TYPES & VARIABLES! (LECTURE)

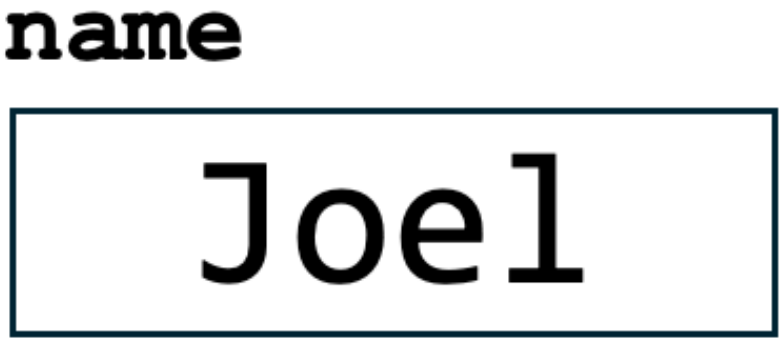
PrairieLearn & Gradescope Demo!

REVIEW: VARIABLES

```
name = "Joel"
```

It creates a **variable** named `name` holding the value `"Joel"`.

variable ↔ box with a name attached to it.



PRINTING VARIABLES

We can print variables, similar to how we print strings or numbers:

```
course_number = 1100
print(course_number) #  1100
```

🖨️👇

```
1100
```

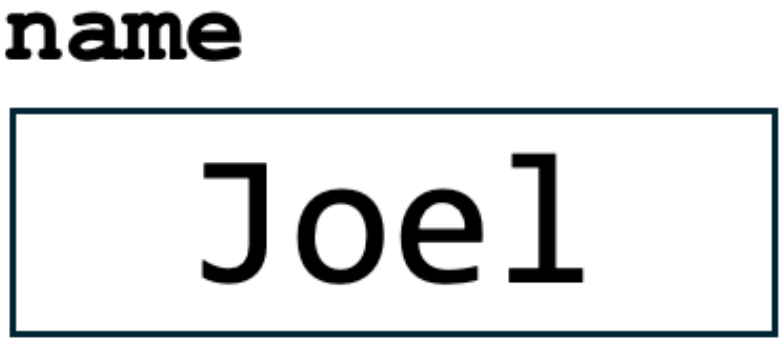
`print()` displays the *value* stored inside of the variable, not its *name!*

REVIEW: VARIABLES

```
name = "Joel"
```

The value within the box can change over the lifetime of the program (i.e. while it is being run or executed).

- variable only stores one value at a time
- previous values are forgotten forever

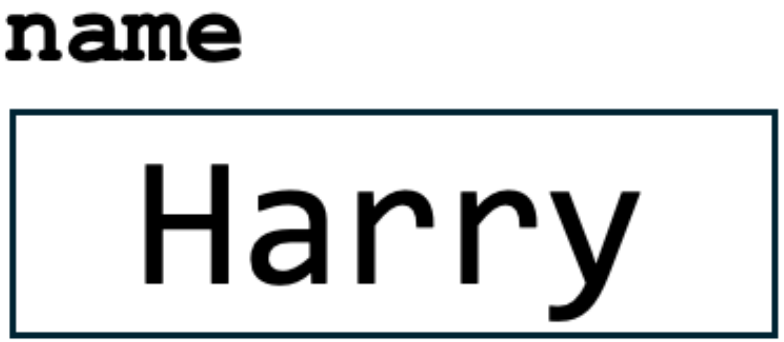


VARIABLES CHANGE

Variables can change over the course of a program, and can only hold one value.

Consider:

```
name = "Harry" # line a  <-----
name = "Joel"  # line b
print(name)
```

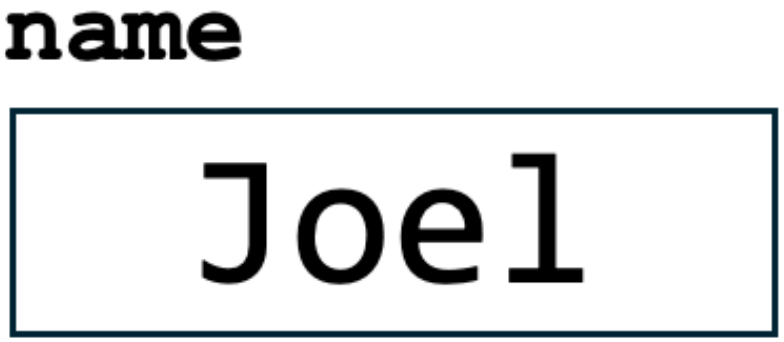


VARIABLES CHANGE

Variables can change over the lifetime of a program, and can only hold one value.

