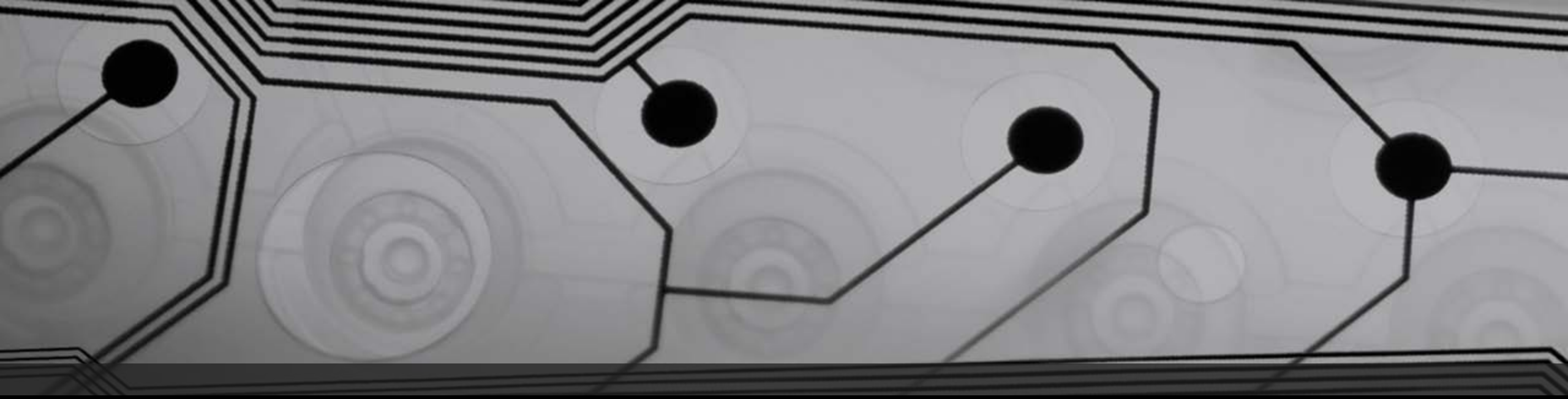


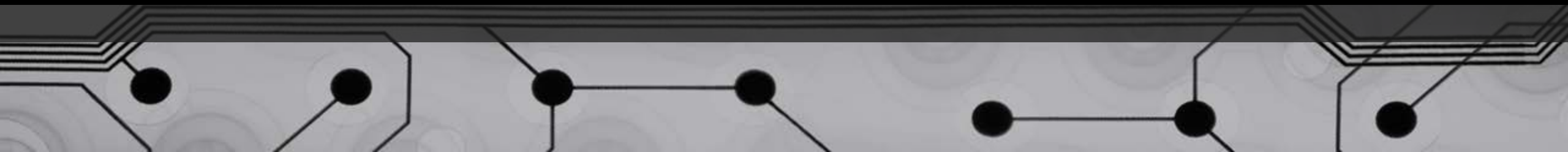
SCRATCH with a dash of Python

Panhandle STEM Conference – July 23-24, 2018

Presented by:

H. Paul Haiduk
Computer Science Coordinator
West Texas A&M University
hhaiduk@wtamu.edu

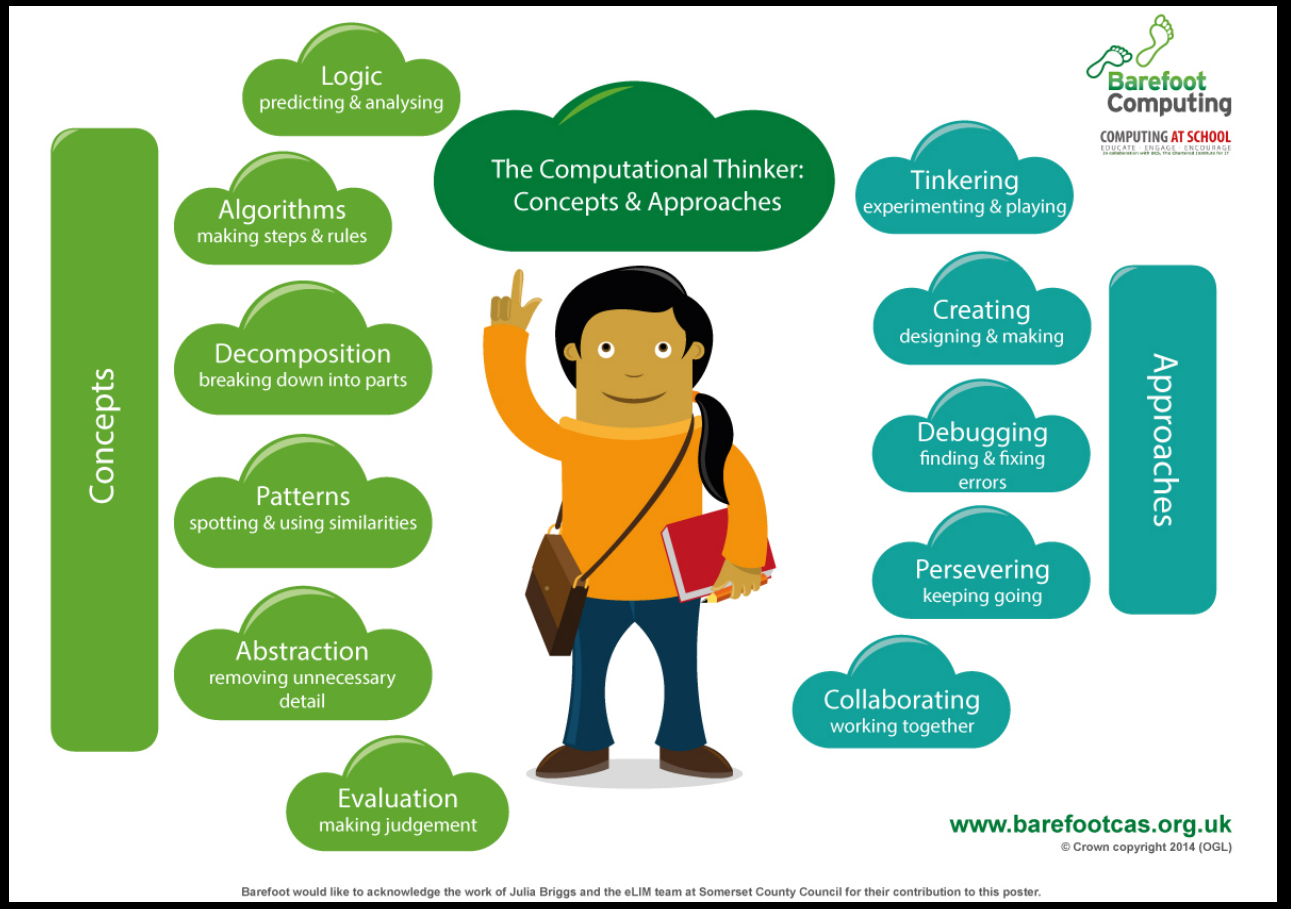
David Kossey
Computer Science/Robotics/Math
West Texas High School
david.kossey@pspcisd.net



Objectives

- Introduce participants to use of a “block-based” approach to implementing algorithms
- Participants will be able to use Computational Thinking to solve problems
- Participants will be able to apply Computer Science Principals to write simple/complex programs
- But . . . First things first. Navigate your browser to https://scratch.mit.edu and join if you have not already done so. Otherwise, login so that we can begin our work together.

Computational Thinking



Computational Thinking Definition

- Computational Thinking is the thought processes involved in formulation of problems and their solutions so that the solutions are represented in a form that can be effectivity carried out by an information-processing agent (Cuny, Snyder, Wing, 2010)

Programming Control Structure

- Sequence of Steps
- Selection/decision/IF statement
- Repetition/loop/iteration

Learning to program:
Difficulties for beginners

Syntax errors

- struggle for hours to fix syntax errors
- Loose confidence
- Frustrating experience
- Run away & never come back if possible!

Why start with block-based?

- Visual Programming Tools like Scratch & Trinket use drag-and-drop programming and enable us to master programming concepts.

- Programs are always ready to run since there are no syntax errors. In other words, they enable us to focus on the logic first & build confidence.

- Following is a meaningful approach:
  - Scratch→ Trinket (Block/Text Based)→Text based Thunder the Robot → Text based Python

What is Scratch

- Scratch is a visual programming environment where you design/create your interactive stories, games & animations. While Scratch has been designed with a typical 3rd or 4th grader in mind, people of all ages can use it to learn the basics of programming in an enjoyable way!
- Scratch involves drag and drop of various blocks together to write programs & such an environment enables the users to focus on the logic & enjoy the learning experience

What is Scratch

- Unlike most Python/Java development environments, there is NO free-form typing, so there are no syntax errors!

- Scratch is not limited to young people though - people of all ages with no programming experience can enjoy its simplicity and learn the basics of programming!

Getting Started!

- Home: http://scratch.mit.edu/
- Each object in Scratch is called a sprite. Default sprite is a cat.
- The background for the sprite is called Stage OR backdrop.
- Scripts can be dragged and placed/appended on to the grey workspace (towards right). Each group of scripts is run by double-clicking the first block of script OR by triggering the event.
- New sprites and backdrops can be added. Each sprite and backdrop is associated with respective sequence of scripts.

Getting Started!

- Home: http://scratch.mit.edu/
- Each object in Scratch is called a sprite. Default sprite is a cat.
- The background for the sprite is called Stage OR backdrop.
- Scripts can be dragged and placed/appended on to the grey workspace (towards right). Each group of scripts is run by double-clicking the first block of script OR by triggering the event.
- New sprites and backdrops can be added. Each sprite and backdrop is associated with respective sequence of scripts.

Getting Started!

Some other Scratch educator resources:

- Visit the ScratchEd website, a community of educators that help each other learn and use Scratch. You can find lessons, activities, project ideas, or simply have your questions answered by a friendly fellow educator.
- Creative Computing Workshop is a free online workshop where you can learn more about using Scratch and supporting computational thinking.
- Scratch Day is a worldwide network of gatherings, where Scratchers meet up, share projects and experiences, and learn more about Scratch. Great for kids and adults!
- The book ***Learn to Program With Scratch: A Visual Introduction to programming with Games, Art, Science, and Math*** *© 2014 by No Starch Press – in 7th printing*

SCRATCH INTERFACE

Variables

- A program can work with a value, e.g. 1, 3.14, “Hello World!”

- Variables - instead of working with a specific value, we give a ***name*** to the value and manipulate the value by referencing the ***name.***

- Variables are one of the most powerful features of programming languages. Allowing us to store and manipulate information.

Types of Variable

- Every variable has a data type: a classification identifying what kind of information it contains–such as floats, integers, or strings.

