Module 2

Variables

Motivation

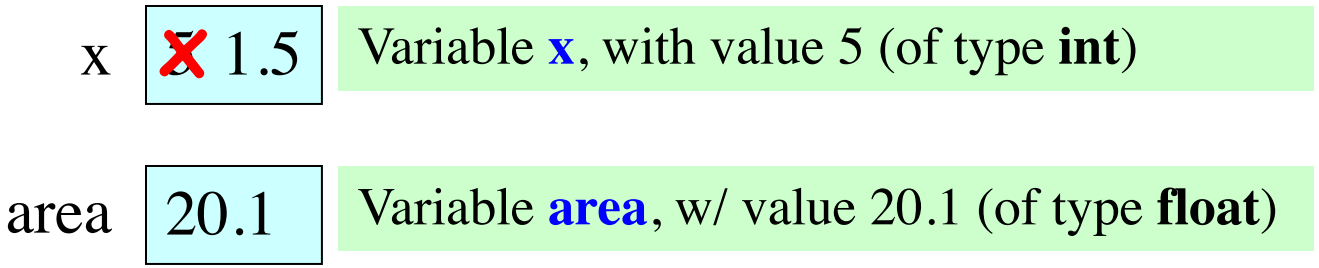
- This series introduces the concept of variables
  - Very powerful programming concept
  - Necessary for more complex Python features
- But variables can be tricky to work with
  - With expressions, we got a value right away
  - A lot of variable features happen invisibly
- This can lead to lot of frustration
  - You think Python is doing one thing
  - It is actually doing something else

Visualization

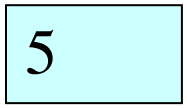
- You need to learn to think like Python thinks
  - Otherwise you and Python will miscommunicate
  - Like a coworker with language/cultural issues
  - A good programmer sees from Python's persp.
- Do this by building **visual models** of Python
  - You imagine what Python is doing invisibly
  - Not exactly accurate; more like **metaphores**
  - We call this skill **visualization**
  - It is a major theme of this course

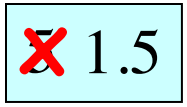
Variables

- A **variable**
  - is a **box** (memory location)
  - with a **name**
  - and a **value** in the box
- Examples:



Variables in Python

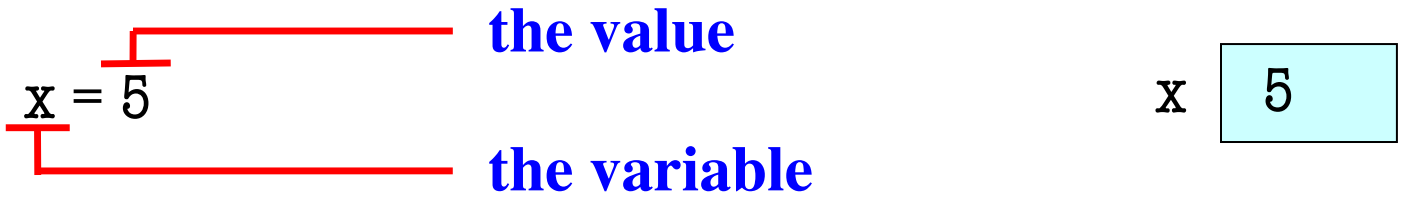
- These boxes represent a "memory location"
- Allows variables to be used in expressions
  - Evaluate to the value that is in the box
  - **Example:** `x` 

`1 + x` evaluates to `6`
- Allows variables to change values
  - **Example:** `x` 

  - They can even change the type of their value
  - This is different than other languages (e.g. Java)

