#3: Variables, Input, and Output

SAMS SENIOR NON-CS TRACK

Last Time

Use anchors to draw text on the screen

Draw images using lines and polygons

Review different approaches towards finding where to put shapes on the screen

Ex2-2 Feedback

Most of you did quite well- great job!

Half the class made mistakes on #1, though.
Let's quickly go over how to solve it.

Today's Learning Goals

Use **variables** to hold and update data

Use **input and output** to write code that supports user interaction

Variables

Statements and Expressions

Before we learn about variables, it may help to clarify the difference between **statements** and **expressions** in code.

A **statement** is a line of code that executes an action. print("foo") and canvas.create_rectangle(0, 0, 400, 400) are examples of statements- when we run that line of code, something changes in the program.

An **expression** is a piece of code that evaluates to a value. 4 - 5 and "Hello" + " World" are both expressions- both can be computed to find a result.

In general, we can combine expressions together to make new expressions, **but we can only have one statement per line**. Additionally, if we run an expression by itself (not as part of a statement), **nothing happens.**

Variables as Data Storage

Sometimes, we want to store expressions so that we can use them multiple times throughout the code without having to re-compute them. We can do this by using a **variable**.

In math, variables are used as placeholders for numbers. Variables in programming are similar- they can **store** a value, then be used as a **placeholder** for that value. Whenever the program reaches a variable in the code, it will **replace** that variable with its current value.

This means that, once we have given a variable a value, we can use that variable anywhere we would have used the value!

Assigning and Using Variables

To create a new variable, we'll use the syntax:

```
<varName> = <expression> # This is a statement- it does an action!
```

The variable name can be anything, as long as it starts with an alphabetic letter. It's usually helpful to give variables names that describe what data they hold.

To use a variable, we can substitute in the variable name where we would have put the expression before. For example:

```
print(<varName> / 2)
```

Variable Use Example

Let's say we want to write a program that greets a person based on their name. We could write a print statement with the name in it directly, or we could use a variable to keep track of the name separately!

```
name = "Denise"
print("Hello " + name)
```

This makes it easier for us to replace Denise with a different name if we want to change the program.

More Complex Assignments

We can store more than individual values in variables- in fact, we can store whole expressions! The variable will just hold onto the result of the expression.

```
x = (5 - 2) ** 2
print(x)
```

Exercise 1: Time Tracker

Go to the schedule page and download the starter file for today's lecture. You'll write exercise code under the comment with the exercise's number.

Exercise 1: write a program that stores the current hour (as an integer) in a variable called **time**. Then use that variable to print out a string about what time it is.

Updating Variables

There is one major difference between variables in math and variables in programming. In math, the variable has one specific value; in programming, a variable can **change its value over time**.

```
x = 4
x = 5
print(x)
```

Think of a variable as a box that can hold objects. When we set a variable equal to a new value, we're removing the original object and putting something new in. This means we can no longer access the first value- if we use the variable, it will just give us the latest thing we put in the box.

Updating Variables

Often, when we update a variable, we want to **modify the current value** that the variable holds.

For example, if we wanted to double the value in a variable x, we would write the code:

```
x = x * 2
```

Note that the expression (the part on the right) evaluates first, then the result is put into the variable (the part on the left).

You do!

Prediction Exercise: what value will x hold after each step of the following program?

```
x = 5
y = x * 5
x = y + 3
```

Exercise 2: Time Change

Exercise 2: Write a line of code that updates the **time** variable you created in the previous exercise so it holds the following hour. Then print out the result.

Note: don't worry about wrapping 12 o'clock to 1 o'clock! Just keep it simple.

Problem Solving with Variables

In programming, we often use variables to hold **important intermediate information**, which makes the code easier to write and read.

We also use variables to hold values that will be used **multiple times**, so that we don't accidentally use the wrong value in one place and mess up the expression.

Variables in Graphics

Variables are especially useful in graphics, since we can use them to hold key information about the locations of shapes.

We can also use two variables that are assigned when we set up the canvas- the **width** and **height** of the window.