Consider:

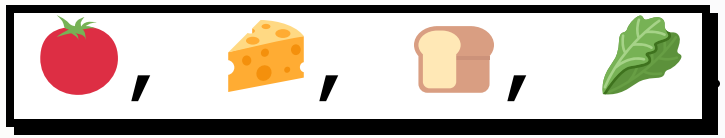
```
name = "Harry" # line a
name = "Joel"  # line b    <---
print(name)
```



Let's do some practice activities to build our understanding of important variable behaviors!

We're going to be working with programs in a sort of loose Python-y thing to describe real actions, but the rules are all the same.

Variables have names and store a single value at a time.

Here's our domain:  Let's assign.

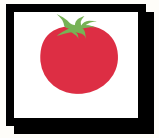
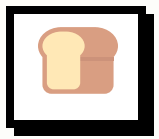
```
stackable_one = _________
stackable_two = _________
stackable_three = ___________
```

```
place_bread()
place(stackable_one)
place(stackable_two)
place(stackable_three)
place_bread()
```

Variables can take on different values over the course of a program.

Our goal: **make the same sandwich as before.**

Our goal: **make the same sandwich as before.** Here's our domain:

- 
-
- 
-

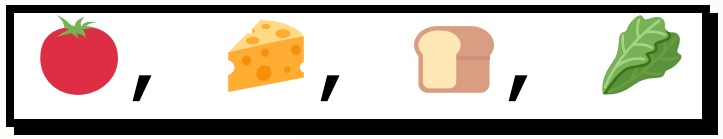
Let's assign values to our variable.

```
stackable = ____
place_bread()
place(stackable)
stackable = ____
place(stackable)
stackable = ____
place(stackable)
place_bread()
```

Variables can be updated in terms of themselves to accumulate changes over time.

Our goal: **make a super filled sandwich.** To start:

```
payload = 👐
```

Ingredient domain: 

```
place_bread()
payload = payload + ____
payload = payload + ____
payload = payload + ____
place(payload)
place_bread()
```

MAKING A SANDWICH MORE EXCITING

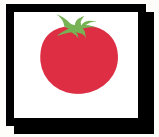
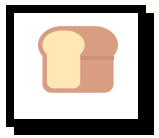
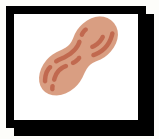
So far, our ingredients (🍅, 🧀, 🍞, 🥬) have all shared the property that they are all *stackable.* We might say that these ingredients all belong to the `stackable` **data type**.

- Reminder: a **data type** is a group of values that share a set of operations in common.
- **L1:** can you think of a few "values" (ingredients) that might be part of a `spreadable` data type?
- **C1:** can you think of a different data type of ingredients for sandwiches? Identify some examples that might fit into it and briefly justify why this corresponds to a different type.

The values we place in our variables have different types, so it's important to think carefully about when we use each type!

Our goal: **make an appropriate sandwich.**

Our domain:

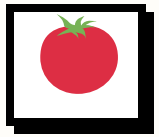
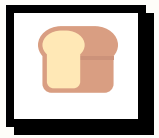
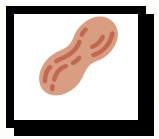
- 
-
- 
-
- 
-
-

```
spreadable = ____
stackable_1 = ____
stackable_2 = ____
place_bread()
spread(spreadable)
place(stackable_1)
place(stackable_2)
place_bread()
```

When you try to apply an operation to a value from the wrong data type, programs crash and bad things happen.

Our goal: **spread the unspreadable and stack the unstackable.** (Make a sandwich where we choose a value from the wrong type.)

Our domain:

- 
-
- 
-
- 
-
-

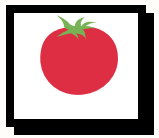
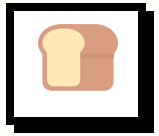
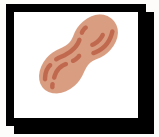
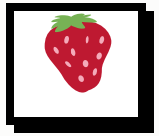
```
spreadable = ____
stackable_1 = ____
stackable_2 = ____
place_bread()
spread(spreadable)
place(stackable_1)
place(stackable_2)
place_bread()
```

Variable names provide important clues as to the intended purpose of that variable. Short or confusing names lead to bugs!

Our goal: **observe what happens when variable names are chosen poorly!**

FIRST ASSIGN OUR VARIABLES

Our domain:

- 
-
- 
-
- 
- 
-

```
bread = ____
b = ____
c = ____
d = ____
e = ____
f = ____
```

MAKE THE SANDWICH!

```
place(bread)
spread(b)
place(c)
place(d + e)
spread(f)
place(bread)
```