Type: Set of values and the operations on them

- Type **int**:
  - **Values**: integers
  - **Ops**: +, –, *, /, %, **
- Type **float**:
  - **Values**: real numbers
  - **Ops**: +, –, *, /, **
- Type **bool**:
  - **Values**: **True** and **False**
  - **Ops**: not, and, or
- Type **str**:
  - **Values**: string literals
    - Double quotes: `"abc"`
    - Single quotes: `'abc'`
  - **Ops**: + (concatenation)

Will see more types in a few weeks

Operator Precedence

- What is the difference between the following?
  - 2*(1+3)
  - 2*1 + 3
- Operations are performed in a set order
  - Parentheses make the order explicit
  - What happens when there are no parentheses?
- **Operator Precedence**: The *fixed* order Python processes operators in *absence* of parentheses

Operator Precedence

- What is the difference between the following?
  - 2*(1+3) **add, then multiply**
  - 2*1 + 3 **multiply, then add**
- Operations are performed in a set order
  - Parentheses make the order explicit
  - What happens when there are no parentheses?
- **Operator Precedence**: The *fixed* order Python processes operators in *absence* of parentheses

Precedence of Python Operators

- **Exponentiation**: **
- **Unary operators**: + –
- **Binary arithmetic**: * / %
- **Binary arithmetic**: + –
- **Comparisons**: < > <= >=
- **Equality relations**: == !=
- **Logical not**
- **Logical and**
- **Logical or**

- Precedence goes downwards
  - Parentheses highest
  - Logical ops lowest
- Same line = same precedence
  - Read "ties" left to right
  - Example: 1/2*3 is (1/2)*3

Expressions vs Statements

Expression

- **Represents** something
  - Python *evaluates it*
  - End result is a value
- Examples:
  - `2.3` — Value
  - `(3+5)/4` — Complex Expression

Statement

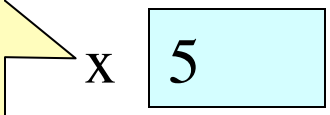
- **Does** something
  - Python *executes it*
  - Need not result in a value
- Examples:
  - `print "Hello"`
  - `import sys`

Will see later this is not a clear cut separation

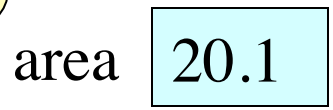
Variables (Section 2.1)

- A **variable**
  - is a **named** memory location (**box**)
  - contains a **value** (in the box)
  - can be used in expressions
- Examples:

Variable names must start with a letter (or _).



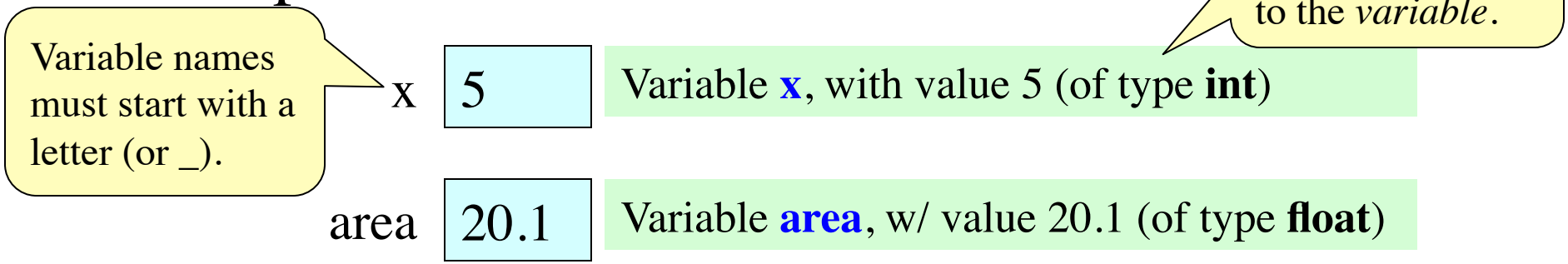
Variable **x**, with value 5 (of type **int**)



Variable **area**, w/ value 20.1 (of type **float**)

Variables (Section 2.1)

- A **variable**
  - is a **named** memory location (**box**)
  - contains a **value** (in the box)
  - can be used in expressions
- Examples:



Variables (Section 2.1)

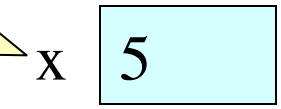
- A **variable**
  - is a **named** memory location (**box**)
  - contains a **value** (in the box)
  - can be used in expressions

The value in the box is then used in evaluating the expression.

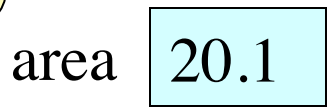
- Examples:

Variable names must start with a letter (or _).