- Common Data Types:
  - int: integer
  - float: floating point number (numbers with decimals)
  - str: string (text)

Variables

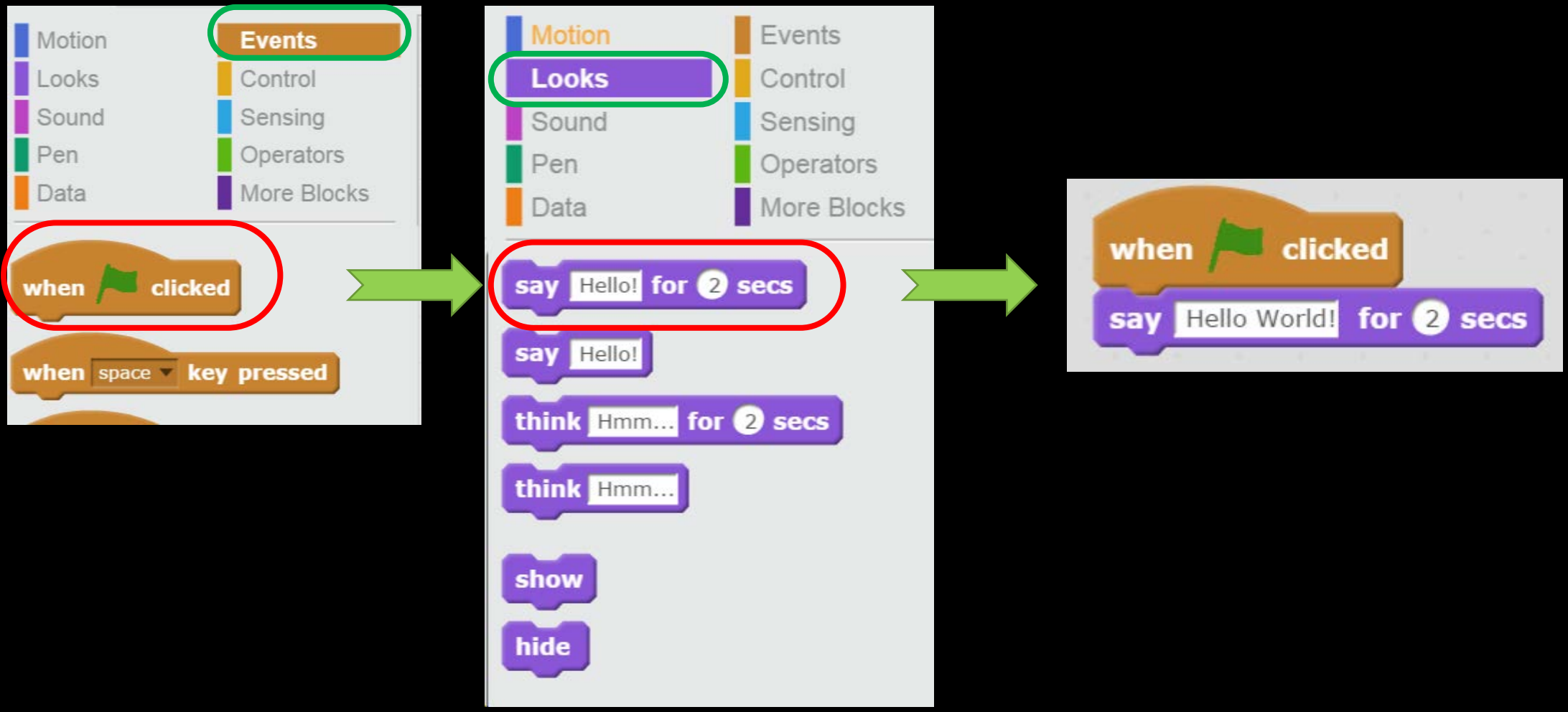
- Examples:

  - message = “Hello World!”
  - n = 17
  - pi = 3.14159265535

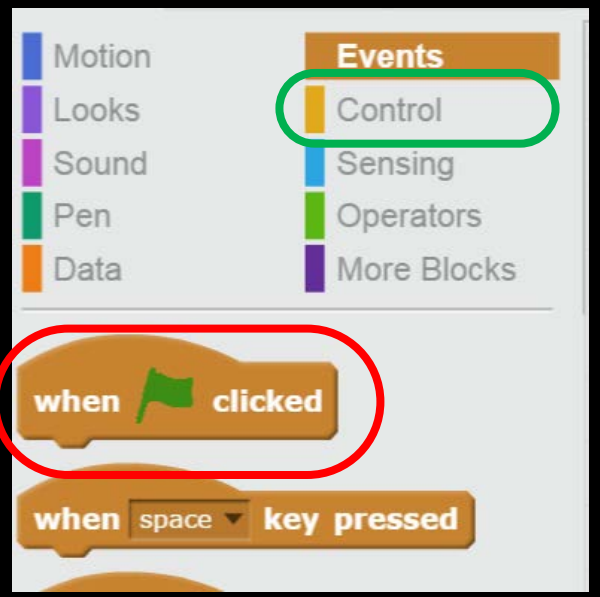
Variable Names & Keywords

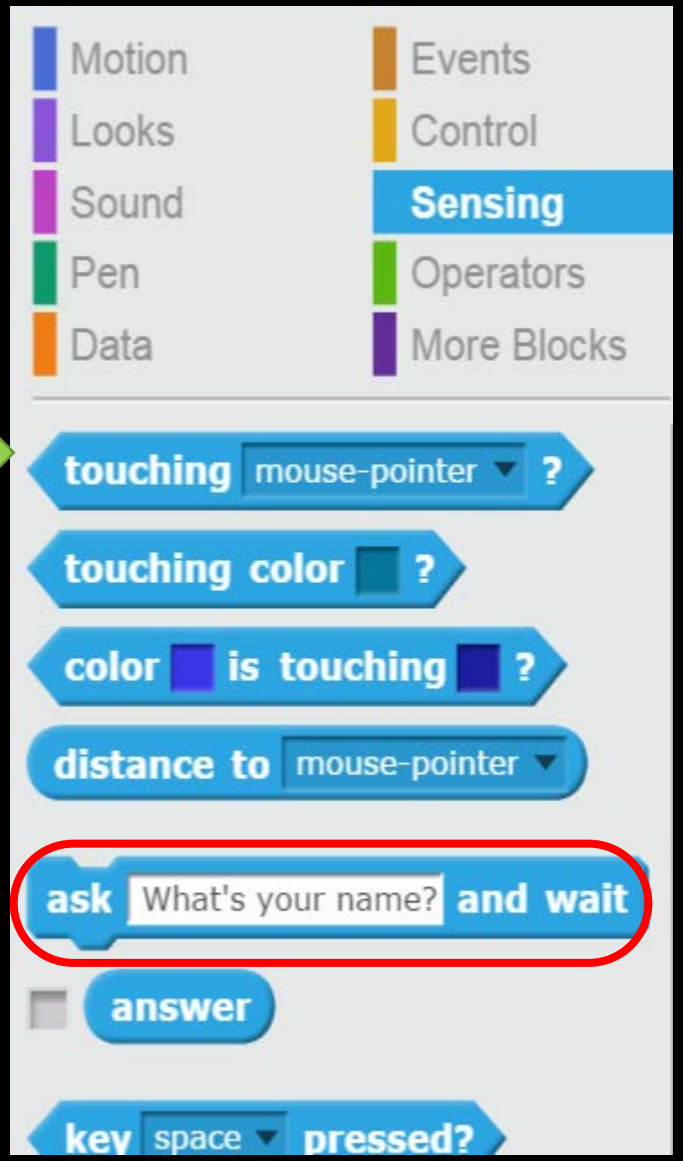
- Conventions for naming your variable:
  - **Descriptive**: document what the variable is used for
  - **Begins with a letter:** preferred to use lower case letter
  - Use **underscore** to separate multiple words - e.g. first_name, heart_rate (called snake case)
  - **Try to be concise:** variable names can be arbitrary long though it harms the readability of the program
  - Do not use **reserved words** (Keywords), e.g. *and, break, else, if.*

