http://www.cs.cornell.edu/courses/cs1110/2019sp

Lecture 2: Variables & Assignments

(Sections 2.1-2.3,2.5)

CS 1110

Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

Helping you succeed in this class

Canvas. You should have received an email on Tuesday. Check your spam folders. This is how we make announcements.

http://www.cs.cornell.edu/courses/cs1110/2019sp/staff/

Consulting Hours. ACCEL Lab Green Room

- Big block of time multiple people (see staff calendar)
- Good for assignment help

Staff Office Hours.

- Staff: 1 person, 1 hour at a time (see staff calendar)
- Good for conceptual help

My Office Hours. Right next door after class (Baker 219) or by appointment (see staff page under "Office Hours")

Piazza. Online forum to ask/answer questions

From last time: **Types**

Type: set of values & operations on them

Type **float:**

- Values: real numbers
- Ops: +, -, *, /, **

Type **int:**

- Values: integers
- Ops: +, -, *, //, %, **

Type **bool:**

- Values: true, false
- Ops: not, and, or

Type **str:**

- Values: string literals
  - Double quotes: **“abc”**
  - Single quotes: **‘abc’**
- Ops: + (concatenation)

Converting from one type to another

aka “casting”

<type>(*<value>*)

```
>>> float(2)
2.0

>>>int(2.6)
2



>>>type(2)
<class ‘int’>
```

converts value 2 to type **float**

converts value 2.6 to type **int**

…different from:

type(*<value>*)

tells you the type

What should Python do?

```
>>> 1/2.6
```

(A) turn 2.6 into the integer 2, then calculate 1/2 → 0.5

(B) turn 2.6 into the integer 2, then calculate 1//2 → 0

(C) turn 1 into the float 1.0, then calculate 1.0/2.6 → 0.3846…

(D) Produce a `TypeError` telling you it cannot do this.

(E) Exit Python

Widening Conversion (OK!)

From a **narrower** type to a **wider** type
(e.g., `int` → `float`)

Python does automatically if needed:

- Example: `1/2.0` evaluates to a `float`: `0.5`
- Example: `True + 1` evaluates to an `int`: `2`
  - `True` converts to `1`
  - `False` converts to `0`

Note: does not work for **str**

- Example: 2 + “ab” produces a TypeError

Narrowing Conversion (OK???)

From a **wider** type to a **narrower** type

(e.g., `float` → `int`)

- causes information to be lost
- Python **never** does this automatically

What about:

```
>>> 1/int(2.6)
0.5
```

Python casts the 2.6 to 2.0 but / is a float division, so Python casts 1 to 1.0 and 2 to 2.0

Types matter!

You Decide:

- What is the right type for my data?
- When is the right time for conversion (if any)

- Zip Code as an **`int`**?
- Grades as an **`int`**?
- Lab Grades as a **`bool`**?
- Interest level as **`bool`** or **`float`**?

What are your goals:

Accuracy? Clarity? Fairness?

Operator Precedence

What is the difference between:

2*(1+3)	2*1 + 3
add, then multiply	***multiply, then add***

Operations performed in a set order

- Parentheses make the order explicit

What if there are no parentheses?

→ **Operator Precedence:** fixed order to processes operators when no 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 (except for **)
  - Example: 1/2*3 is (1/2)*3

- Section 2.5 in your text
- See website for more info
- Major portion of Lab 1

Operators and Type Conversions

Operator Precedence

Exponentiation: **

Unary operators: + –

Binary arithmetic: * / %

Binary arithmetic: + –

Comparisons: < > <= >=

Equality relations: == !=

Logical not

Logical and

Logical or

Evaluate this expression:

False + 1 + 3.0 / 3

A. 3

B. 3.0

C. 1.3333

D. 2

E. 2.0

Operators and Type Conversions

Operator Precedence

Exponentiation: **

Unary operators: + –

Binary arithmetic: * / %

Binary arithmetic: + –

Comparisons: < > <= >=

Equality relations: == !=

Logical not

Logical and

Logical or

Evaluate this expression:

