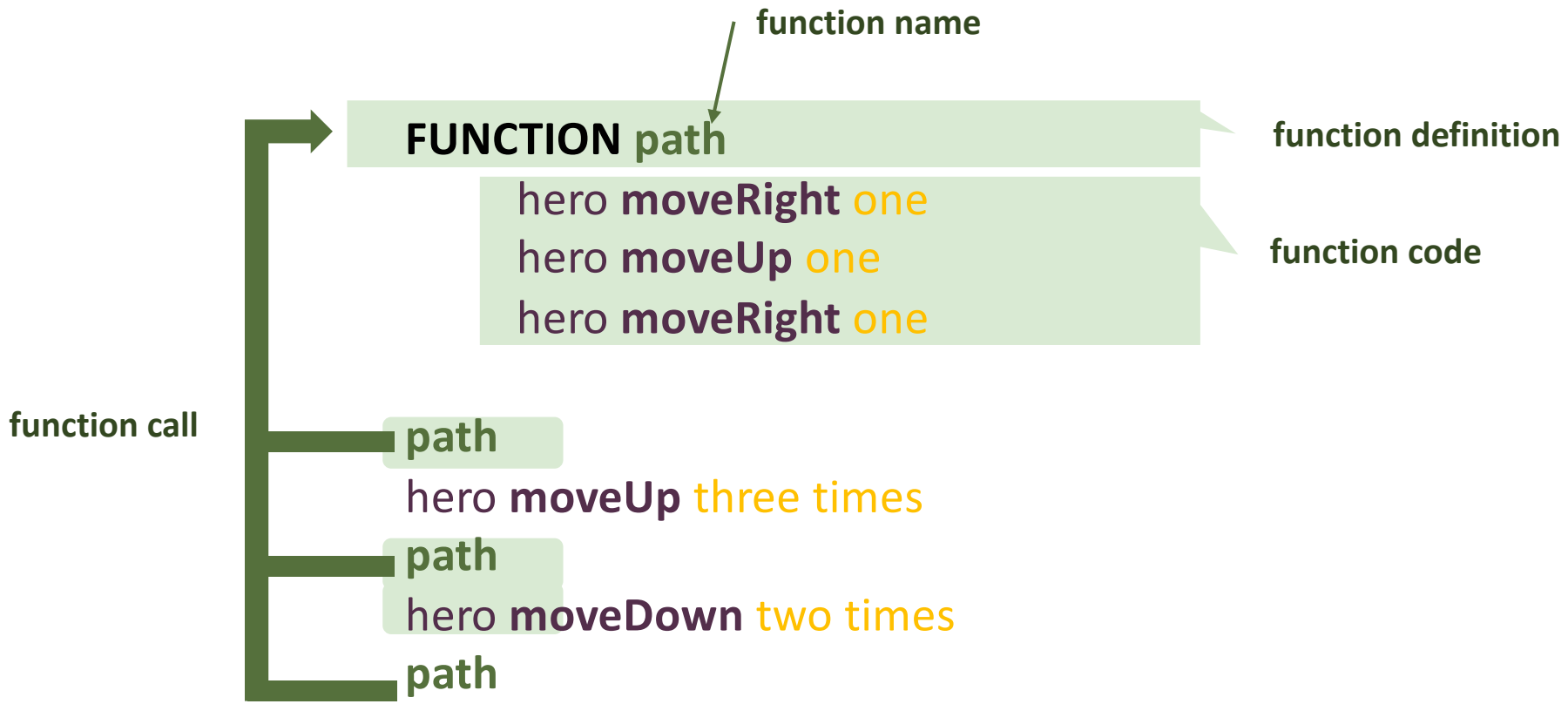
Everytime, Markov plays **rock and rock**, then he always **follows with paper**. What **other patterns** do you see?