Hello World on Scratch

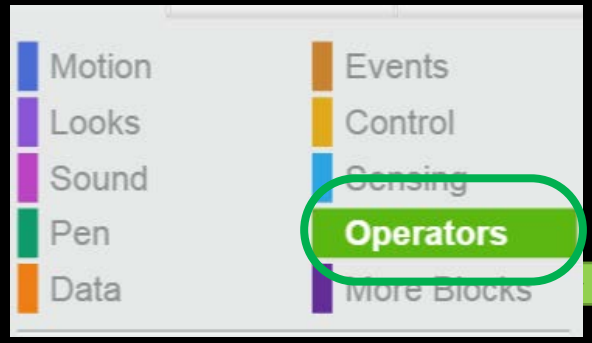


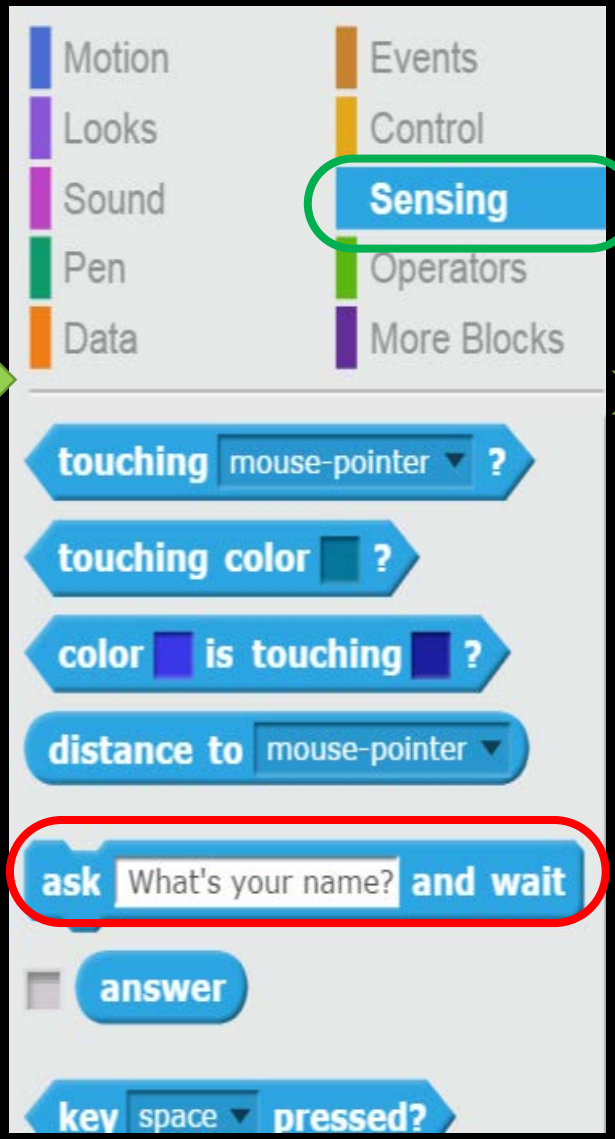
Hello Name on Scratch





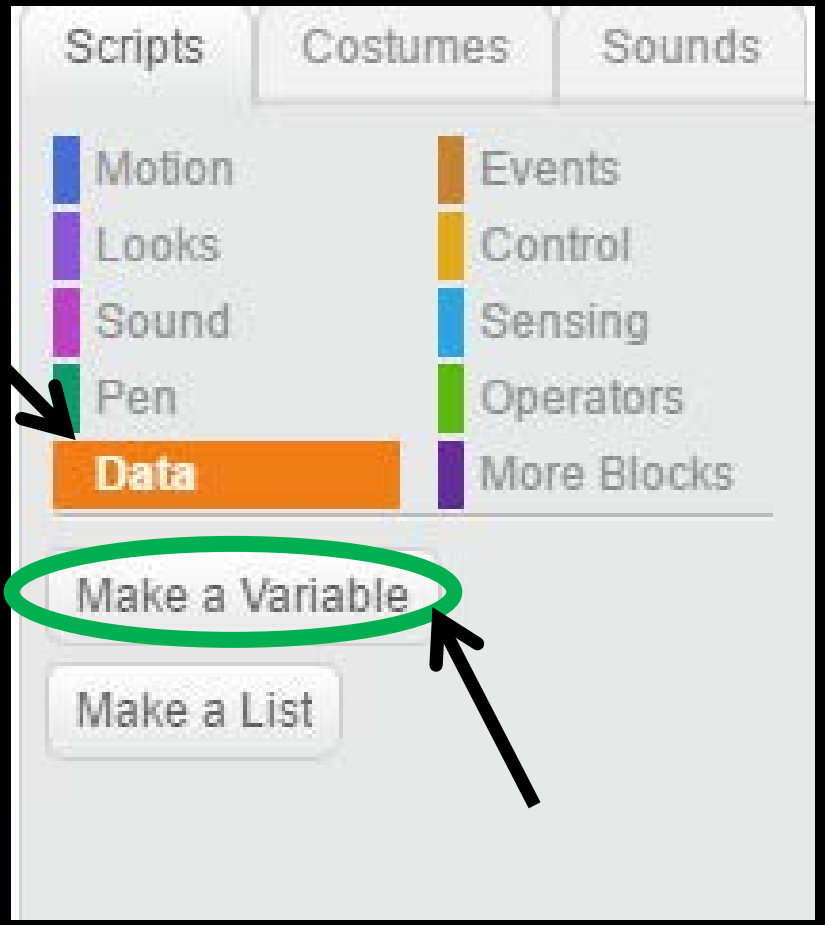
Hello Name on Scratch





Hello to you in Scratch

Making Variables

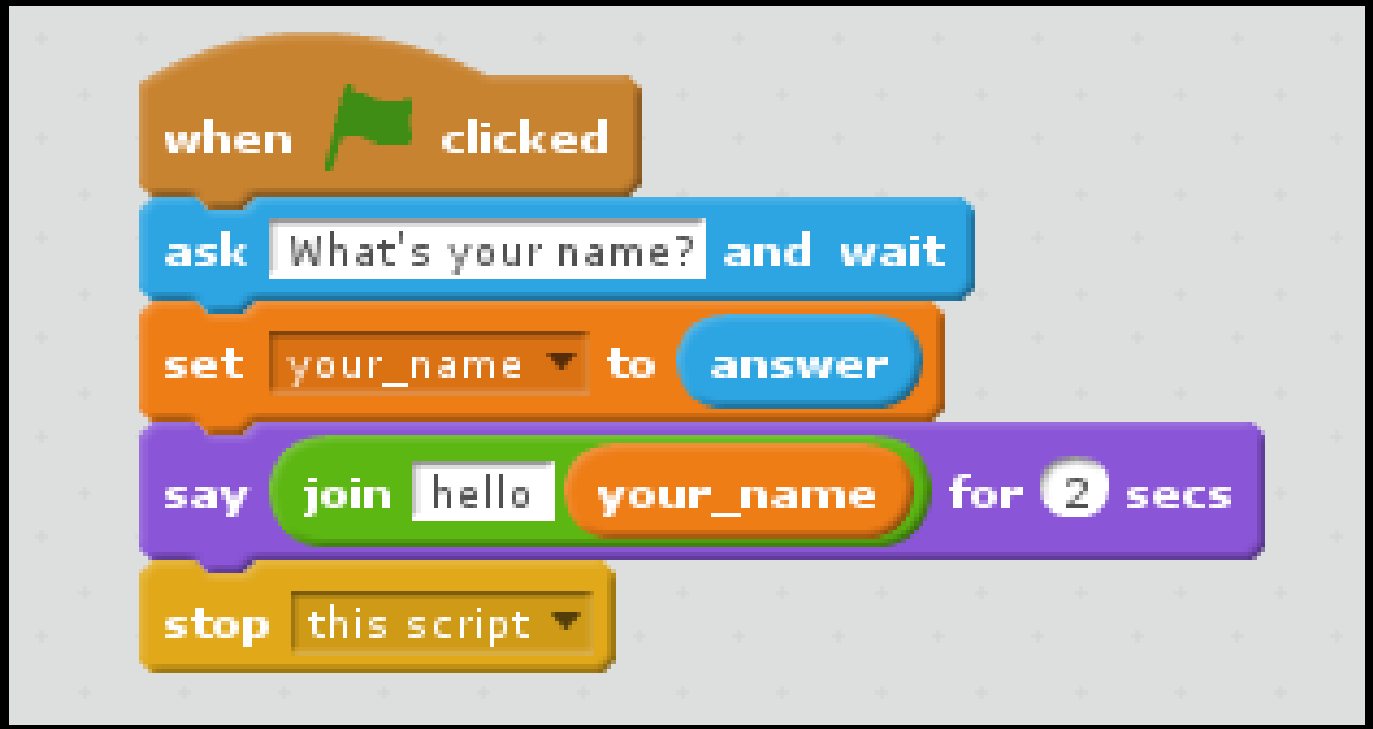


- To make a variable, go to the "Data" menu and select "Make a Variable"
- Enter the name of your variable in the pop up window.
- Press Enter!

Using Variables

- To use a variable in a program
  - Navigate to the “Data” tab
  - Grab the orange block with your variable’s name on it
  - Place the orange block anywhere in your program that you would normally use a value

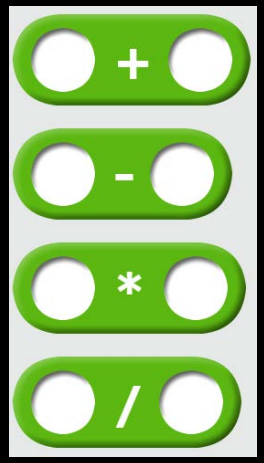
Say Hello to your_name

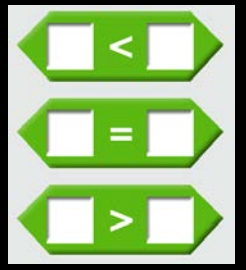


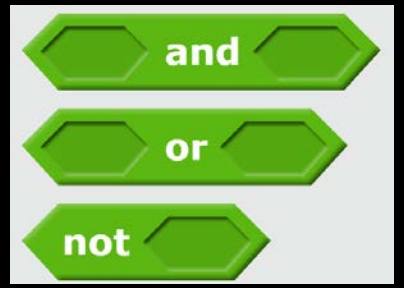
Operators

- Operators are the constructs which can manipulate values or values of variables or expressions (operands).

- Types of Operators
  - Arithmetic Operators
  - Relational (Comparison) Operators
  - Assignment Operators
  - Logical Operators
  - Membership Operators
  - Identity Operators







Expressions

- Now let's try to combine variables and operators:

An ***expression*** is a combination of values, operators, variables, and expressions that evaluates to a single value.

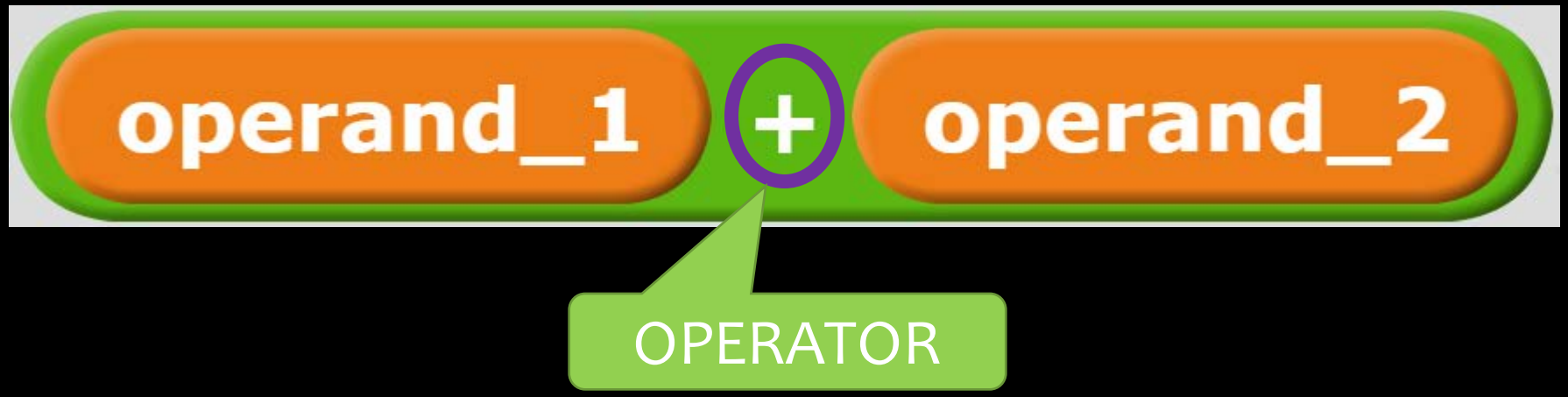
- For example: If speed is 50, what does ((speed / 2) + 10) evaluate to?

Operators vs Operands

operator: special symbol represents computation

operand: values/variables/expressions the operator is applied to

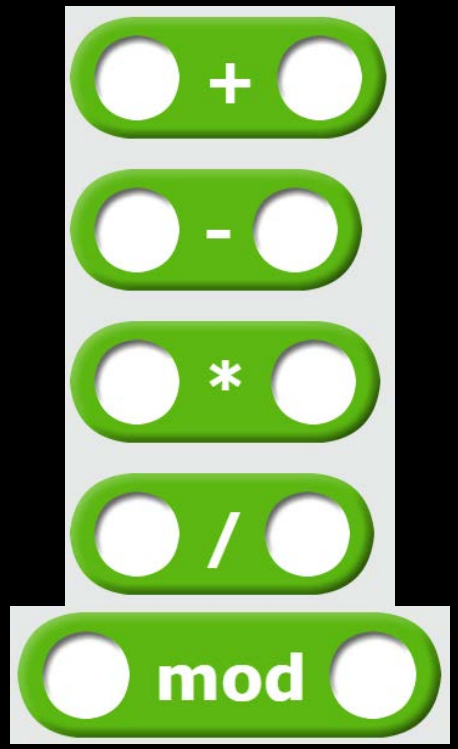
Operands can be fixed values, variables or even expressions!

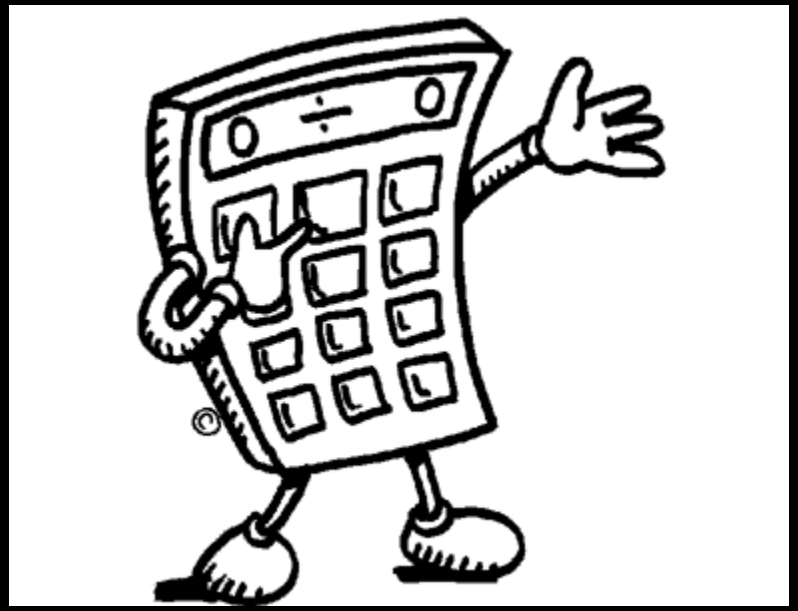


Arithmetic Operators

Arithmetic Operators work like normal arithmetic.

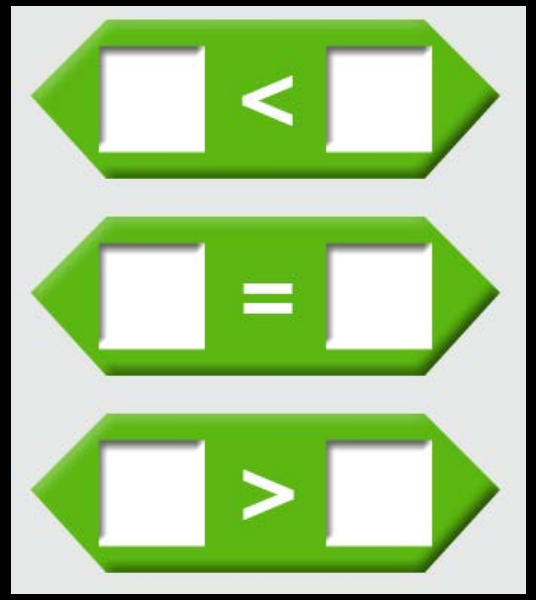
When evaluated, they return values.