The type belongs to the *value*, not to the *variable*.



Variable **x**, with value 5 (of type **int**)



Variable **area**, w/ value 20.1 (of type **float**)

Variables (Section 2.1)

- A **variable**
  - is a **named** memory location (**box**)
  - contains a **value** (in the box)
  - can be used in expressions

The value in the box is then used in evaluating the expression.

- Examples:

The type belongs to the *value*, not to the *variable*.

Variable names must start with a letter (or _).

x | 5 — Variable **x**, with value 5 (of type **int**)

area | 20.1 — Variable **area**, w/ value 20.1 (of type **float**)

1e2 is a **float**, but e2 is a variable name

Variables and Assignment Statements

- Variables are created by **assignment statements**
  - Create a new variable name and give it a value

```
x = 5
```

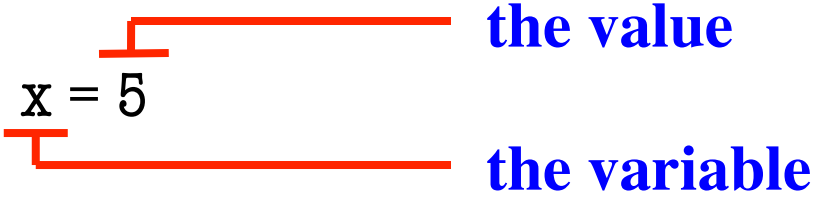
- This is a **statement**, not an **expression**
  - Tells the computer to DO something (not give a value)
  - Typing it into >>> gets no response (but it is working)
- Assignment statements can have expressions in them
  - These expressions can even have variables in them

```
x = x + 2
```

Two steps to execute an assignment:
1. evaluate the expression on the right
2. store the result in the variable on the left

Variables and Assignment Statements

- Variables are created by **assignment statements**
  - Create a new variable name and give it a value



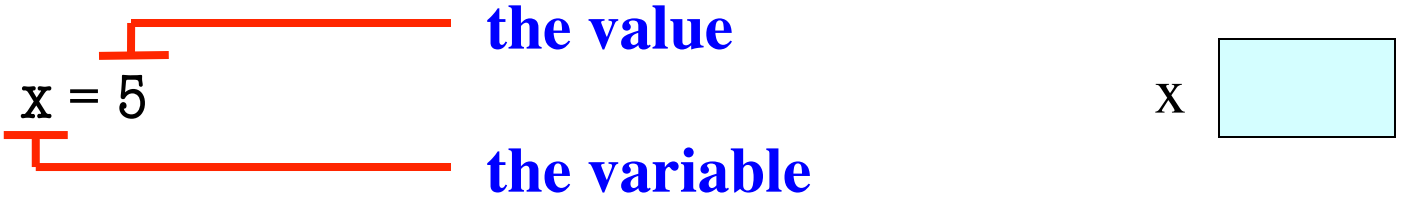
- This is a **statement**, not an **expression**
  - Tells the computer to DO something (not give a value)
  - Typing it into >>> gets no response (but it is working)
- Assignment statements can have expressions in them
  - These expressions can even have variables in them

```
x = x + 2
```

Two steps to execute an assignment:
1. evaluate the expression on the right
2. store the result in the variable on the left

Variables and Assignment Statements

- Variables are created by **assignment statements**
  - Create a new variable name and give it a value



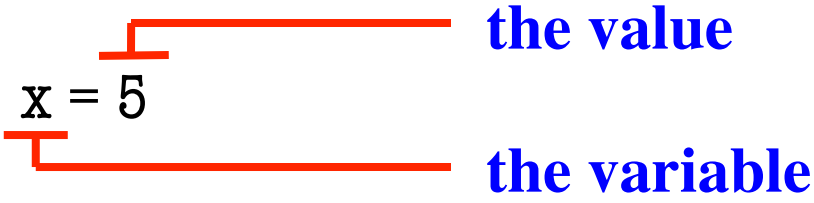
- This is a **statement**, not an **expression**
  - Tells the computer to DO something (not give a value)
  - Typing it into >>> gets no response (but it is working)
- Assignment statements can have expressions in them
  - These expressions can even have variables in them

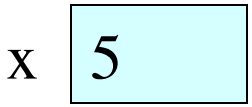
```
x = x + 2
```

Two steps to execute an assignment:
1. evaluate the expression on the right
2. store the result in the variable on the left

Variables and Assignment Statements

- Variables are created by **assignment statements**
  - Create a new variable name and give it a value





