Lecture 2

Variables & Assignment

Announcements for Today

If Not Done Already

- Install Python
  - Make sure right version
  - Make sure Kivy works
- Register your iClicker
- Sign into CMS
  - Fill out the Survey
  - Complete AI Quiz

Lab 1

- Labs are due at next class
  - So lab 1 is due **now**
  - By *end* of the lab section
  - Try to finish them *before*
- Makes T/W a little tight
  - Only 2 days (not 5)
  - Will keep them small
- Getting behind is bad!

Helping You Succeed in this Class

- **Consultants.** Phillips 318 (after hours)
  - Daily office hours (see website) with consultants
  - Very useful when working on assignments
- **AEW Workshops**. Additional discussion course
  - Runs parallel to this class – completely optional
  - See website; talk to advisors in Olin 167.
- **Ed Discussions.** Forum to ask and answer questions
  - Go here first **before** sending question in e-mail
- **Office Hours.** Talk to the professor!
  - Couches in Statler Balcony between classes

Labs vs. Assignments

Labs

- Held twice a week
- Graded on **completeness**
  - Always S/U
  - Try again if not finished
- Indirect affect on grade
  - Can miss up to 3 labs
  - After that, grade reduced
- Similar to language drills
  - Simple, but take time

Assignments

- Every two weeks
  - First one due Sep. 15
- Graded on **correctness**
  - Assign points out of 100
- But **first** one is for *mastery*
  - Resubmit until perfect grade
- 40% of your final grade
- Can work with a partner!
  - Mixer coming soon

Academic Integrity

- Every semester we have cases of *plagiarism*
  - Claiming the work of others as your own
  - This is an **Academic Integrity violation**
- This course has a very specific policy
  - Do not listen to (non-staff) upperclassmen
  - Look at the course website for the new details
- Complete Academic Integrity Quiz on CMS
  - Must complete successfully to stay in class

AI(ntegrity) on AI(intelligence)

- Automated tools (ChatGPT) have gotten good
  - Can do a lot in an introductory course
  - Will change how we develop software
- But they also interfere with learning
  - This is a course about the basics
  - You need to learn to write your own code
- Use of automated tools is **restricted**
  - Can use to generate examples for yourself
  - But cannot use it **for any code you submit to us**

iClickers

- Have you registered your iClicker?
- If not, visit (free service; no surcharge!)
  - https://cs1110.cs.cornell.edu/py/clicker
- See the course web page for more:
  - http://www.cs.cornell.edu/courses/cs1110/2024fa
  - Click "Materials/Textbook"
  - Look under "iClickers"

Warm-Up: Using Python

- How do you plan to use Python?

A. I want to work mainly in the Phillips lab
B. I want to use my own Windows computer
C. I want to use my own Macintosh computer
D. I want to use my own Linux computer
E. I will use whatever I can get my hands on

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

Example: bool

- **Values:** True, False
  - That is it.
  - Must be capitalized!
- **Three Operations**
  - b and c
    (True if **both** True)
  - b or c
    (True if **at least one** is)
  - not b
    (True if b is **not**)
- Made by **comparisons**
  - **int**, **float** operations
  - But produce a **bool**
- Order comparisons:
  - i < j, i <= j
  - i >= j, i > j
- Equality, inequality:
  - i == j (**not** =)
  - i != j

Example: str

- **Values:** text, or *sequence of characters*
  - String literals must be in quotes
  - Double quotes: `"Hello World!"`, `" abcex3$g<&"`
  - Single quotes: `'Hello World!'`, `' abcex3$g<&'`
- **Operation:** + (catenation, or concatenation)
  - `'ab' + 'cd'` evaluates to `'abcd'`
  - concatenation can only apply to strings
  - `'ab' + 2` produces an **error**

Converting Values Between Types

- Basic form: *type*(*expression*)
  - This is an expression
  - Evaluates to value, converted to new type
  - This is sometimes called **casting**
- **Examples:**
  - `float`(2) evaluates to `2.0` (a **float**)
  - `int`(2.6) evaluates to `2` (an **int**)
  - Note information loss in 2nd example