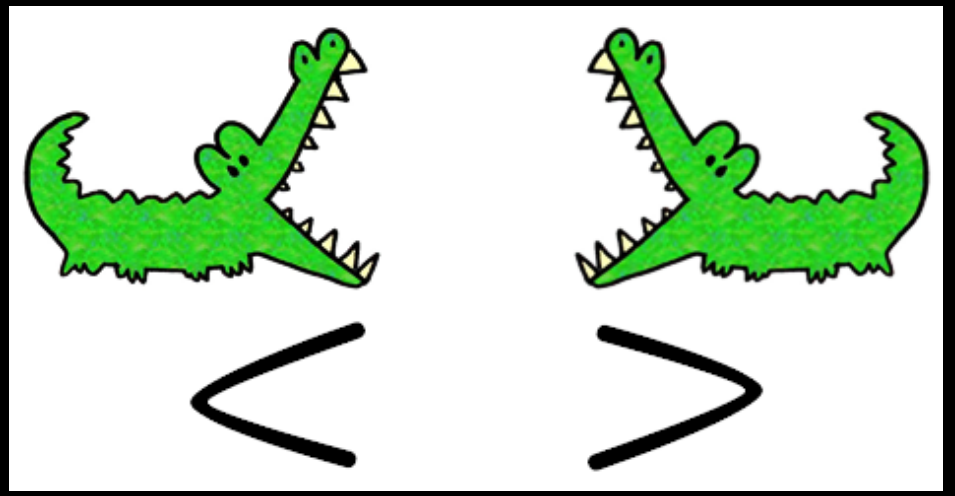




Comparison Operators

Comparison Operators are either true or false, depending on the operands.

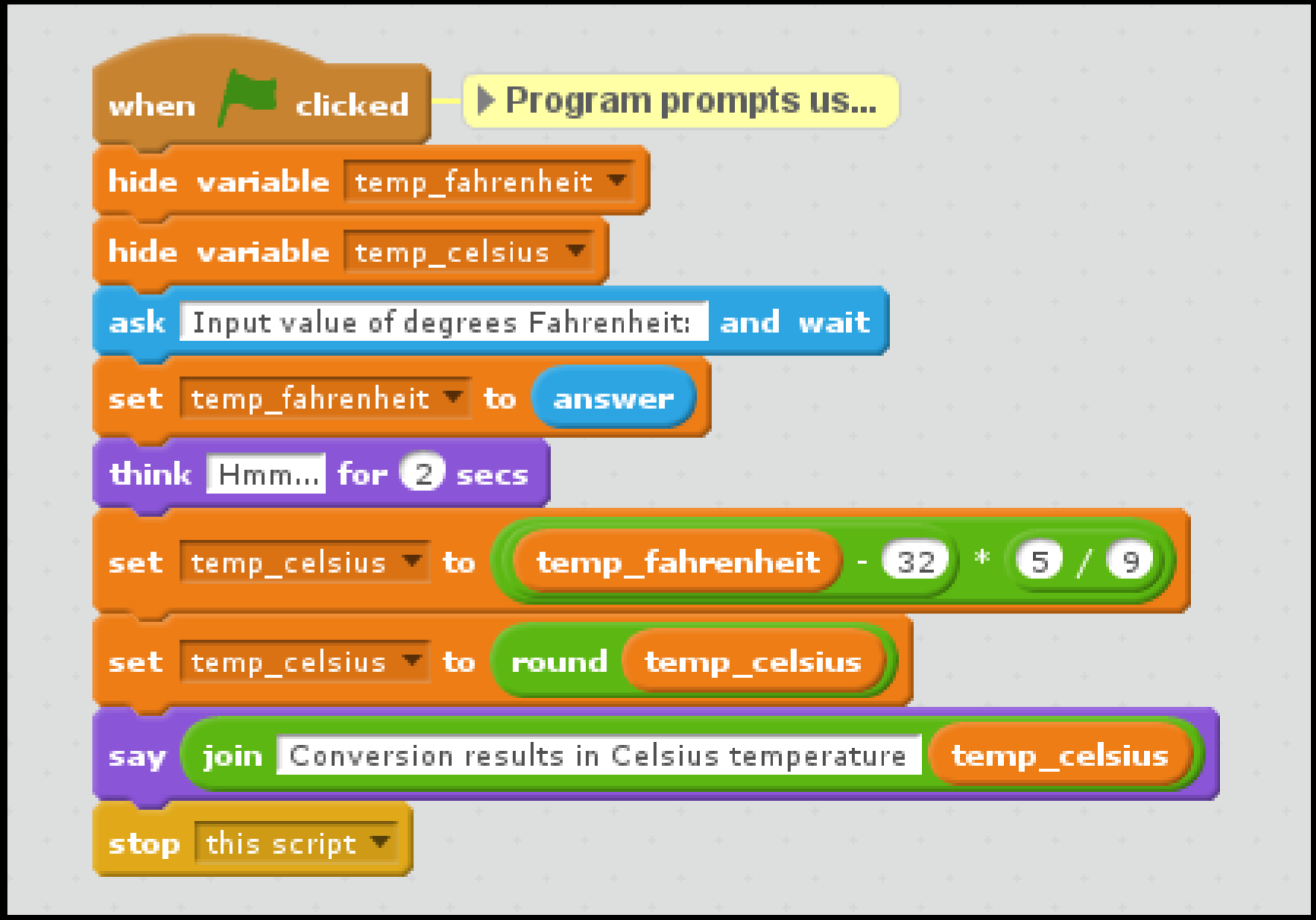




Your turn!

Make a Scratch program that converts temperatures from Fahrenheit to Celsius.

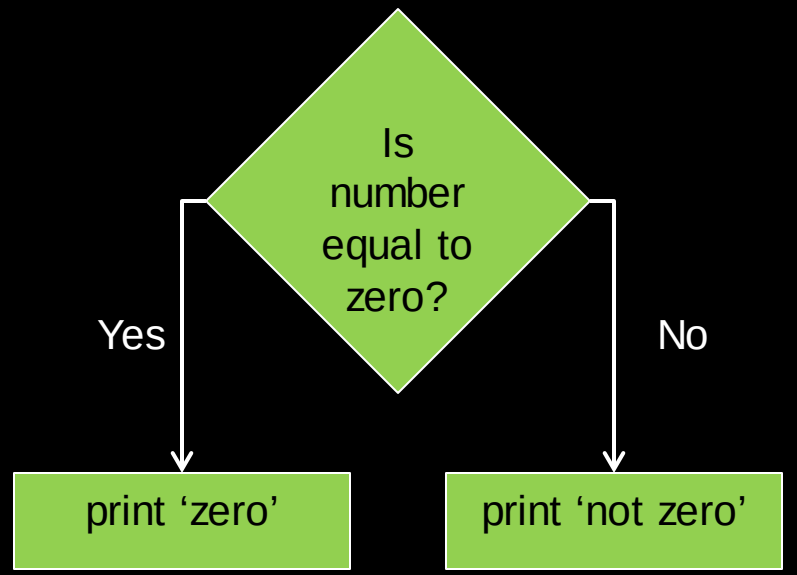
$$T_C = (T_F - 32) * \frac{5}{9}$$


when clicked
Program prompts us...
hide variable temp_fahrenheit
hide variable temp_celsius
ask Input value of degrees Fahrenheit: and wait
set temp_fahrenheit to answer
think Hmm... for 2 secs
set temp_celsius to temp_fahrenheit - 32 * 5 / 9
set temp_celsius to round temp_celsius
say join Conversion results in Celsius temperature temp_celsius
stop this script

Decisions

We often must write our programs to behave differently depending upon conditions

If the number is zero print ‘zero’, otherwise, print ‘not zero’



Conditionals

Common conditionals

<	less than
<=	less than or equal
>	greater than
>=	greater than or equal to
==	equal
!=	not equal

Examples

- If n is assigned a value of 10…

  n < 10 is False

  n <= 10 is True

  n > 5 is True

  n >= 5 is True

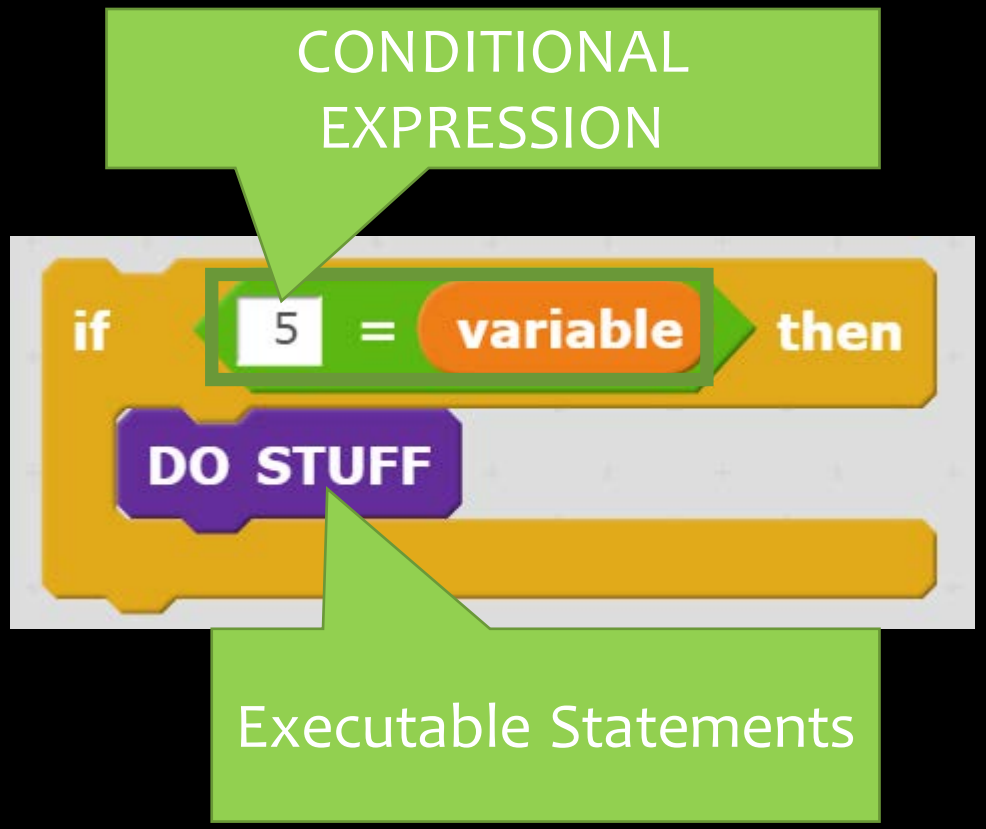
  n == 10 is True

  n != 10 is False

If Statement

If the conditional is true, the program will execute the statements in the "DO STUFF" section.