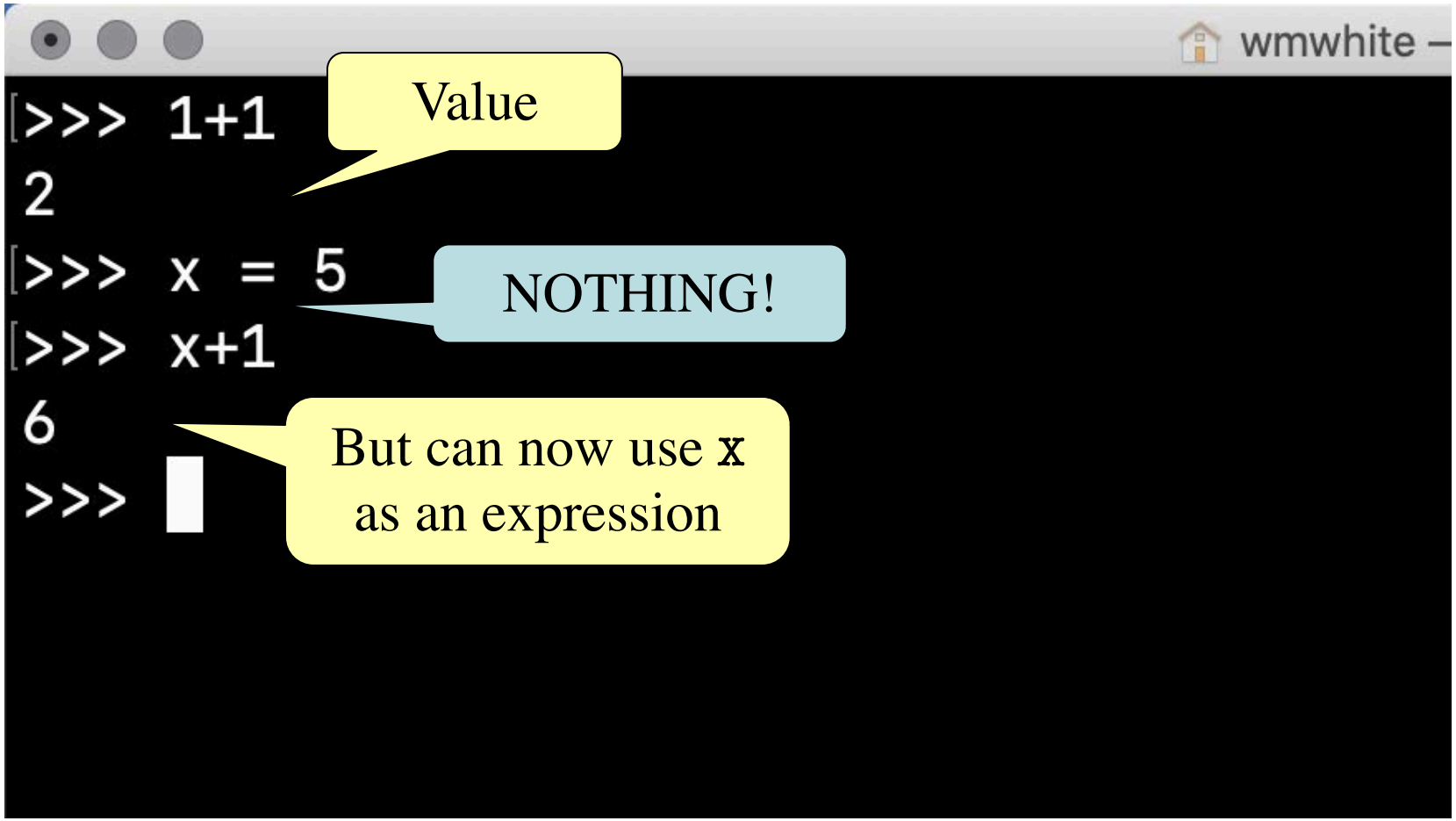
Creating Variables

- So how do we make a variable in Python?
  - Cannot do just with expressions
  - Expressions give us a value
  - We want to *command* Python to make a box
- Variables are created by **assignment statements**



- This is a **statement**, not an **expression**
  - **Expression**: Something Python turns into a value
  - **Statement**: Command for Python to do something

Expressions vs Statments



Naming Variables

- Python limits what names you can use
  - Names must only contain letters, numbers, _
  - They cannot start with a number
  - Also cannot be a **reserved word** (will see later)
- **Examples**
  - `el` is a valid name
  - `le2` is not valid (it is a float)
  - `a_b` is a valid name
  - `a+b` is not valid (it is an + on two variables)

Variables Do Not Exist Until Made

- Example:

```
>>> y
Error!
>>> y = 3
>>> y
3
```

- Changes our model of Python
  - Before we just typed in one line at a time
  - Now program is a **sequence** of lines