Converting Values Between Types

- Conversion is measured *narrow* to *wide*

bool ⇒ int ⇒ float

- **Widening:** Convert to a wider type
  - Python does automatically
  - **Example:** `1/2.0` evaluates to `0.5`
- **Narrowing:** Convert to a narrower type
  - Python never does automatically
  - **Example:** `float(int(2.6))` evaluates to `2.0`

Operator Precedence

- What is the difference between these two?
  - 2*(1+3)
  - 2*1 + 3

Operator Precedence

- What is the difference between these two?
  - 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 no parentheses?

Operator Precedence

- What is the difference between these two?
  - 2*(1+3) **add, then multiply**
  - 2*1 + 3 **multiply, then add**
- O
  -
  -

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

- There is a video about this
- See website for more info
- Was major portion of Lab 1

Expressions vs Statements

Expression

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

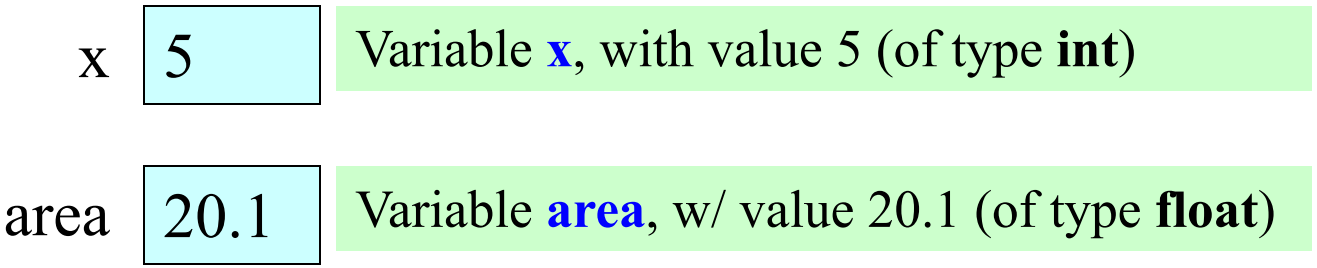
Statement

- **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

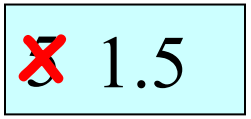
Variables

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



Using Variables

- Variables can be used in expressions
  - Evaluate to the value that is in the box
  - **Example**: `x` 

`1 + x` evaluates to **6**
- Variables can change values
  - **Example**: `x` 

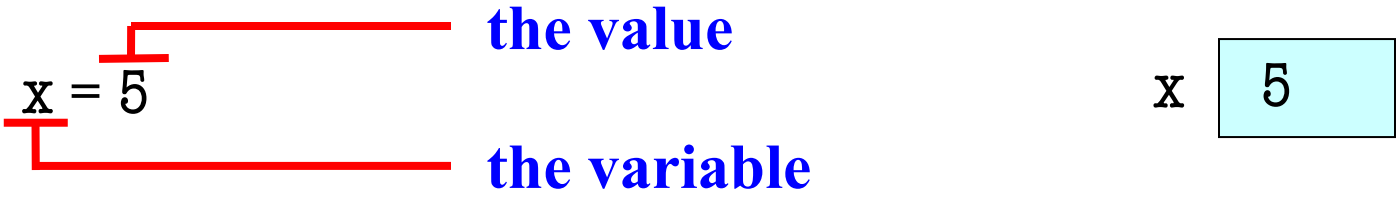
`1 + x` evaluates to **2.5**
  - Can even change the **type** of their value
  - Different from other languages (e.g. Java)

Naming Variables

- Python has strict rules of how to assign names
  - Names must only contain letters, numbers, _
  - They cannot start with a number
- **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 + on two variables)

Variables and Assignment Statements

- 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
  - Difference is that has no value itself
- **Example**:

```
>>> x = 5
(NOTHING)
```