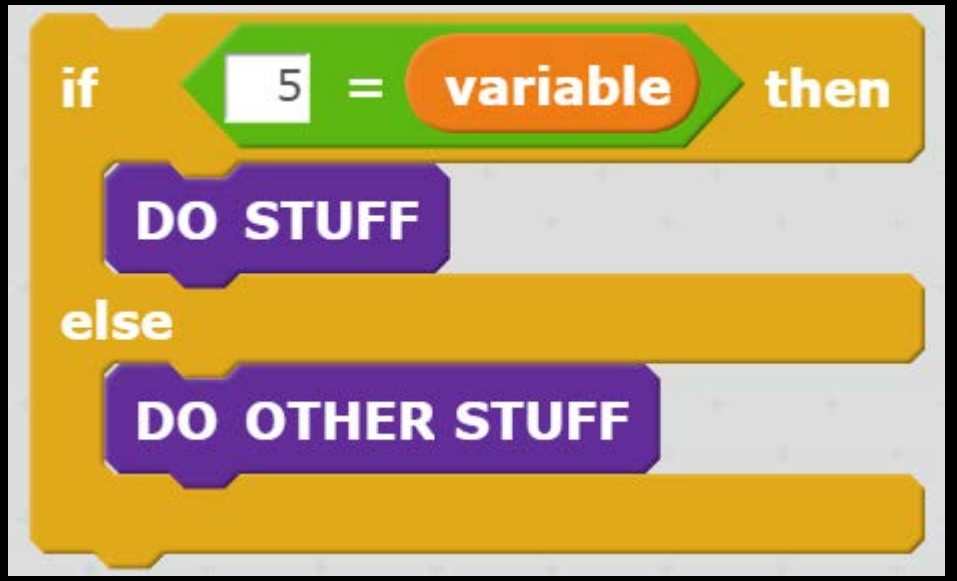
Otherwise, nothing will happen.



If/Else Statement

Much like the if statement, except now if the condition is false, the "DO OTHER STUFF" block is executed.

Under all circumstances, exactly one of the code blocks will execute, either "DO STUFF", or "DO OTHER STUFF"

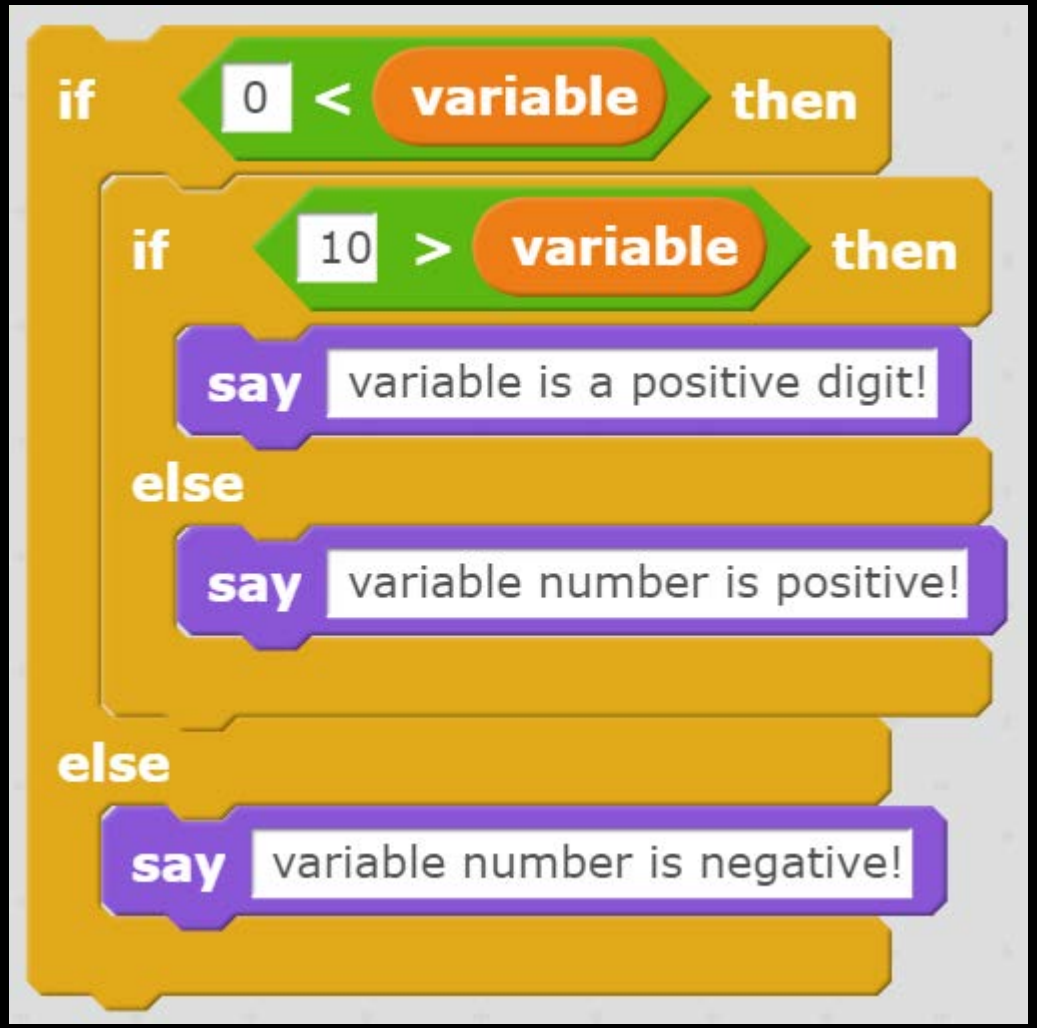


Nested If Statements

If statements and if/else statements can be nested, to give more complex behavior.

What does the program on the right say if variable is:

- -7
- 15
- 4

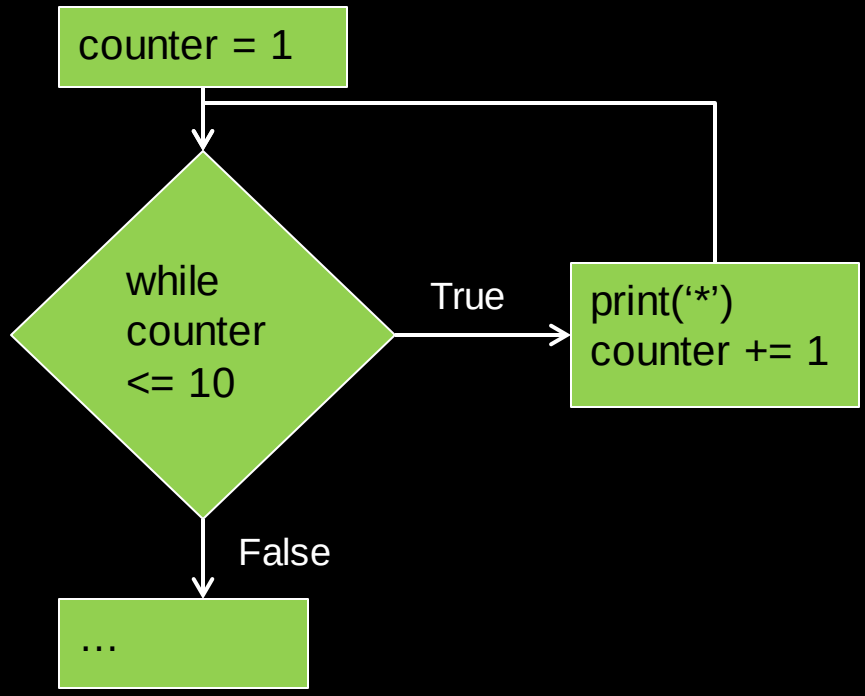


Looping

We often want to repeat an action

Example

print an asterisk 10 times



For Loop

- Looping in a range is so common, there is a “For” loop

- One use of a For loop is executing a certain number of times

Looping with Scratch

- What do you think the following code will do?

- Compare the block based to text based coding.

- ***Caution: Scratch displays "Hello" 10 times in same spot – Python displays "Hello" on 10 lines vertically***


```
for count in range(10):
  print('Hello!')
```

*Index for block based starts at 1. However, for text based, it starts at 0.

For Loop

The for loop below starts with counter taking on the values 1 up to BUT NOT INCLUDING 11 with a step size of 1

```
for counter in range(1, 11, 1):
    print('*')
```

Difference between for loops and repeat until loops

- Repeat until loops use a condition to determine the number of steps

  -Example, repeat until you are tired, do pushups

  -NOTE: The Scratch repeat until tests the condition before entering the block it controls
  Thus, it is really a while loop *BUT* condition is reversed

- NOTE: Python has no repeat until – it does, however, have a while

- For loops have a set number of steps

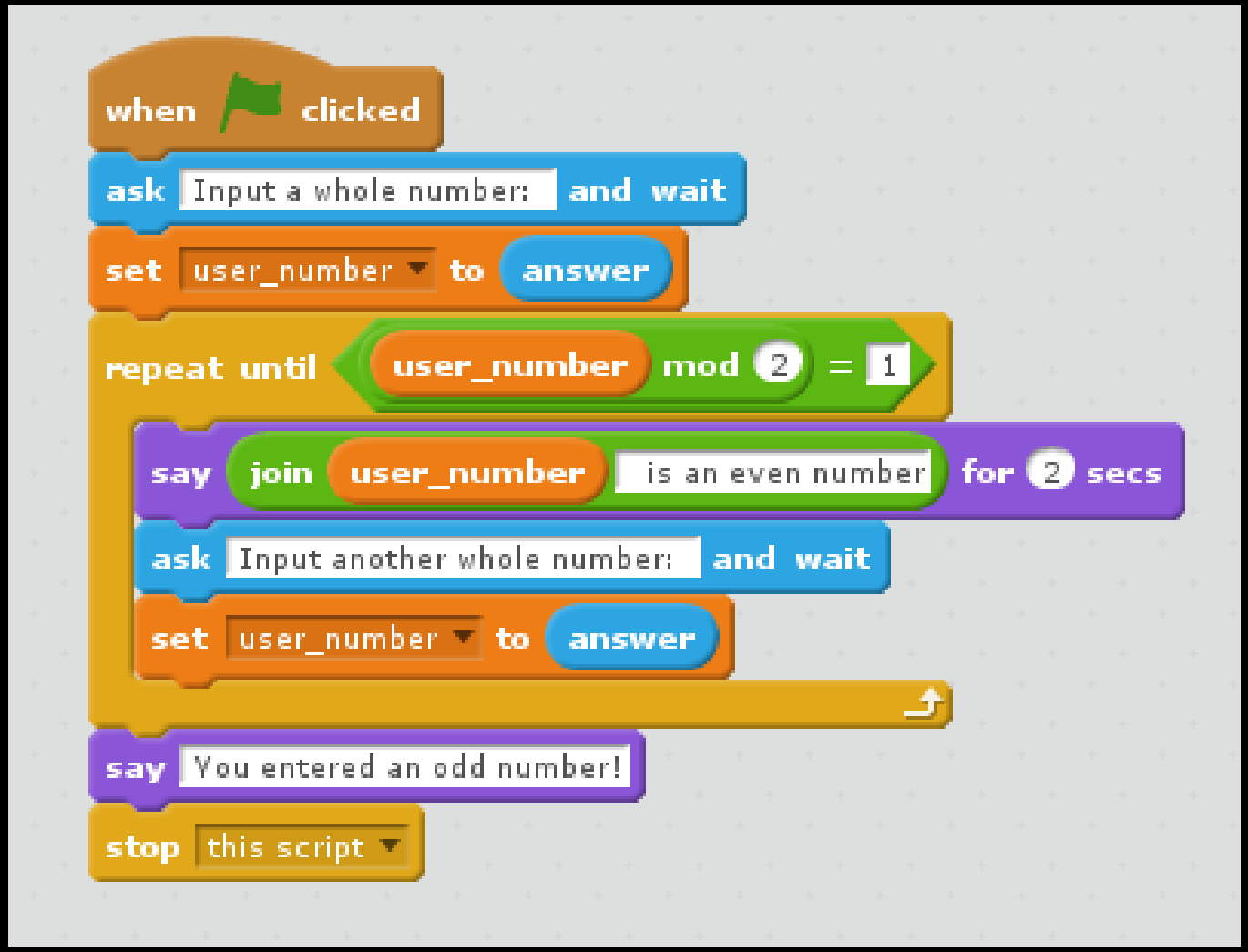
  -Example, you must do 10 pushups

Repeat Until/While Loop Application

- In a loop, continually request numbers as input and then print them as long as they are even
- Example output:

```
What is your number? 4
4
What is your number? 2
2
What is your number? 3
You entered an odd number.
```