$$\texttt{False + 1 + }\underline{\texttt{3.0 / 3}}$$

$$\underline{\texttt{False + 1}}\texttt{ + 1.0}$$

$$\underline{\texttt{1 + 1.0}}$$

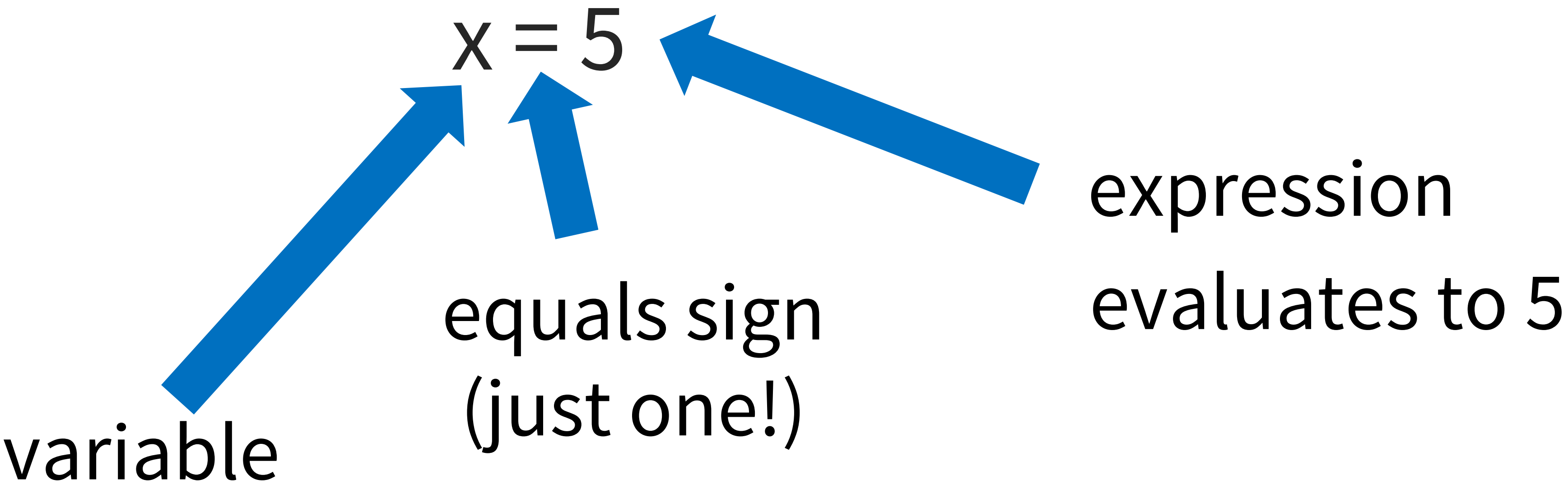
$$\texttt{2.0}$$

New Tool: Variable Assignment

An *assignment statement:*

- takes an *expression*
- evaluates it, and
- stores the *value* in a *variable*

Example: (read right to left)



x = 5

variable

equals sign
(just one!)

expression
evaluates to 5

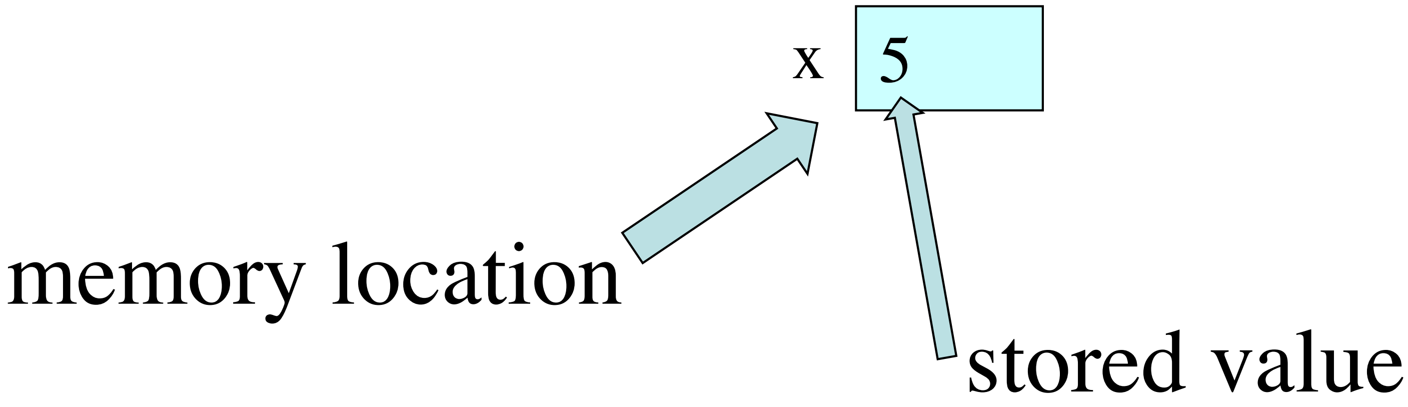
Executing Assignment Statements

```
>>> x = 5
>>>
```

Press ENTER and…

Hmm, looks like nothing happened…

- But something did happen!
- Python *assigned* the *value* 5 to the *variable* x
- Internally (and invisible to you):



x

5

memory location

stored value

Retrieving Variables

```
>>> x = 5
>>> x
5
>>>
```

Press ENTER and…

Interactive mode tells me the value of x

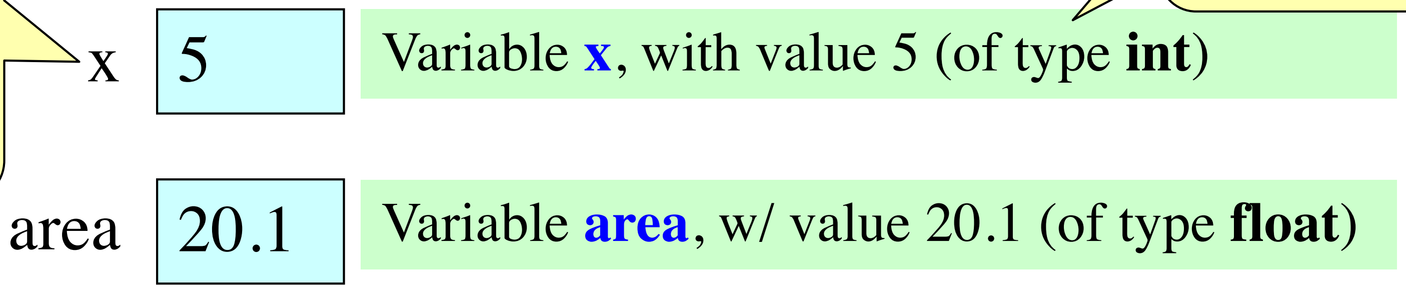
In More Detail: Variables (Section 2.1)

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

- 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**)

In More Detail: Statements

```
>>> x = 5
>>>
```

Press ENTER and…

Hm, looks like nothing happened…

- 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)

Expressions vs. Statements

Expression

- **Represents** something
  - Python *evaluates it*
  - End result is a value
- Examples:
  - `2.3`

    Value
  - `(3+5)/4`

    Complex Expression
  - `x == 5`

Statement

- **Does** something
  - Python *executes it*
  - Need not result in a value
- Examples:
  - `x = 2 + 1`
  - `x = 5`

Look so similar

but they are not!