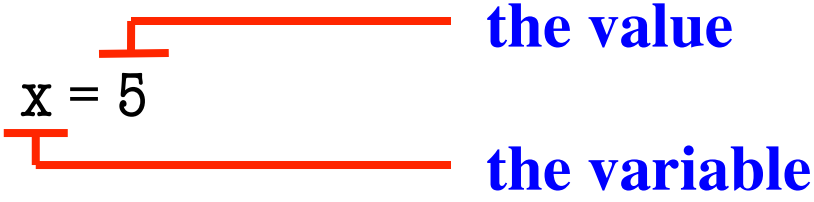
- This is a **statement**, not an **expression**
  - Tells the computer to DO something (not give a value)
  - Typing it into >>> gets no response (but it is working)
- Assignment statements can have expressions in them
  - These expressions can even have variables in them

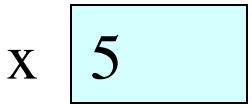
```
x = x + 2
```

Two steps to execute an assignment:
1. evaluate the expression on the right
2. store the result in the variable on the left

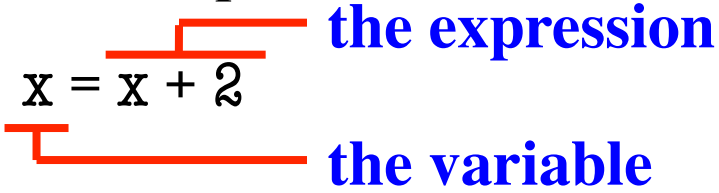
Variables and Assignment Statements

- Variables are created by **assignment statements**
  - Create a new variable name and give it a value





- This is a **statement**, not an **expression**
  - Tells the computer to DO something (not give a value)
  - Typing it into >>> gets no response (but it is working)
- Assignment statements can have expressions in them
  - These expressions can even have variables in them



Two steps to execute an assignment:
1. evaluate the expression on the right
2. store the result in the variable on the left

Variables and Assignment Statements

- Variables are created by **assignment statements**

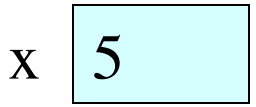
"gets"

Create a new variable name and give it a value

```
x = 5
```

the value

the variable



- This is a **statement**, not an **expression**
  - Tells the computer to DO something (not give a value)
  - Typing it into >>> gets no response (but it is working)
- Assignment statements can have expressions in them
  - These expressions can even have variables in them

the expression

```
x = x + 2
```

the variable

Two steps to execute an assignment:
1. evaluate the expression on the right
2. store the result in the variable on the left

Execute the statement: `x = x + 2`

- Draw variable x on piece of paper:

Execute the statement: `x = x + 2`

- Draw variable x on piece of paper:

- Step 1: evaluate the expression `x + 2`
  - For x, use the value in variable x
  - Write the expression somewhere on your paper

Execute the statement: `x = x + 2`

- Draw variable x on piece of paper:

- Step 1: evaluate the expression `x + 2`
  - For x, use the value in variable x
  - Write the expression somewhere on your paper
- Step 2: Store the value of the expression in `x`
  - Cross off the old value in the box
  - Write the new value in the box for x

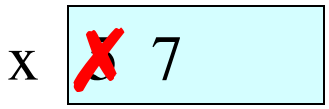
Execute the statement: `x = x + 2`

- Draw variable x on piece of paper:

- Step 1: evaluate the expression `x + 2`
  - For x, use the value in variable x
  - Write the expression somewhere on your paper
- Step 2: Store the value of the expression in `x`
  - Cross off the old value in the box
  - Write the new value in the box for x
- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Execute the statement: `x = x + 2`

- Draw variable x on piece of paper:

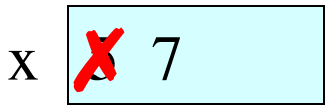


- Step 1: evaluate the expression `x + 2`
  - For x, use the value in variable x
  - Write the expression somewhere on your paper
- Step 2: Store the value of the expression in `x`
  - Cross off the old value in the box
  - Write the new value in the box for x
- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Execute the statement: x = x + 2

A: I did it correctly!
B: I drew another box named x
C: I did something else
D: I did nothing—just watched

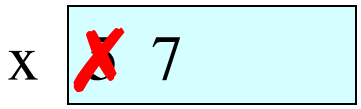
- Draw variable x on piece of pa



- Step 1: evaluate the expression x + 2
  - For x, use the value in variable x
  - Write the expression somewhere on your paper
- Step 2: Store the value of the expression in x
  - Cross off the old value in the box
  - Write the new value in the box for x
- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

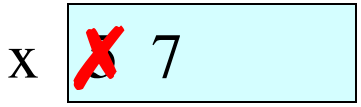
Execute the statement: `x = 3.0 * x + 1.0`