While Loop Application – Solution (Blocks)



While Loop Application – Solution (Text)

```
user_number = int(input('Enter a whole number: '))

while user_number % 2 == 0:
    print( user_number,  'is an even number')
    user_number = int(input('Enter another whole number: '))
#end while
print('You entered an odd number!')
```

Common Problems

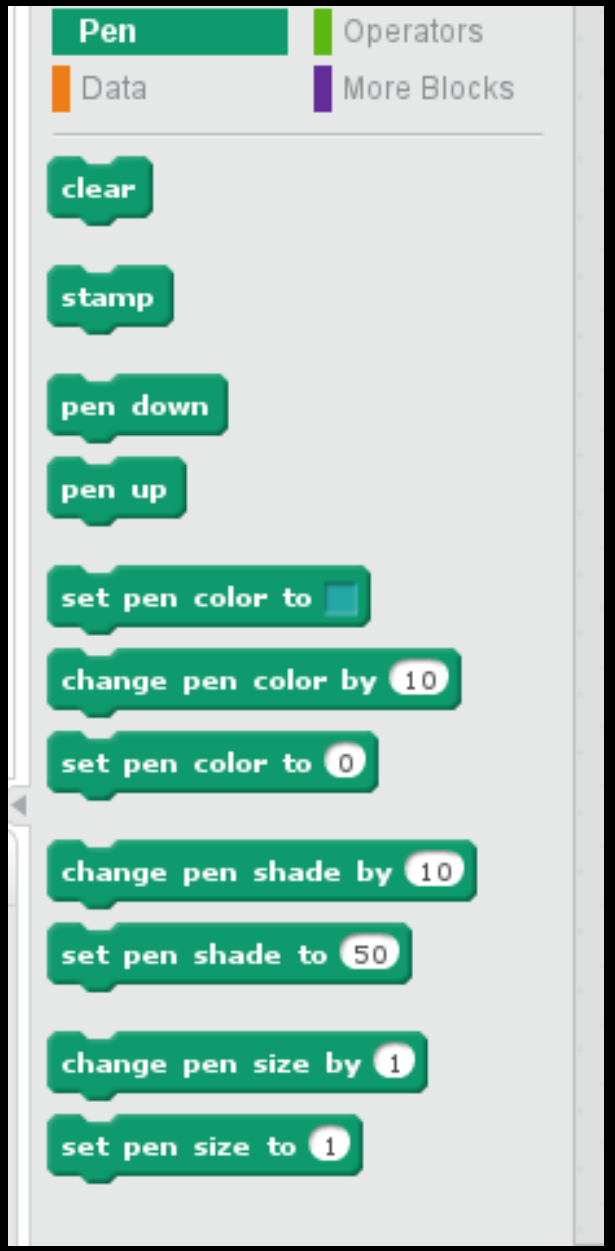
- An ”infinite loop” occurs when a while loop goes forever and never meets the condition to stop!

```
i = 1
while i < 10:
    print(i)
#end while
```

- If you accidentally initialize a variable to something that does not pass the condition, it will not enter the loop

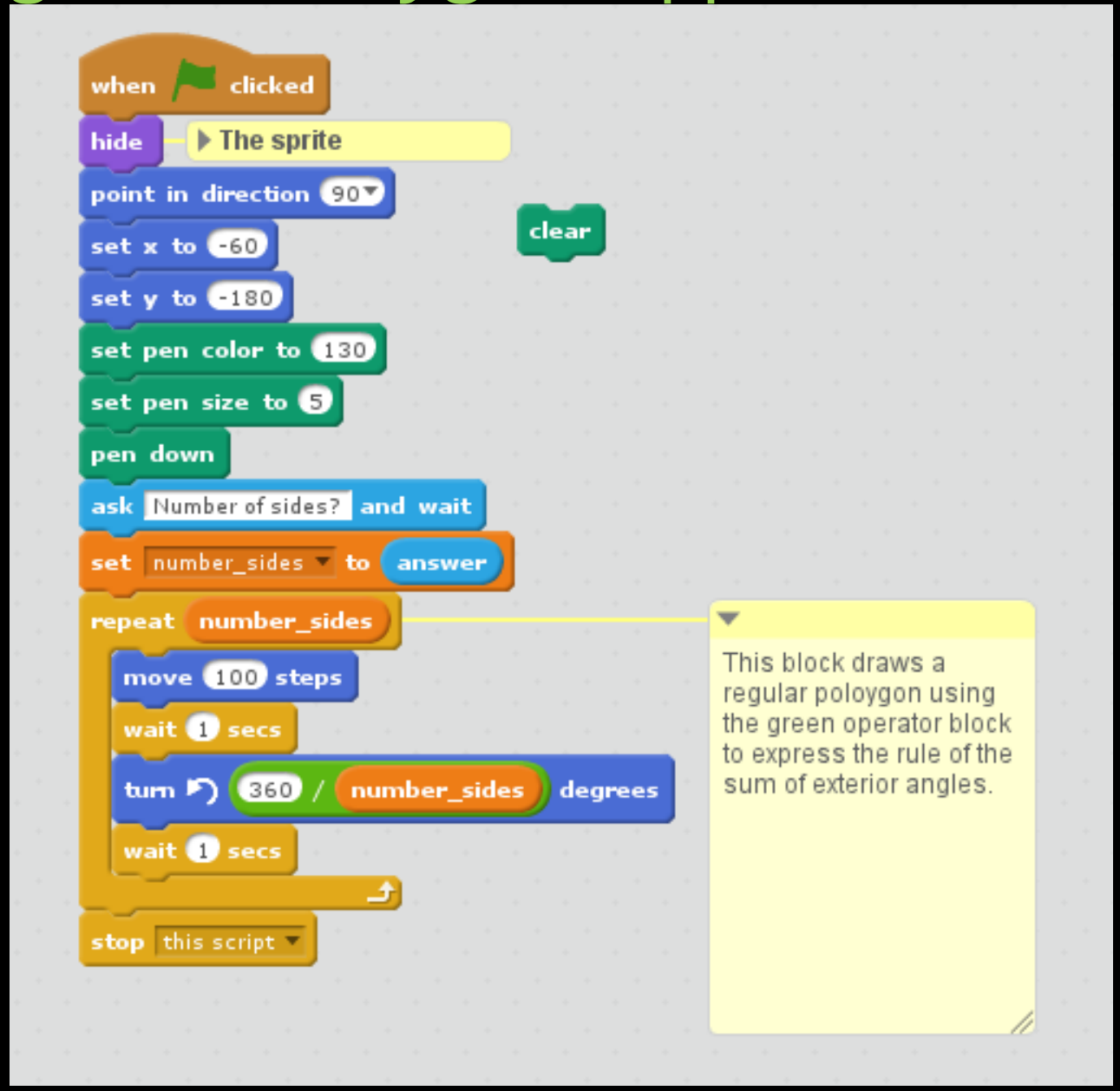
```
i = 11
while i < 10:
    print(i)
    i += 1
#end while
```

- We can use the blocks within pen to create shapes.



Scratch – Regular Polygon Application

- Write a program that takes in the number of sides (greater or equal to 3) in a regular polygon and creates the corresponding regular polygon. For example, if you input a 4, your pen should draw a square.



Lists

- Arrays in Python are referred as Lists, and are slightly different from other traditional programming languages such as C or Java
- A *List* is a sequence of values in Python, where in the sequence the values can be of various data types.
- For example:

[10, 20, 30, 40] ← *list of integers*
['Club', 'Spade', 'Diamond', 'Heart'] ← *list of strings*

['h. paul', 12345, 26, 'WTAMU'] ← *list of integers & strings*

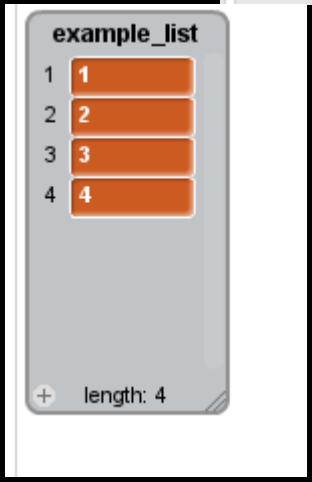
NOTE: Python lists can be of any type, so you can mix e.g. integers, strings, etc in lists.

Lists Block/Text Based

- To create a list in Python, simply enclose the elements (separated by comma) in square brakets

For example: [1, 2, 3, 4]

- To create a list in Scratch, Make a List and then go to stage and click the + on length to get number of elements desired and fill in values.



Accessing Elements of a List

- *Traversing* a list visits each value in a list in order.

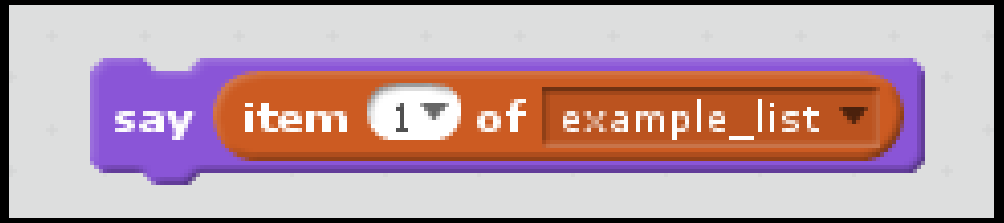
- We can also visit a specific element(s) within a list in order to use the value or modify the value

Accessing an element in a list

- We can visit/modifty any element in list by specifying index (text based starts at 0 and block based starts at 1):

For example:
```
example_list= ['cheddar', 'swiss', 'gouda‘, ‘limburger’]
print(example_list[0])
```

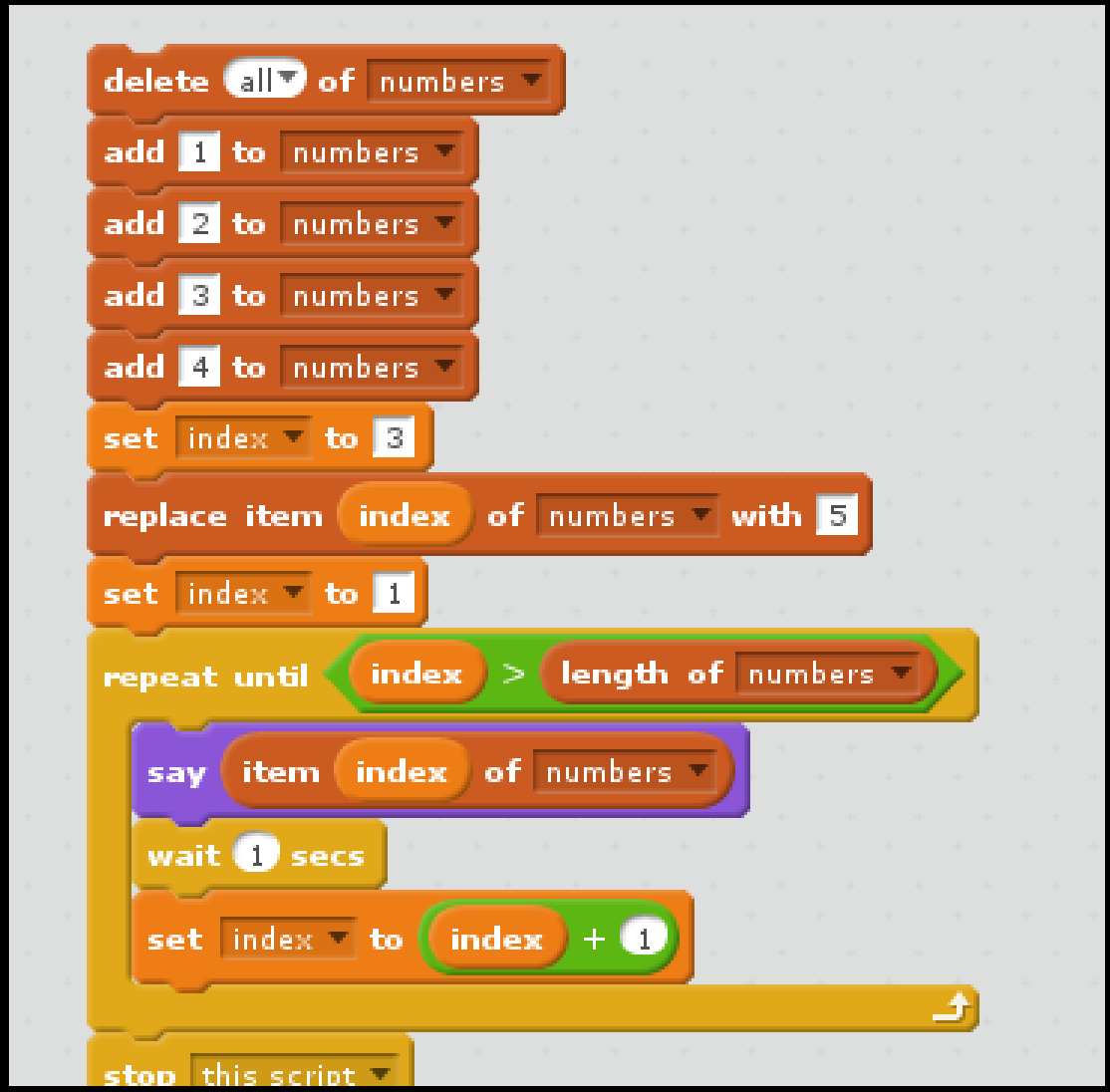
- In Scratch, we create the list example_list, expand its length to four elements manually, enter the values, and then can display the first value in list with:



Modifying an element of a list

- We can visit/modifty any element in list by specifying index:

For example: numbers = [1, 2, 3, 4]
numbers[2] = 5
print(numbers)



Finding an element within a list

Operator	Description
in	Evaluates to true if it finds a value in the specified sequence and false otherwise.
not in	Evaluates to true if it does not finds a value in the specified sequence and false otherwise.

```
cheeses = ['cheddar', 'swiss', 'gouda']
print('swiss' in cheeses)
```

The output of these would be true since 'swiss' is in the cheeses list.

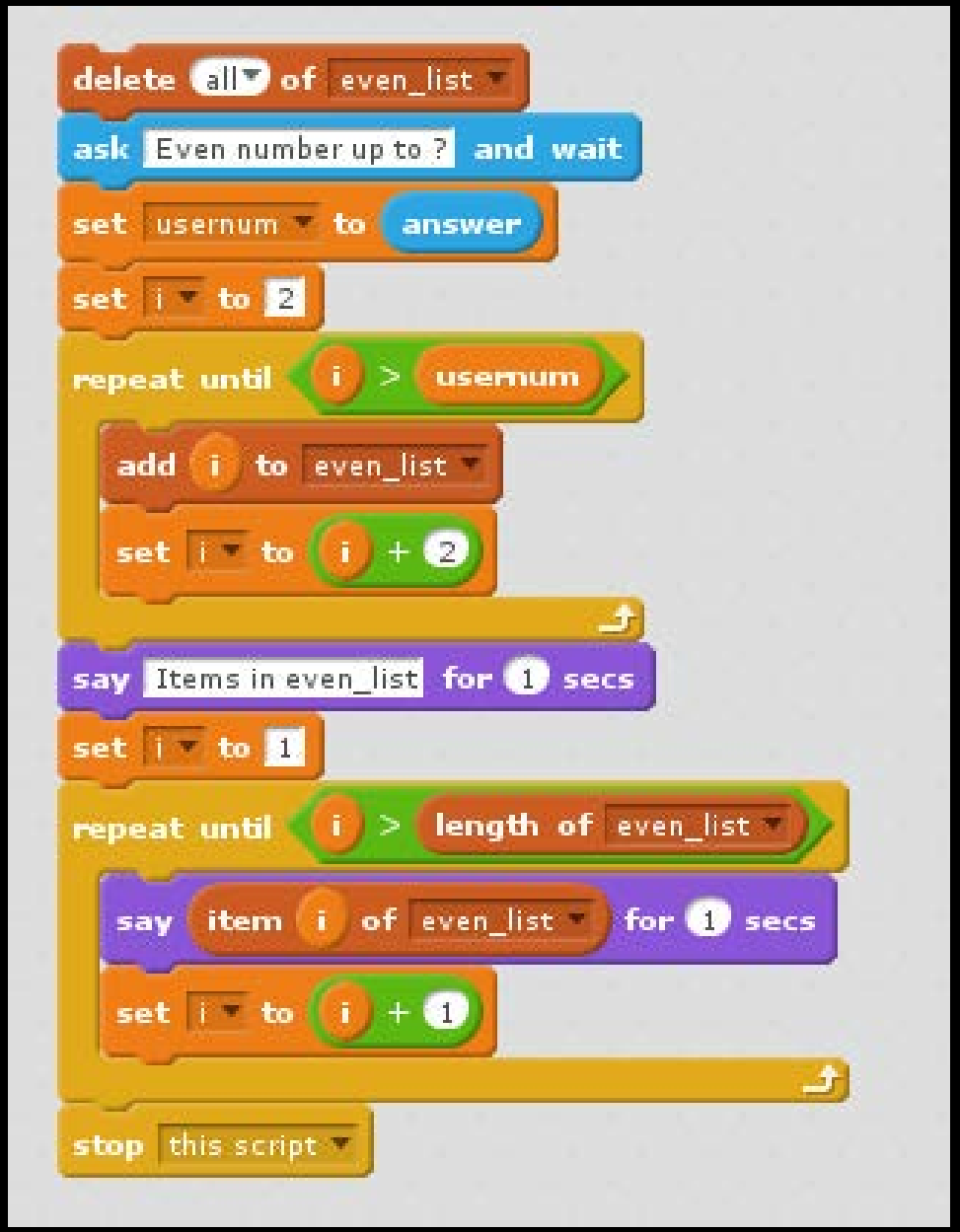
When working with List

- The index of First Element in a list is 0 for text based and 1 for block based.
- Use function len(list name) returns the number of elements in the list.
- Use function range(n) returns a list of Indices from 0 to n-1, where n is length of the list.
- Use listname.reverse() to reverse the elements within the list (text based only)
- Use listname.sort() to sort the elements in ascending order (text based only)

Exploring Lists

Take a number and store in a list only even numbers up to the number provided by the user. NOTE: Code for Scratch uses variable even_list.

```
#Python code
even_list = [ ]
usernum = int(input('Even number up to ?'))
for i in range(2, usernum+1, 2):
    even_list.append(i)
#end for
print('Items in even_list')
print(even_list)
```



Procedural Abstraction

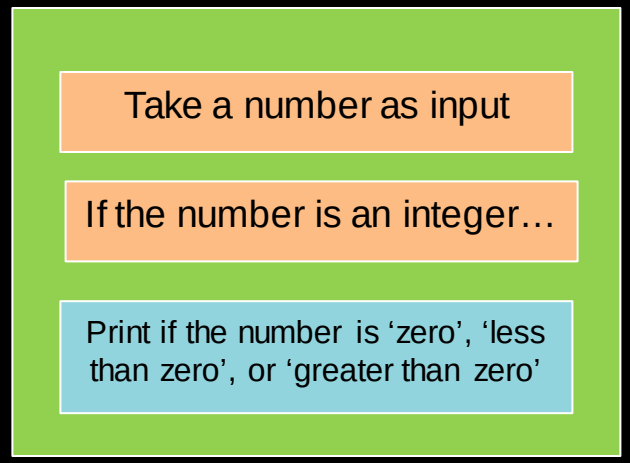
One programming skill is “decomposition”.

- Break a larger problem down into smaller problems

Example:

- Write a program that takes a number as input and if the number is an integer print if the number is ‘zero’, ‘less than zero’, or ‘greater than zero’

Sample problem

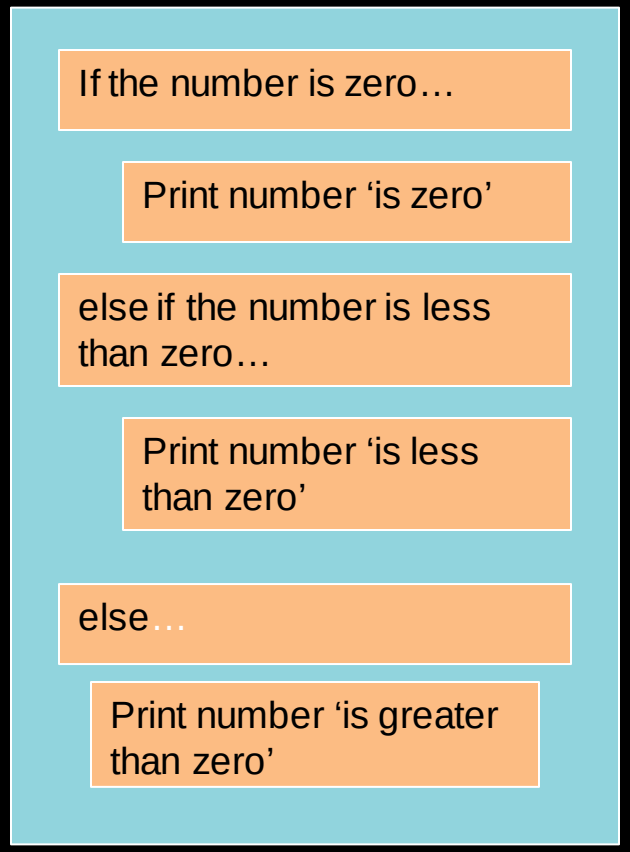


We can decompose the example and solve each part

Example:

Print if the number is ‘zero’, ‘less than zero’, or ‘greater than zero’

Print if the number is ‘zero’…



- Can take one part of the problem and further decompose that part
- This decomposition now resembles computer pseudocode
- Can write this part of the program as a “procedure”

Why procedures?