You can assign more than literals

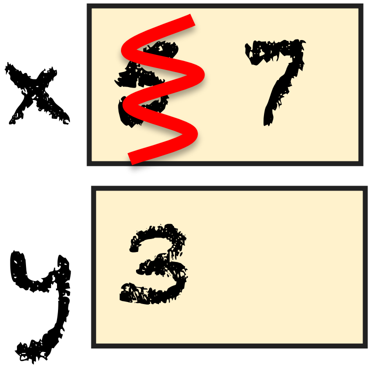
```
>>> x = 5
>>> x = 3.0 ** 2 + 4 – 1
>>> x = 2 + x
```

“x gets 5”

“x gets the value of this expression”

“x gets 2 plus the current value of x”

Keeping Track of Variables



- Draw boxes on paper:

```
>>> x = 5
```

- New variable declared?

```
>>> y = 3
```

Write a new box.

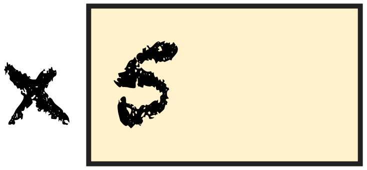
- Variable updated?

```
>>> x = 7
```

Cross out old value. Insert new value.

Execute the Statement: `x = x + 2`

Draw variable x on piece of paper:

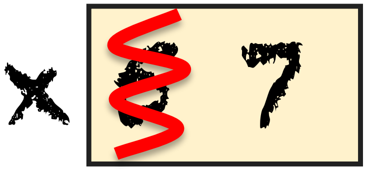


1. Evaluate the expression `x + 2`
   - For `x`, use the value in variable `x`
   - Write the expression somewhere on your paper
2. Store the value of the expression in `x`
   - Cross off the old value in the box
   - Write the new value in the box for `x`

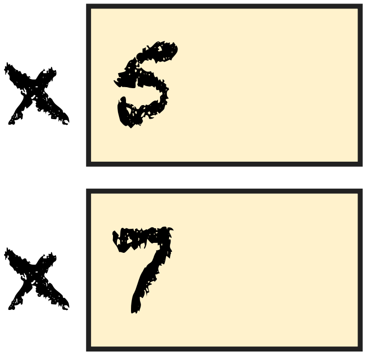
Did you do the same thing as your neighbor ?
If not, *discuss*.

Which one is closest to your answer?

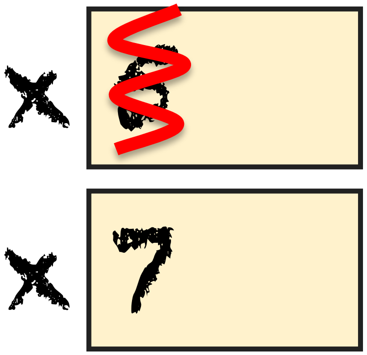
A.



B.



C.



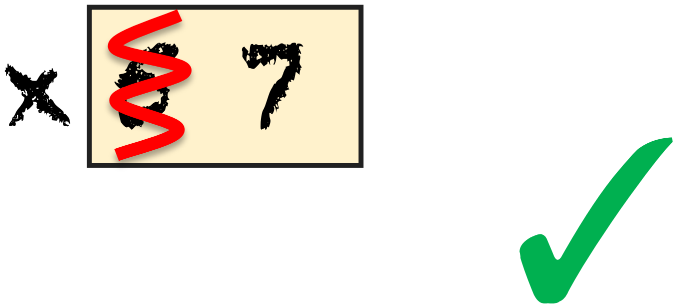
D.

¯_(ツ)_/¯

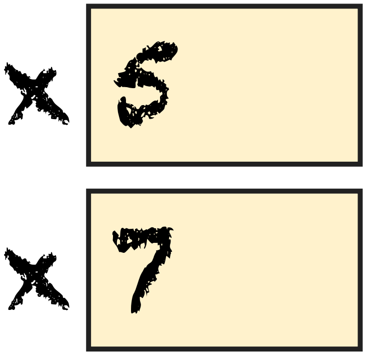
x = x + 2

And The Correct Answer Is…

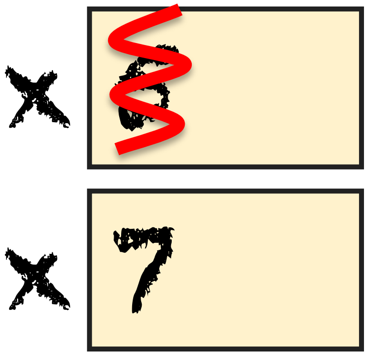
A.



B.



C.



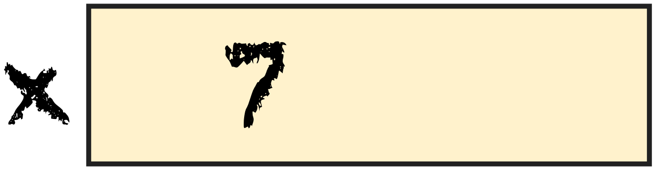
D.

¯_(ツ)_/¯

x = x + 2

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

Begin with this:

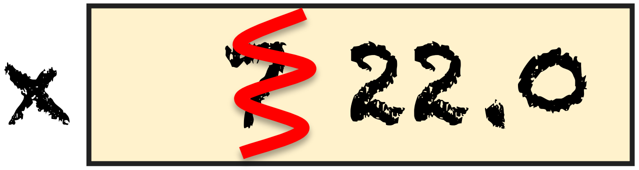


1. **Evaluate** the expression `3.0*x+1.0`
2. **Store** its value in `x`

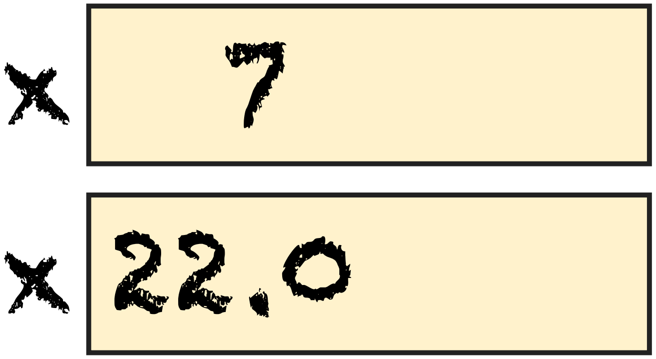
Did you do the same thing as your neighbor ? If not, *discuss*.