But can now use `x` as an expression

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

Assignments May Contain Expressions

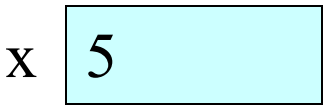
- **Example**: `x = 1 + 2`
  - Left of equals must always be variable: `1 + 2 = x`
  - Read assignment statements right-to-left!
  - Evaluate the expression on the right
  - Store the result in the variable on the left
- We can include variables in this expression
  - **Example**: `x = y+2`
  - **Example**: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

x	5
y	2

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

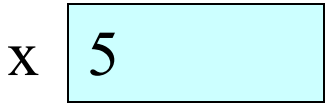
Execute the Statement: x = x + 2

- Draw variable x on piece of paper:



Execute the Statement: x = x + 2

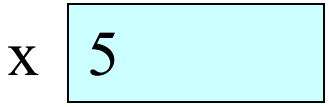
- Draw variable x on piece of paper:



- 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:

  x | 5

- 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.

Which One is Closest to Your Answer?

A:

x 7

B:

x 5

x 7

C:

x

x 7

D:

¯_(ツ)_/¯

Which One is Closest to Your Answer?

A:

x 7

B:

x 5

x 7

C:

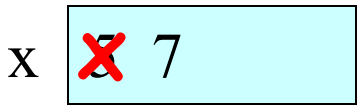
x

x 7

x = x + 2

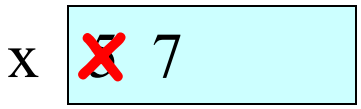
Execute the Statement: x = 3.0 * x + 1.0

- You have this:



Execute the Statement: x = 3.0 * x + 1.0

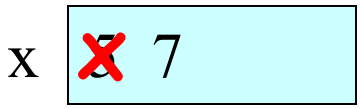
- You have this:



- 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.

Which One is Closest to Your Answer?

A:

x ~~7~~ ~~7~~ 22.0

B:

x ~~7~~ 7

x 22.0

C:

x ~~7~~ ~~7~~

x 22.0

D:

¯_(ツ)_/¯

Which One is Closest to Your Answer?

A:

x 22.0

B:

x 7

x 22.0

C:

x

x 22.0

x = 3.0 * x + 1.0

Execute the Statement: x = 3.0 * x + 1.0

- You now have this:

  x | ~~7.5~~ ~~7.5~~ 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:

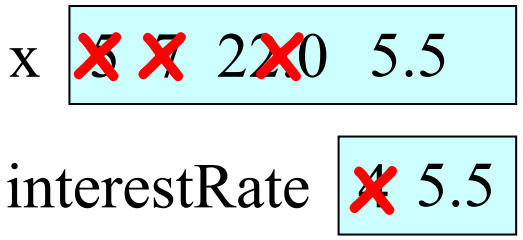
- Execute this assignment:

```
interestRate = x / interestRate
```

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

Which One is Closest to Your Answer?

A:

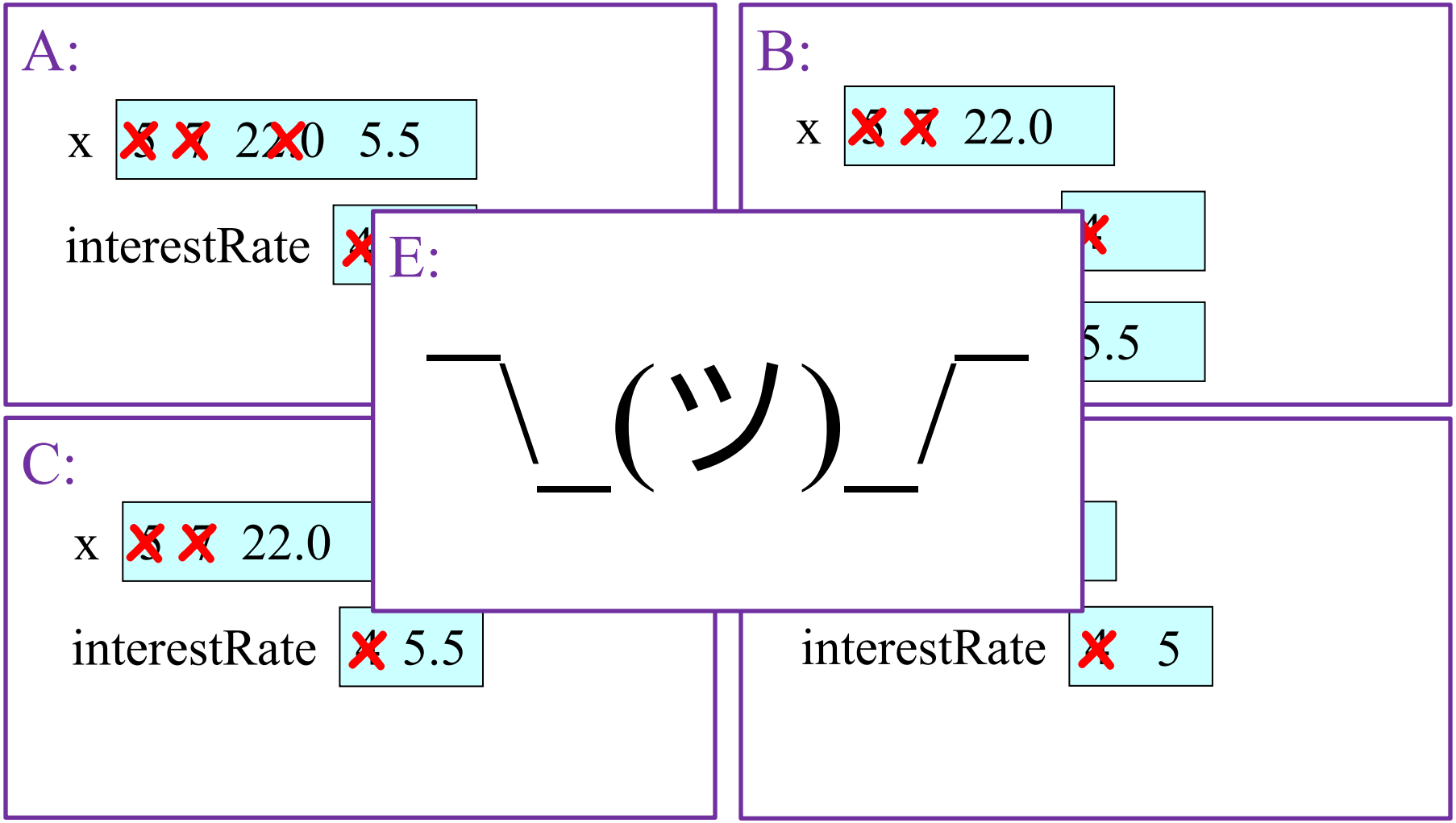


B:

C:

D:

Which One is Closest to Your Answer?



Which One is Closest to Your Answer?

```
interestRate = x/interestRate
```

B:

x 22.0

interestRate

interestRate 5.5

C:

x 22.0

interestRate 5.5

D:

x 22.0

interestRate 5

Exercise: Understanding Assignment

- You now have this:

x ~~x~~ ~~x~~ 22.0 interestRate ~~x~~ 5.5

- 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.

Which One is Closest to Your Answer?

A:

x 22.0

interestRate 27.5

B:

x 22.0

interestRate 5.5

intrestRate 27.5

C:

x 22.0 27.5

interestRate 5.5

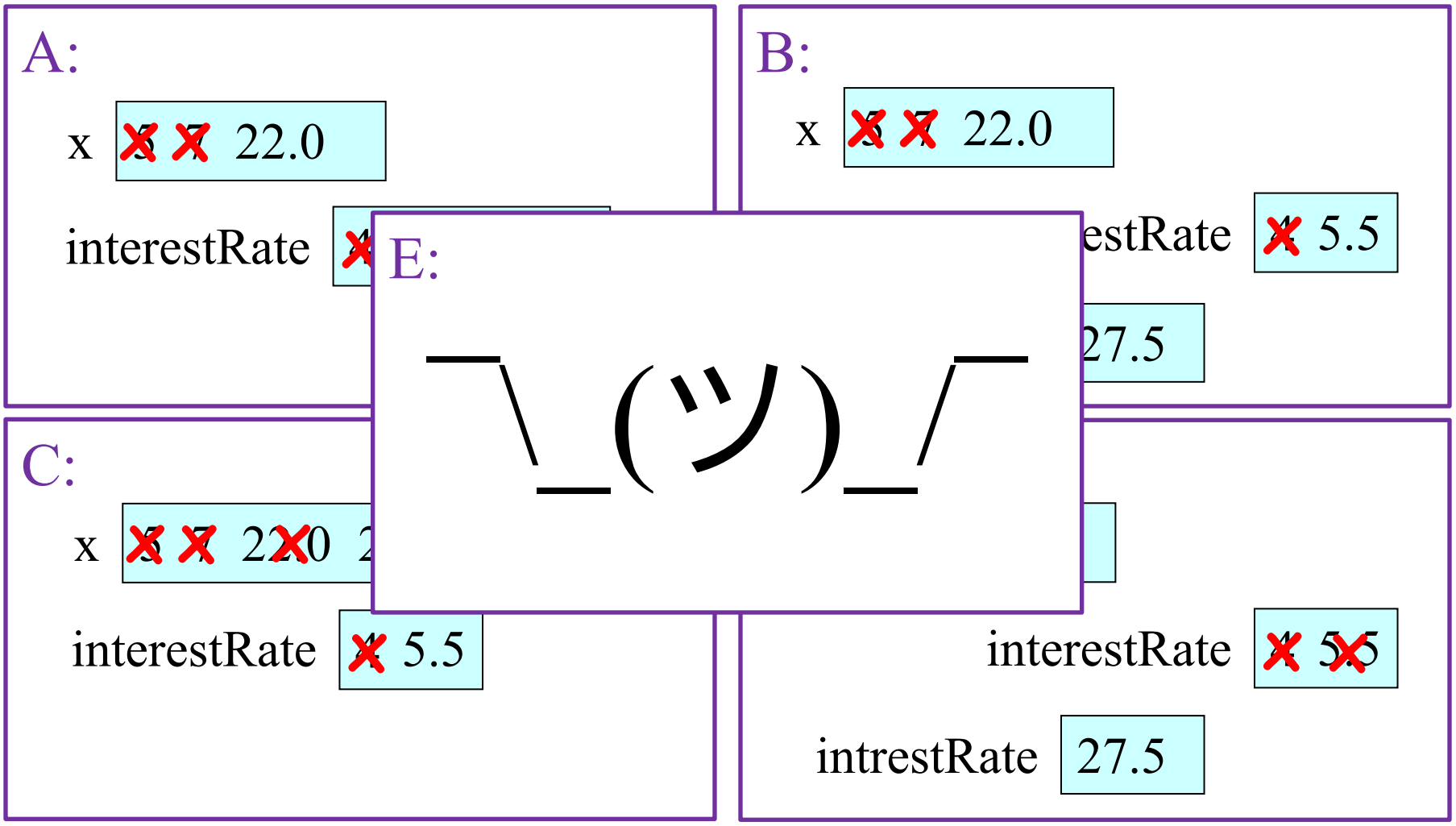
D:

x 22.0

interestRate 5.5

intrestRate 27.5

Which One is Closest to Your Answer?



Which One is Closest to Your Answer?

A:

x 22.0

interestRate 27.5

B:

x 22.0

interestRate 5.5

intrestRate 27.5

```
intrestRate =  x + interestRate
```

e

Which One is Closest to Your Answer?

A:

x 22.0

interestRate 27.5

B:

x 22.0

interestRate 5.5

intrestRate 27.5

```
intrestRate =  x + interestRate
```

e

Spelling mistakes in Python are bad!!

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 = 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
  - This is true in Java , C, C++, etc.

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**