- You have this:



Execute the statement: `x = 3.0 * x + 1.0`

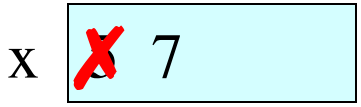
- You have this:



- Execute this command:
  - Step 1: **Evaluate** the expression `3.0 * x + 1.0`
  - Step 2: **Store** its value in x

Execute the statement: `x = 3.0 * x + 1.0`

- You have this:



- Execute this command:
  - Step 1: **Evaluate** the expression `3.0 * x + 1.0`
  - Step 2: **Store** its value in x
- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Execute the statement: x = 3.0 * x + 1.0

- You have this:

  x | ~~7.0~~ 22.0

- Execute this command:
  - Step 1: **Evaluate** the expression 3.0 * x + 1.0
  - Step 2: **Store** its value in x
- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Execute the statement: x = 3.0 * x + 1.0

A: I did it correctly!
B: I drew another box named x
C: I did something else
D: I did nothing –just watched

- You have this:

  x 22.0

- Execute this command:
  - Step 1: **Evaluate** the expression 3.0 * x + 1.0
  - Step 2: **Store** its value in x
- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Execute the statement: `x = 3.0 * x + 1.0`

- You now have this:

  x ~~7.0~~ 22.0

- The command:
  - Step 1: **Evaluate** the expression `3.0 * x + 1.0`
  - Step 2: **Store** its value in x
- This is how you execute an assignment statement
  - Performing it is called **executing the command**
  - Command requires both **evaluate** AND **store** to be correct
  - Important *mental model* for understanding Python

Exercise: Understanding Assignment

- Add another variable, interestRate, to get this:

- Execute this assignment:

```
interestRate = x / interestRate
```

- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Exercise: Understanding Assignment

- Add another variable, interestRate, to get this:

- Execute this assignment:

```
interestRate = x / interestRate
```

- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Exercise: Understanding Assignment

- Add another variable, interestRate, to get this:

- Execute this assignment:

```
interestRate = x / interestRate
```

- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

A: I did it correctly!
B: I drew another box called "interestRate"
C: I stored the value in the box for x
D: I thought it would use **int** division
E: I did something else (or nothing)

Exercise: Understanding Assignment

- You now have this:

- Execute this assignment:

```
intrestRate =  x + interestRate
```

- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Exercise: Understanding Assignment

- You now have this:

x ~~22.0~~ 22.0 interestRate ~~5.5~~ 5.5 intrestRate 27.5

- Execute this assignment:

```
intrestRate = x + interestRate
```

- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Exercise: Understanding Assignment

- You now have this:

x ~~22.0~~ | interestRate ~~5.5~~ | intrestRate 27.5

- Execute this assignment:

```
intrestRate = x + interestRate
```

- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

A: I did it correctly!
B: I stored the value in "interestRate"
C: I stored the value in x
D: I did something else (or nothing)

Exercise: Understanding Assignment

- You now have this:

x ~~22.0~~ | interestRate ~~5.5~~ | intrestRate 27.5

- Execute this assignment:

```
intrestRate = x + interestRate
```

- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Spelling mistakes in Python are bad!!

A: I did it correctly!
B: I stored the value in "interestRate"
C: I stored the value in x
D: I did something else (or nothing)

Dynamic Typing

- Python is a **dynamically typed language**
  - Variables can hold values of any type
  - Variables can hold different types at different times
  - Use `type(x)` to find out the type of the value in x
  - Use names of types for conversion, comparison
- The following is acceptable in Python:

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

- Alternative is a **statically typed language** (e.g. Java)
  - Each variable restricted to values of just one type

```
type(x) == int
x = float(x)
type(x) == float
```

Dynamic Typing

- Python is a **dynamically typed language**
  - Variables can hold values of any type
  - Variables can hold different types at different times
  - Use `type(x)` to find out the type of the value in x
  - Use names of types for conversion, comparison

```
type(x) == int
x = float(x)
type(x) == float
```

- The following is acceptable in Python:

```
>>> x = 1
```
← x contains an **int** value

```
>>> x = x / 2.0
```
← x now contains a **float** value

- Alternative is a **statically typed language** (e.g. Java)
  - Each variable restricted to values of just one type

Dynamic Typing

- Often want to track the type in a variable
  - What is the result of evaluating x / y?
  - Depends on whether x, y are **int** or **float** values
- Use expression `type(<expression>)` to get type
  - `type(2)` evaluates to `<type 'int'>`
  - `type(x)` evaluates to type of contents of x
- Can use in a boolean expression to test type
  - `type('abc') == str` evaluates to **True**