Which one is closest to your answer?

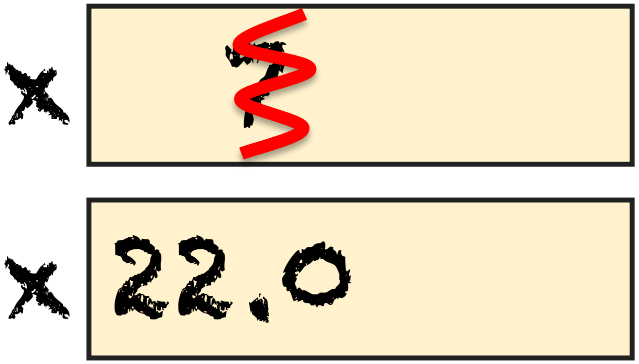
A.



B.



C.



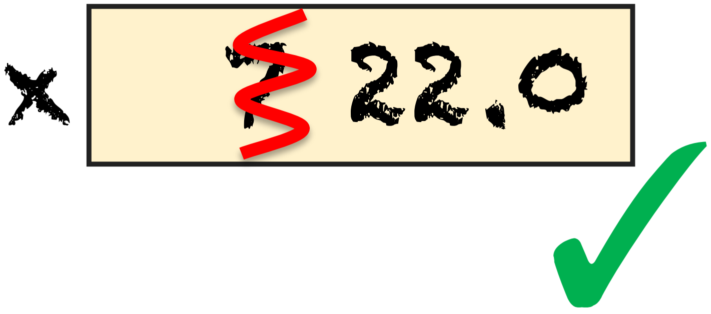
D.

¯_(ツ)_/¯

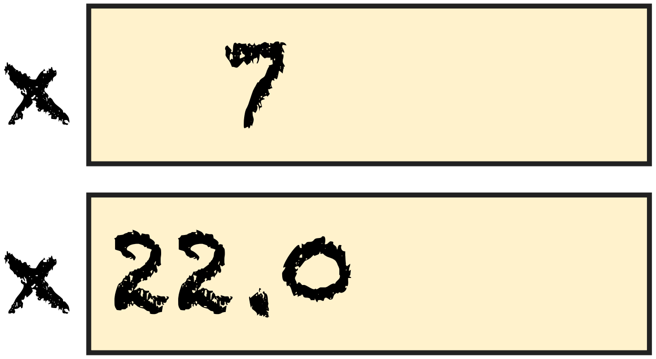
```
x = 3.0*x+1.0
```

And The Correct Answer Is…

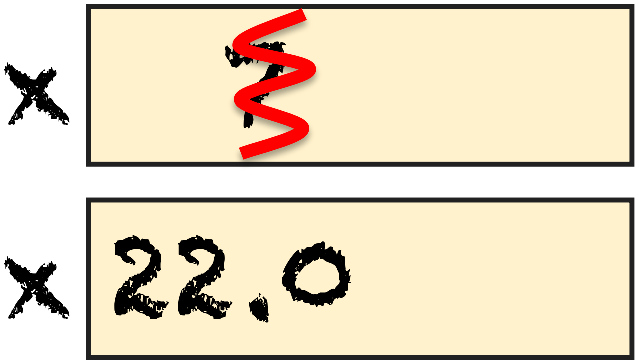
A.



B.



C.



D.

¯_(ツ)_/¯

`x = 3.0*x+1.0`

Executing an Assignment Statement

The command: `x = 3.0*x+1.0`

"Executing the command":

1. **Evaluate** right hand side `3.0*x+1.0`
2. **Store** the value in the variable `x`'s box

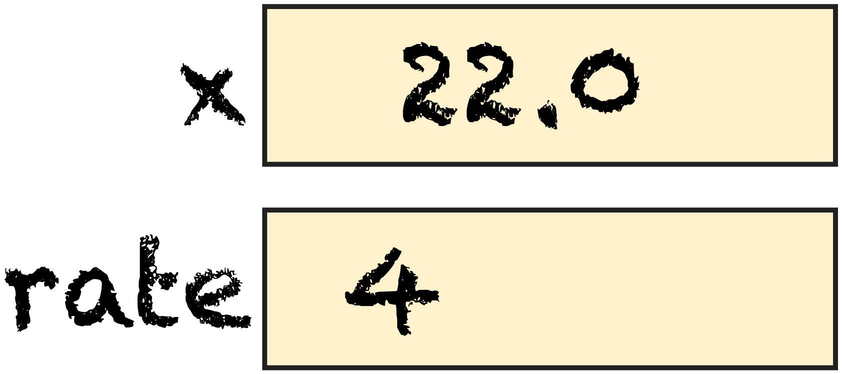
- Requires both evaluate AND store steps
- Critical mental model for learning Python

Exercise 1: Understanding Assignment

Begin with:

Declare a new variable:

```
>>> rate = 4
```

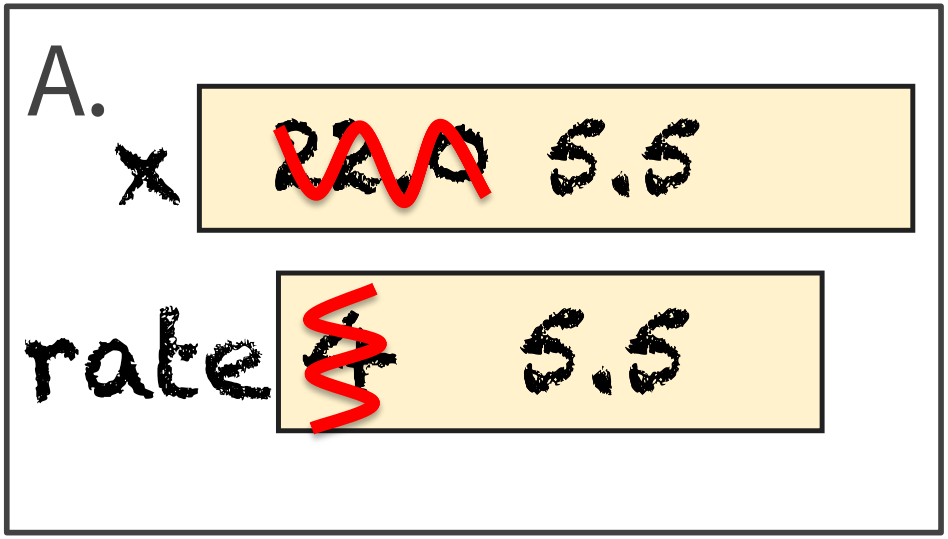


Execute this assignment:

```
>>> rate = x / rate
```

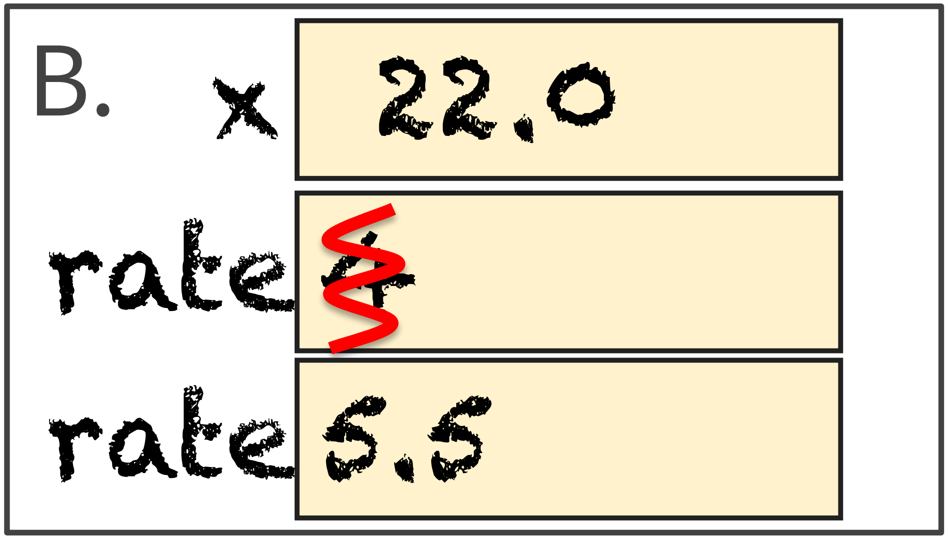
Did you do the same thing as your neighbor ? If not, *discuss*.

Which one is closest to your answer?


A.

x ~~22.0~~ 5.5

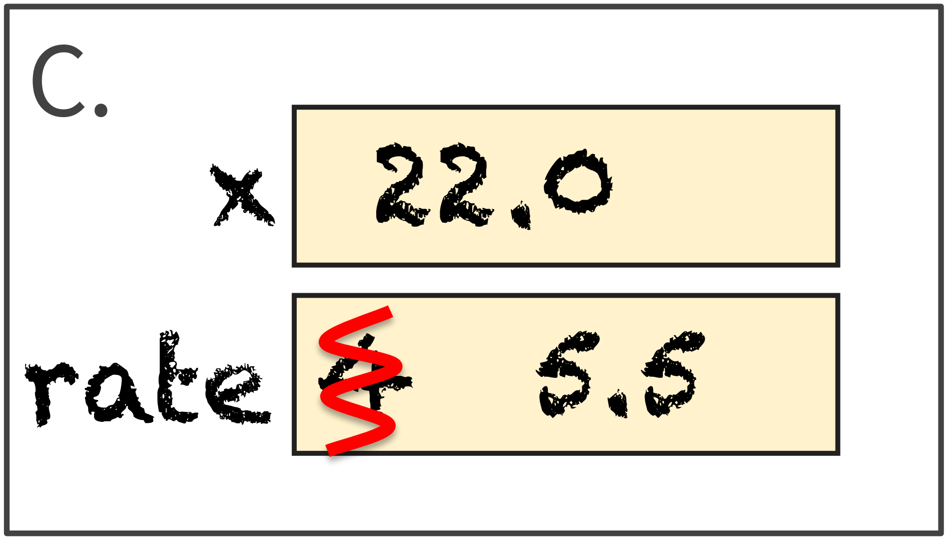
rate ~~4~~ 5.5


B.