Variables Do Not Exist Until Made

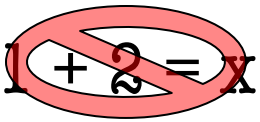
- Example:

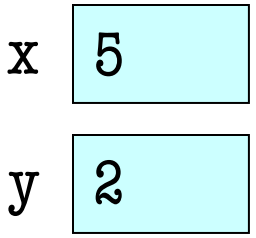
```
>>> y
Error!
>>> y = 3
>>> y
3
```

```
>>> x = 3
>>> y = 4
>>> x+y
7
>>> x = 2.5
>>> x+y
6.5
```

- Changes our model of Python
  - Before we just typed in one line at a time
  - Now program is a **sequence** of lines

Assignments May Contain Expressions

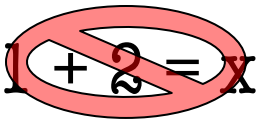
- **Example**: `x = 1 + 2`
  - Left of equals must always be variable: `1 + 2 = x` 
  - Read assignment statements right-to-left!
  - Evaluate the expression on the right
  - Store the result in the variable on the left
- We can include variables in this expression
  - **Example**: `x = y+2`
  - **Example**: `x = x+2`

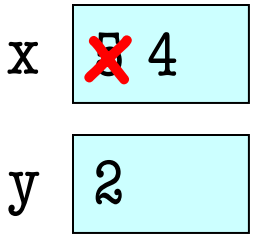


This is not circular!
Read right-to-left.

Assignments May Contain Expressions

- **Example**: `x = 1 + 2`
  - Left of equals must always be variable: `1 + 2 = x`
  - Read assignment statements right-to-left!
  - Evaluate the expression on the right
  - Store the result in the variable on the left
- We can include variables in this expression
  - **Example**: `x = y+2`
  - **Example**: `x = x+2`

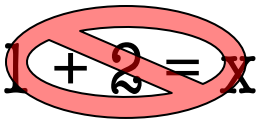


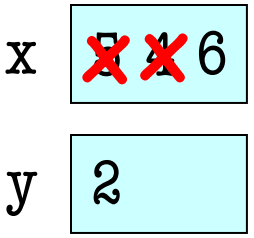


This is not circular!
Read right-to-left.

Assignments May Contain Expressions

- **Example**: `x = 1 + 2`
  - Left of equals must always be variable: `1 + 2 = x`
  - Read assignment statements right-to-left!
  - Evaluate the expression on the right
  - Store the result in the variable on the left
- We can include variables in this expression
  - **Example**: `x = y+2`
  - **Example**: `x = x+2`





This is not circular!
Read right-to-left.

About Crossing Off

- The crossing off is a helpful mental model
  - Emphasizes that the old value was deleted
  - But it shouldn't stay there over time
  - Else might think a box remembers old values
- So what do we do?
  - Cross off at the time we execute the statement
  - But gone when we revisit the variable later
- Again, part of our **visualization**

Python is Dynamically Typed

- What does it mean to be **dynamically typed**?
  - Variables can hold values of any type
  - Variables can hold different types at different times
- The following is acceptable in Python:

```
>>> x = 1
>>> x = x / 2.0
```

  ← x contains an **int** value

  ← x now contains a **float** value

- Alternative is a **statically typed language**
  - Each variable restricted to values of just one type
  - **Examples**: Java, C, C++

Dynamic Typing

- Often want to track the type in a variable
  - Would typing x+y cause an error?
  - Depends on whether x, y are **int**, **float**, or **str** values
- Use expression `type(<expression>)` to get type
  - `type(2)` evaluates to `<class 'int'>`
  - `type(x)` evaluates to type of contents of x
- What is a class?
  - In Python it is a synonym for a type
  - In Python 2, the word type would be there

Going Meta

- The types are themselves values!
  - Can assign them to variables: `x = int`
  - Can use them in expressions: `int == float`
- What is their type? It is **type**.
  - `type(int)` evaluates to `<class 'type'>`
- Can use in a boolean expression to test type
  - `type('abc') == str` evaluates to **True**
  - `type(x) == str` evaluates to **False**