For example, if we wanted to write code that drew a circle in the middle of the window, we could write:

```
cx = width / 2
cy = height / 2
r = 100
canvas.create_oval(cx - r, cy - r, cx + r, cy + r)
```

Exercise 3: Diamond

Exercise 3: write tkinter code to draw a diamond with the text SHINE in the middle, as shown to the right. For full credit, you must use **at least two** variables that hold relevant data about the shapes you draw.

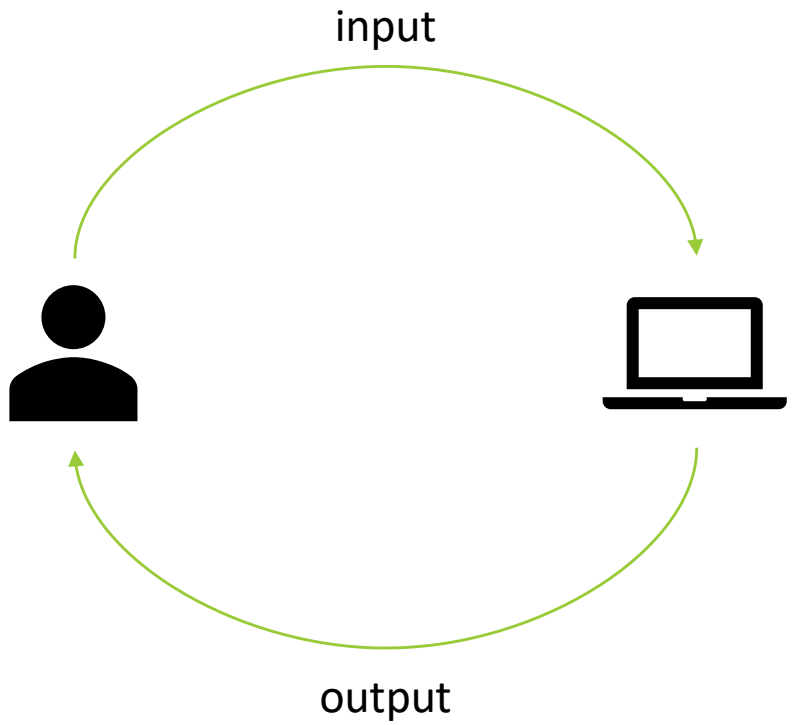
Hint: which coordinates are used most often in the shapes? These are most likely to be useful variables.

Input and Output

Computer Communication

When you use a program (like your internet browser, or Pyzo), you're communicating with the computer. You give the computer **input** on what you want to do, and it produces **output** based on your requests.

When we write programs, we can capture input and produce output directly. This lets us make programs that a user could interact with on their own!



Program Output

We've already learned about two different forms of program output!

Print commands let us print text to the interpreter. We've already seen how this can be used to show the user messages.

Graphics can be used to show the user images on the screen. This is also a form of output.

Program Input

Input is a bit trickier. We can technically provide input in the program itself, but you don't want the user to need to change the program in order to interact with it.

Another way to get user input is to **ask the user for input while the program runs**. We can do this using the built-in command **input()**. Input takes a string (the prompt that the user will be shown), then pauses the program while the user types a response. When the user presses Enter/Return, input evaluates to the text that the user entered.

```
response = input("How are you? ")
print(response)
```

Non-text Input

Occasionally, we want to ask the user for a number instead of text. However, the program will still parse the result of input() as text.

How can we get the number? Use the built-in command **int()**, which takes a string and returns the integer version of the number in the string. Warning: this will crash the code if the string does not contain a number!

```
age = input("What's your age? ")
ageNum = int(age) # will crash if the string does not hold a number!
```

Fun fact: you can also turn ints into strings with the built-in command **str()**.

```
print("You'll be 20 in " + str(20 - ageNum) + " years")
```

Exercise 4: Greeting Program

Exercise 4: write a program that asks the user for their name and where they're from, then prints out a personalized greeting using those two pieces of information.

For example, if I entered "Kelly" and "Maryland", my program might respond:

"Hello Kelly! I hear Maryland is nice this time of year."

If you finish early, try adding more interaction to your program! Just make sure to leave those first two parts in.

Today's Learning Goals

Use **variables** to hold and update data

Use **input and output** to write code that supports user interaction