x 22.0

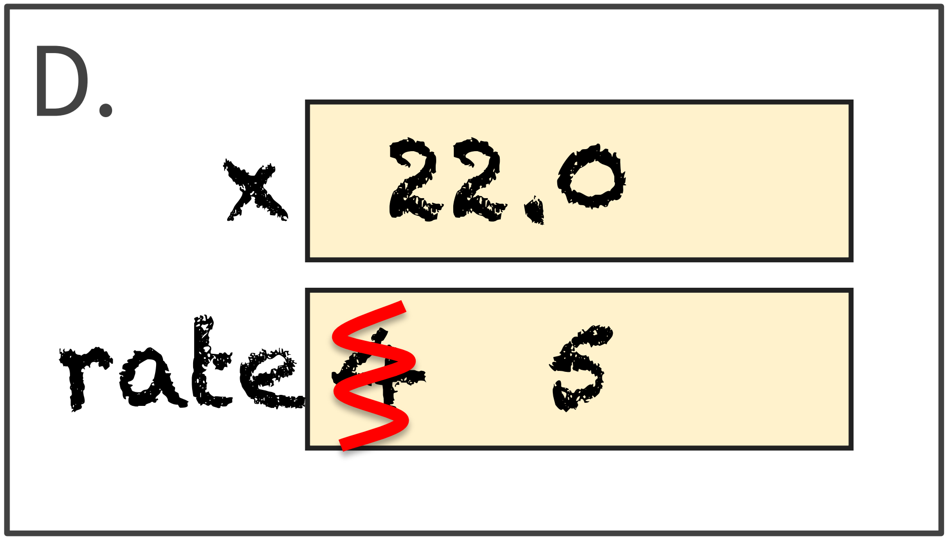
rate ~~4~~

rate 5.5


C.

x 22.0

rate ~~4~~ 5.5


D.

x 22.0

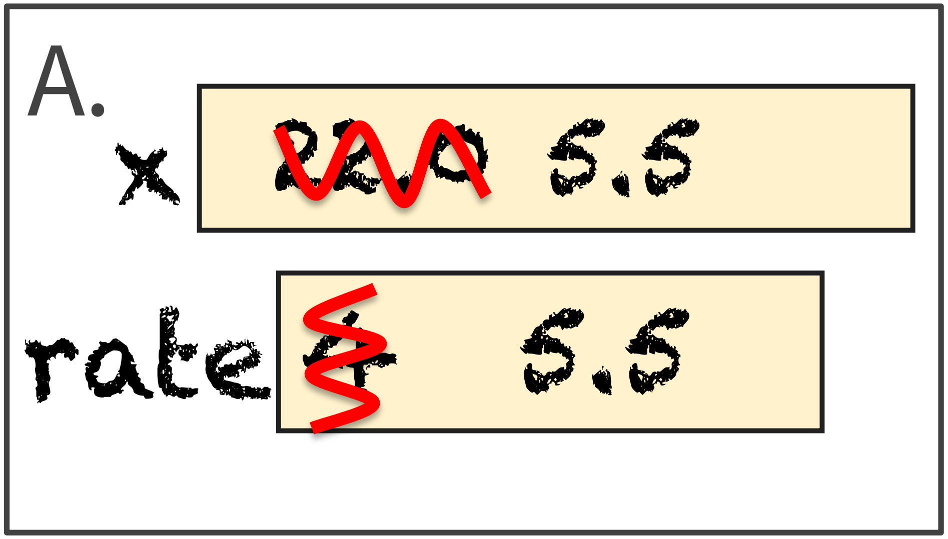
rate ~~4~~ 5

E. ¯_(ツ)_/¯

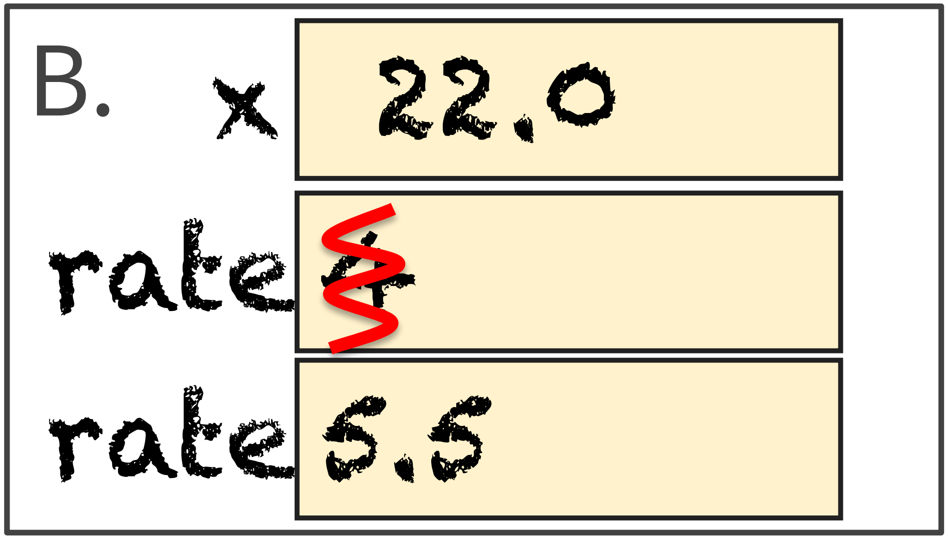
rate = x / rate

And The Correct Answer Is…

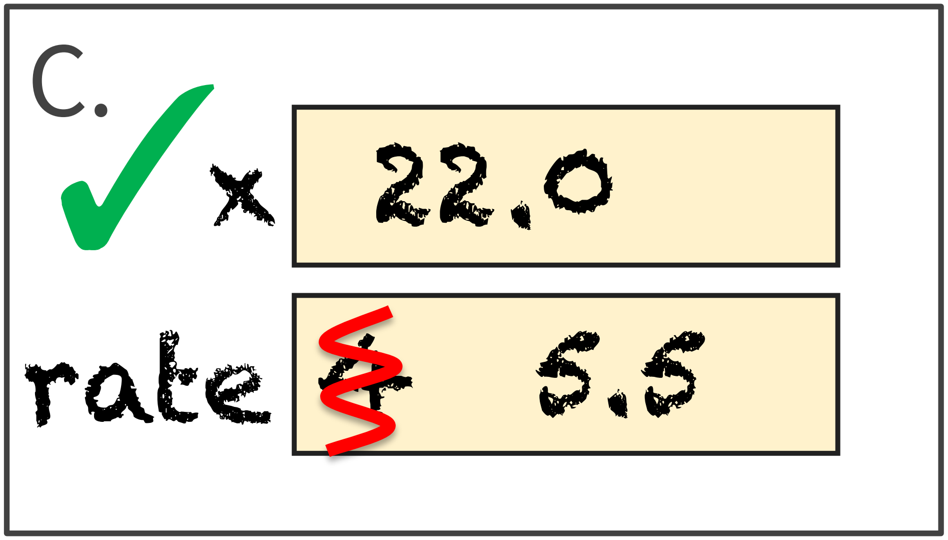
A.