- Can write parts of the program
  - Useful when the whole program is too big to grasp at one time
  - Allows us to test pieces of the program before building on those pieces
  - Allows us to reduce redundancy by creating functions that get called multiple times

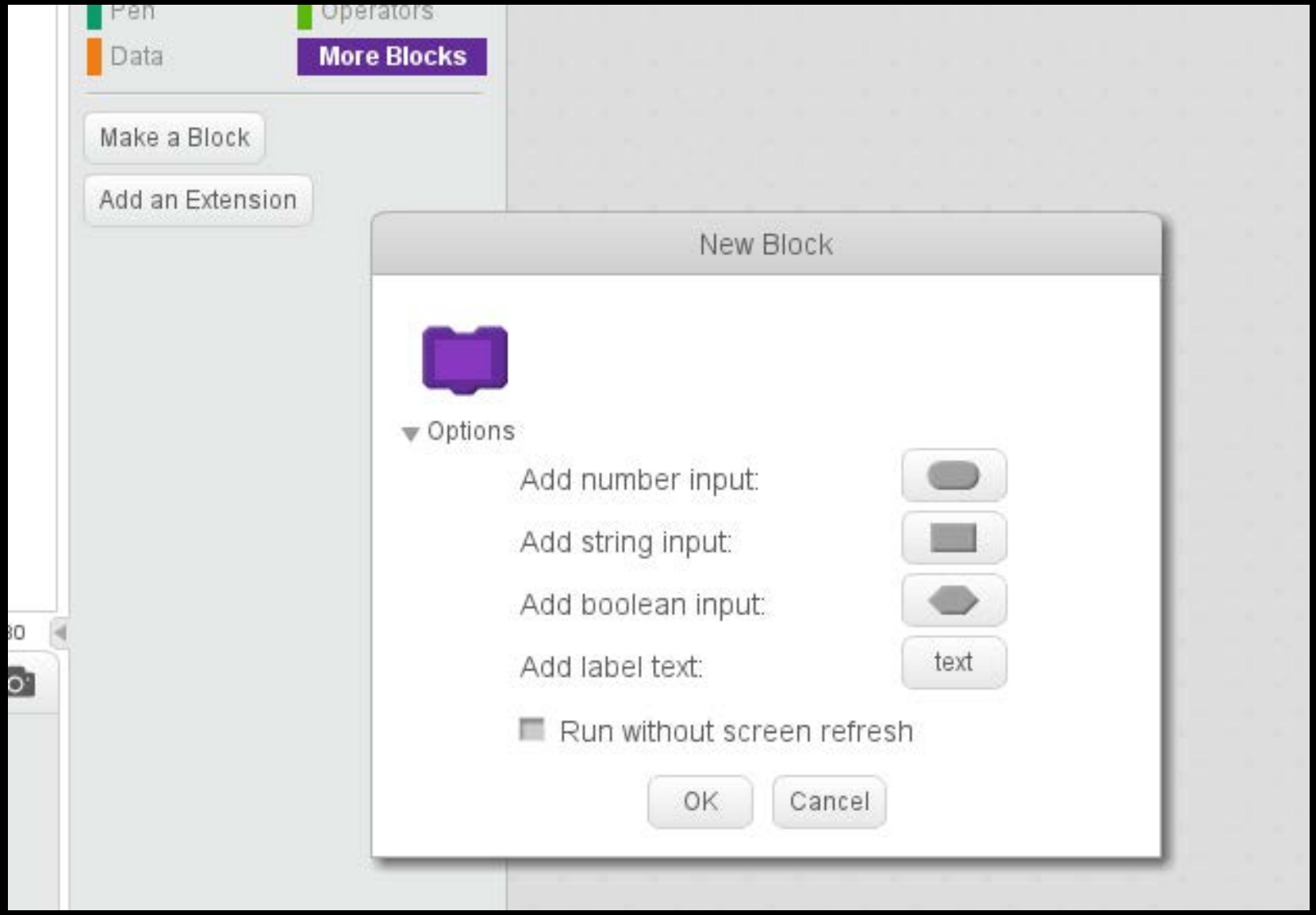
Procedures/Functions in Python

- Create a function in Python using the def keyword
- def is followed by the function name
- Parameters, if any, follow the function name
- The body is indented following the definition

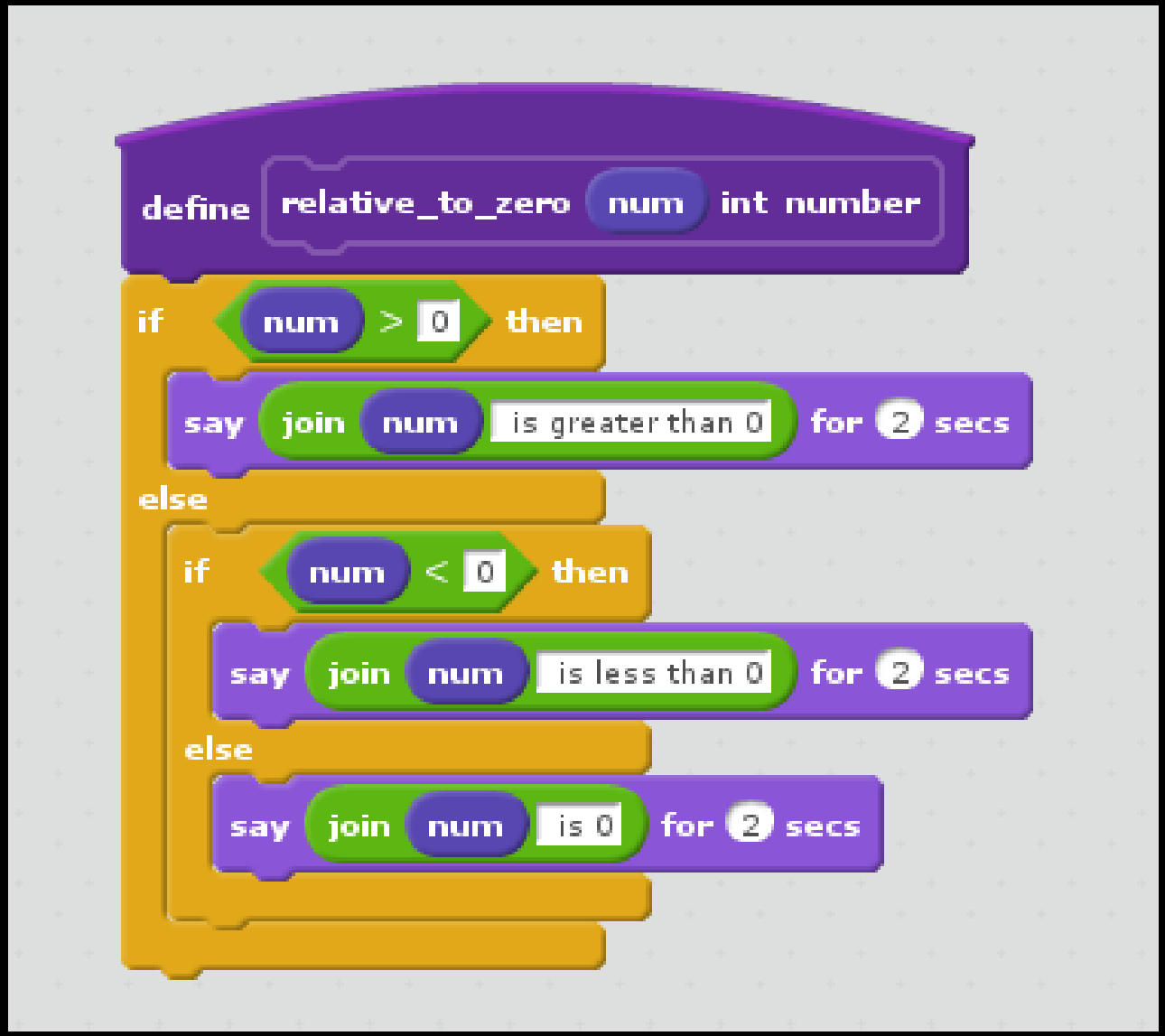
Defining a Procedure(Text)

```
def relative_to_zero(num):
    if num > 0:
        print(num, “is greater than zero”)
    elif num < 0:
        print(num, “is less than zero”)
    else:
        print(num, “is zero”)
     #end if
#end relative_to_zero
```

Defining the procedure in Scratch



The procedure in Scratch



The Python program testing relative_to_zero procedure

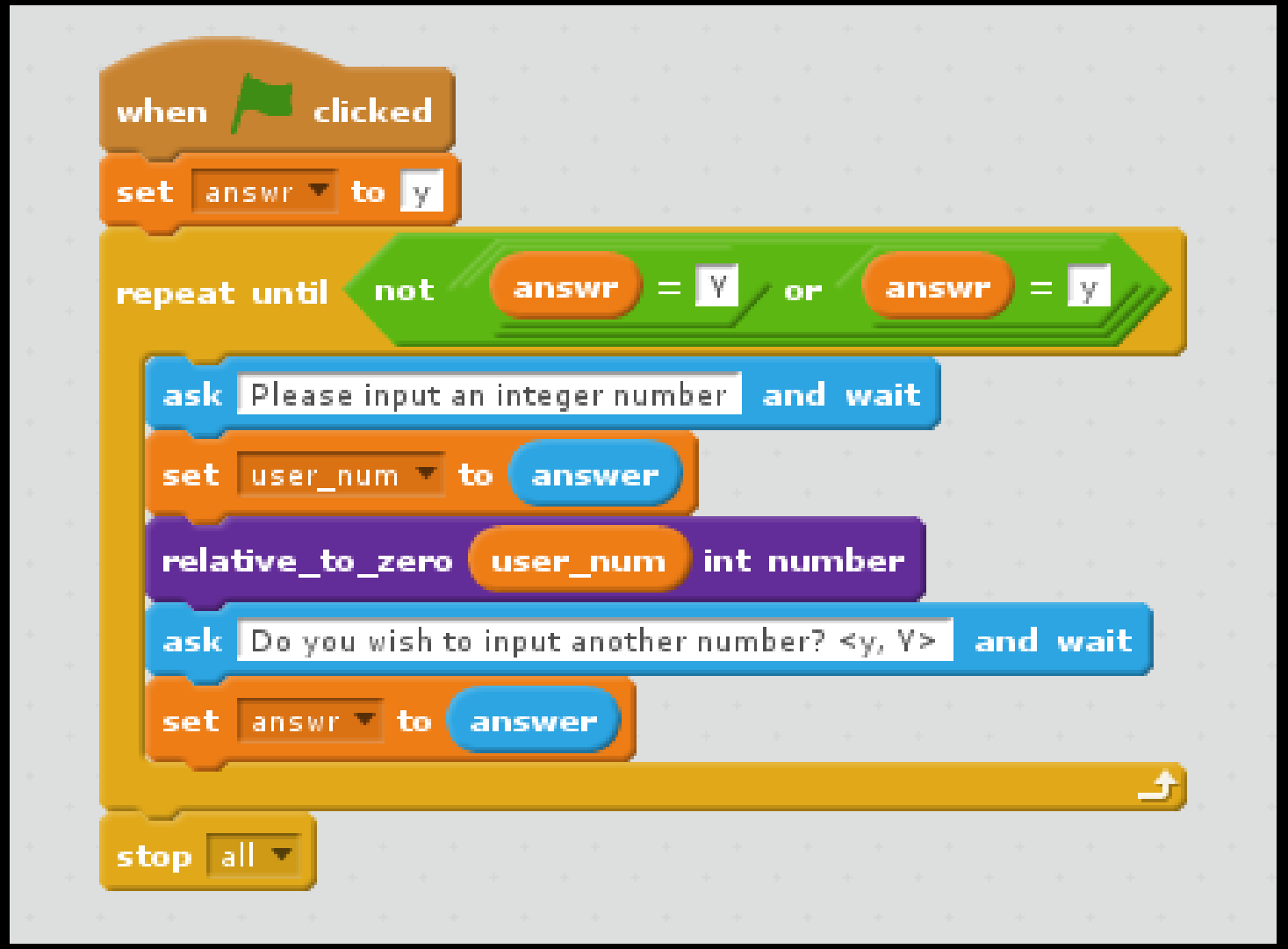
```
#!/usr/bin/env python3

def relative_to_zero(num):
   if num > 0:
       print(num, "is greater than zero")
   elif num < 0:
       print(num, "is less than zero")
   else:
       print(num, "is zero")
   #end if
#end relative_to_zero

# main program starts here
answer = 'Y'

while answer in ['y', 'Y']:
    user_num = int(input("Please input an integer number "))
    relative_to_zero(user_num)
    answer = input("Do you wish to input another number? <y,Y> ")
#end while
```

The Scratch program testing relative_to_zero procedure



NOTE: No functions in Scratch

- There is no function defining mechanism in Scratch 2.0 – the current version of Scratch

- There appears to be no function defining mechanism in Scratch 3.0 – although there are many voices requesting such.

That's all . . . . .

Questions?

Comments?

Observations?

- Presented by:

- H. Paul Haiduk
  Computer Science Coordinator
  West Texas A&M University
  hhaiduk@wtamu.edu

  David Kossey
  Computer Science/Robotics/Math
  West Texas High School
  david.kossey@pspcisd.net

Presentation available on github @ https://github.com/HHaiduk/thunder