Writing Functions

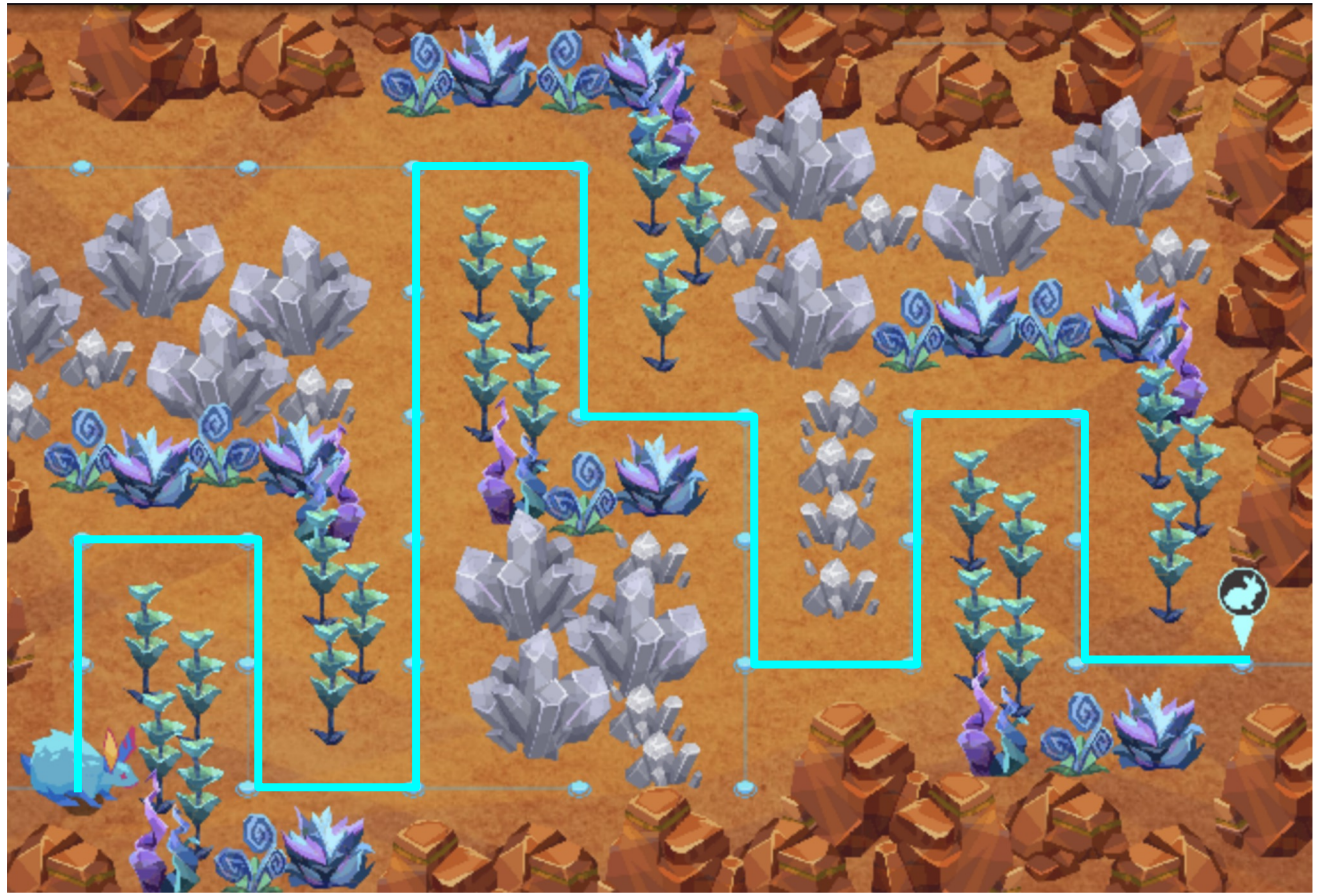
Function Review

A **function** contains a piece of code that you can use over and over again in your program. It provides a shortcut that helps your code stay organized and easier to read.