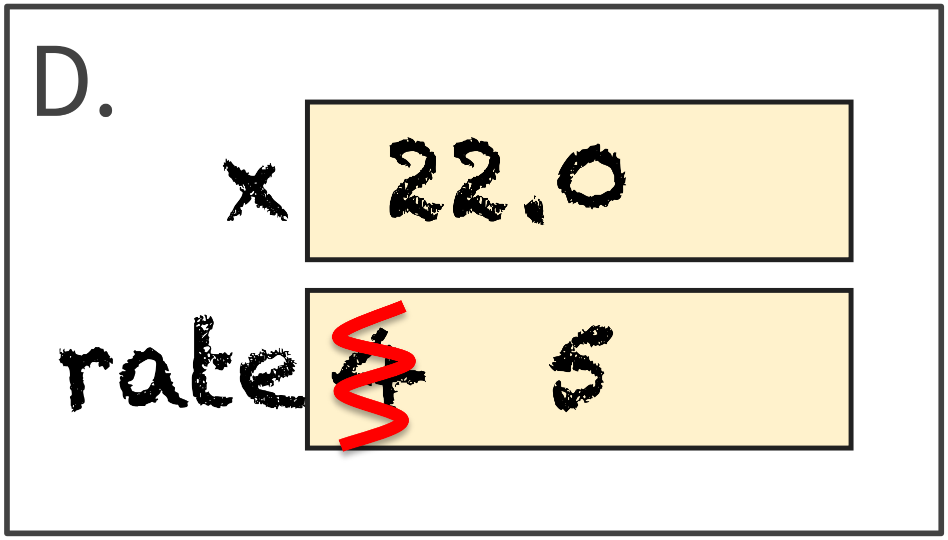
B.



C.



D.



```
rate = x / rate
```

Dynamic Typing

Python is a **dynamically typed** language

- Variables can hold values of any type
- Variables can hold different types at different times

The following is acceptable in Python:

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

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

Alternative: a **statically typed** language

- Examples: Java, C
- Each variable restricted to values of just one type

More Detail: Testing Types

Command: `type(<value>)`

Can test a variable:

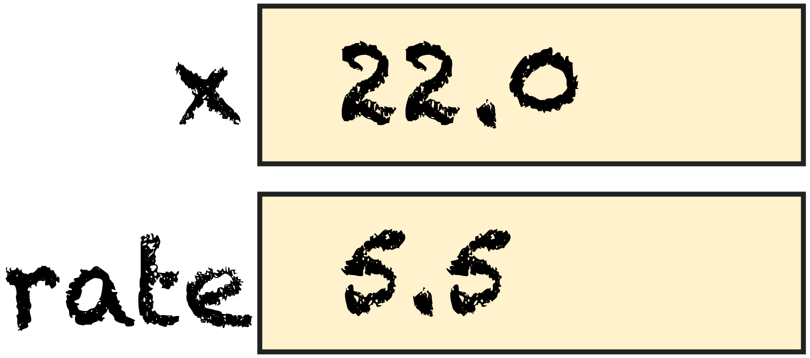
```
>>> x = 5
>>> type(x)
<class 'int‘>
```

Can test a type with a Boolean expression:

```
>>> type(2) == int
True
```

Exercise 2: Understanding Assignment

Begin with:



Execute this assignment:

```
>>> rat = x + rate
```

Did you do the same thing as your neighbor ? If not, *discuss*.

Which one is closest to your answer?

A.

x ~~22.0~~ 27.5

rate 5.5

B.

x 22.0

rate 5.5

rat 27.5

C.

x 22.0

rate ~~5.5~~ 27.5

D.

x 22.0

rate ~~5.5~~

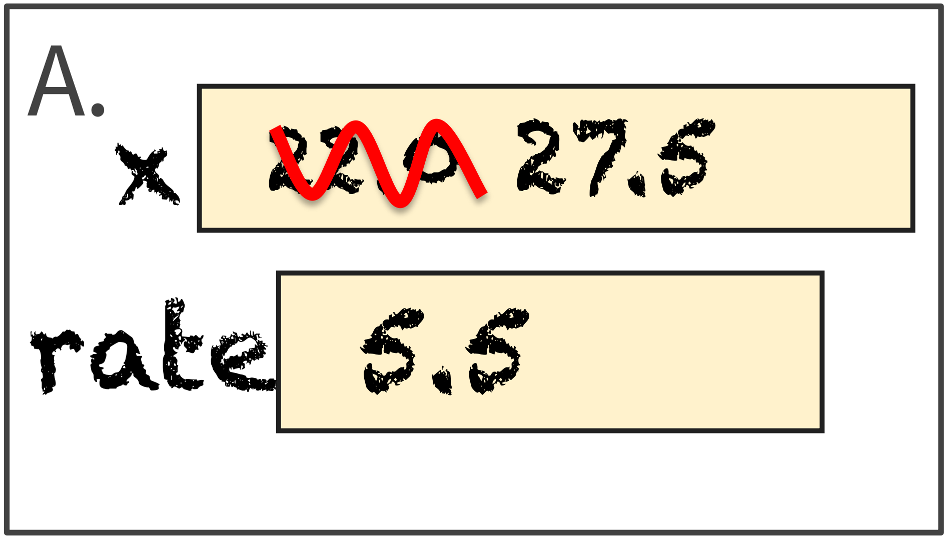
rat 27.5

E. ¯_(ツ)_/¯

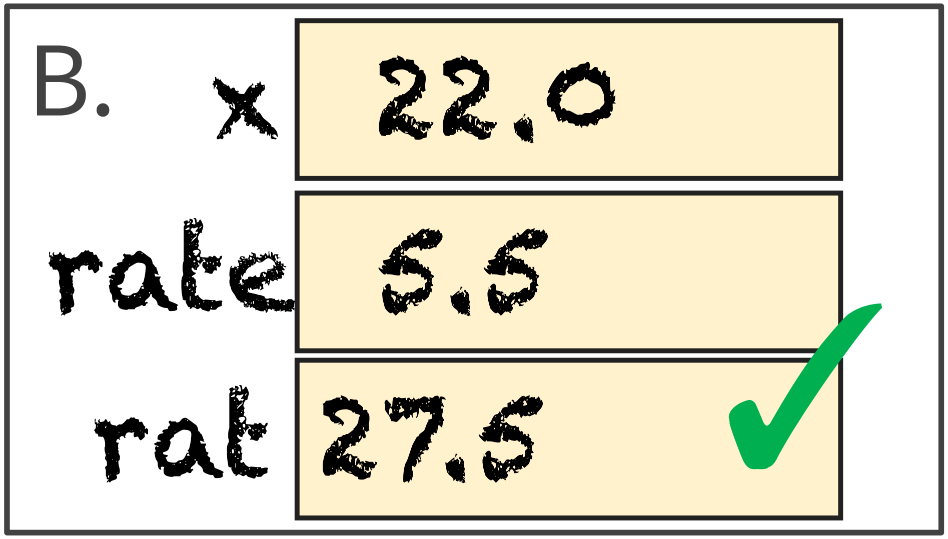
rat = x + rate

And The Correct Answer Is…

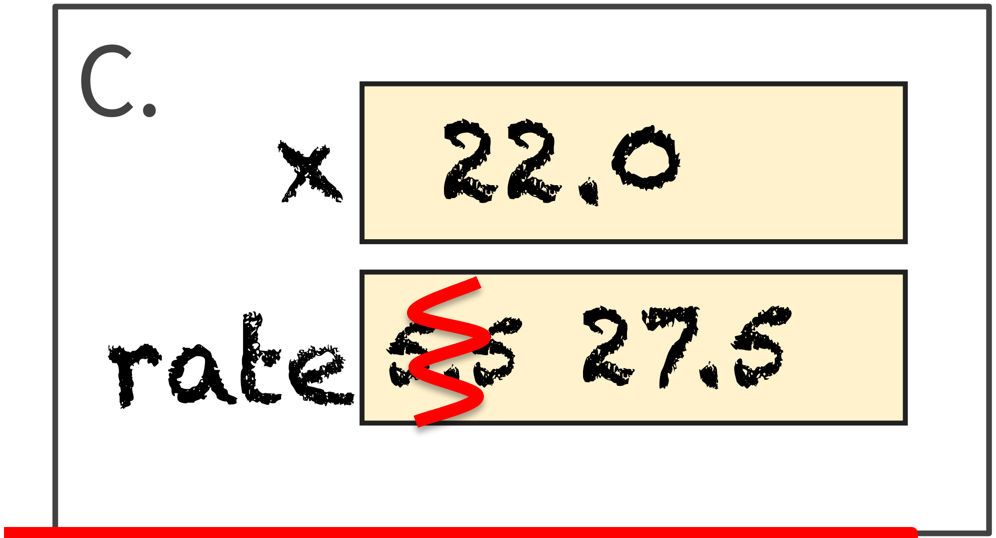
A.



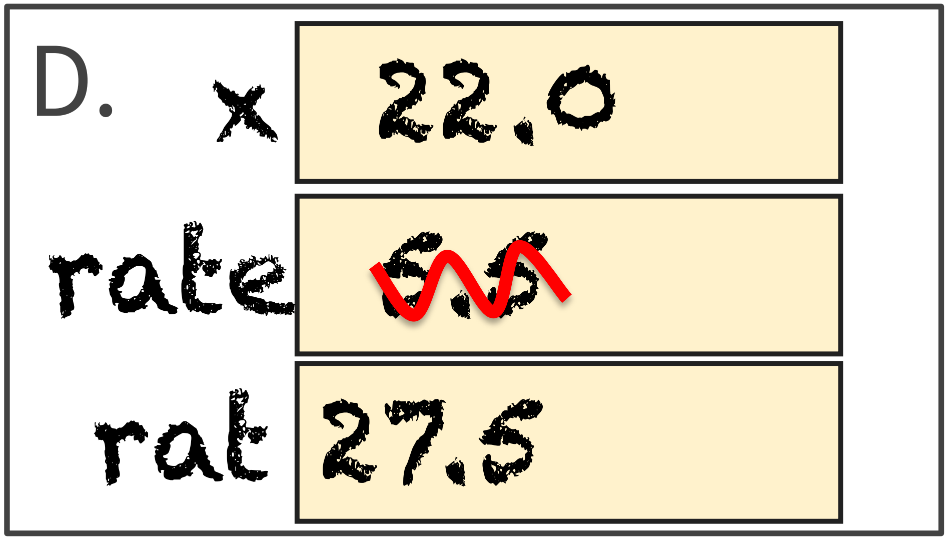
B.



C.



D.



Spelling Matters!

rat = x + rate