Writing Functions

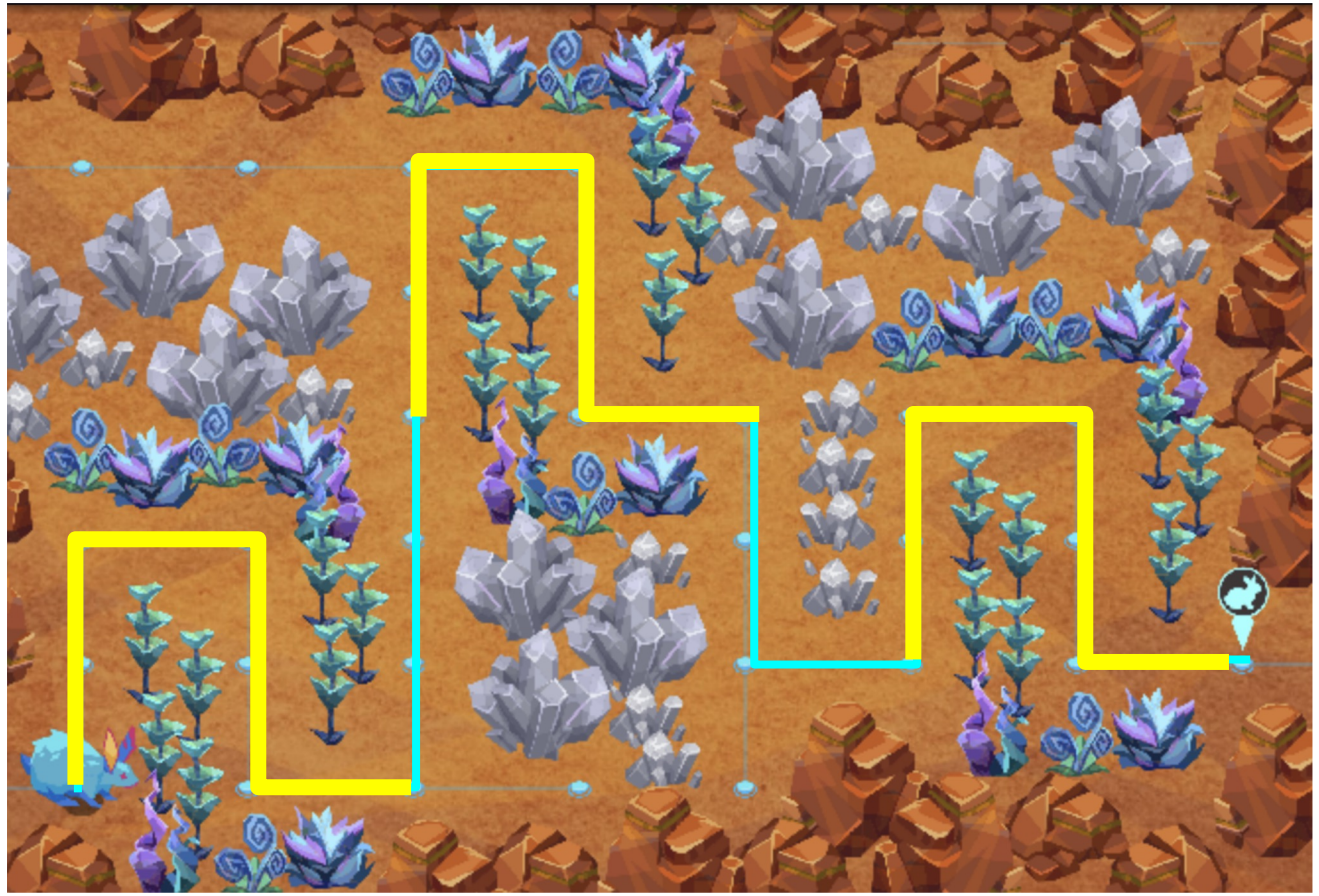
How do you decide what code should go inside your function?



- Trace the path that you need to take
- Identify any repeated movement patterns in the path

Writing Functions

The **repeated movement pattern** is the code that should go inside the function.



```
FUNCTION path
    noodles moveUp two
    noodles moveRight one
    noodles moveDown two
    noodles moveRight one
```

```
path
noodles moveUp three
path
noodles moveDown two
noodles moveRight one
path
```

Function Syntax: Python

Things to remember when defining and writing functions.

```
def crossRiver():
    hero.jumpRight()
    mouse.moveRight(2)

crossRiver()
hero.moveUp(2)
crossRiver()
```

- **def** - used to start the definition of the function
- **crossRiver** - choose a name for your function that will help you remember what it can do
- **()** - make sure you put () after the name of your function when you define it
- **:** - the definition line needs to end with a colon
- make sure your function code is indented

Function Syntax: JavaScript

Things to remember when defining and writing functions.

```
function crossRiver(){
    hero.jumpRight();
    mouse.moveRight(2);
}
crossRiver();
hero.moveUp(2);
crossRiver();
```

- **function** - used to start the definition of the function
- **crossRiver** - choose a name for your function that will help you remember what it can do
- **()** - make sure you put () after the name of your function when you define it
- **{}** - the function code needs to start and end with brackets (just like loops & conditionals)
- make sure your function code is indented

Concept Check: Functions with Mouse

Define and write the ladder() function to help Mouse get to the exit.

```
# define and write the ladder function here

ladder()
mouse.moveRight()
ladder()
mouse.moveUp()
```