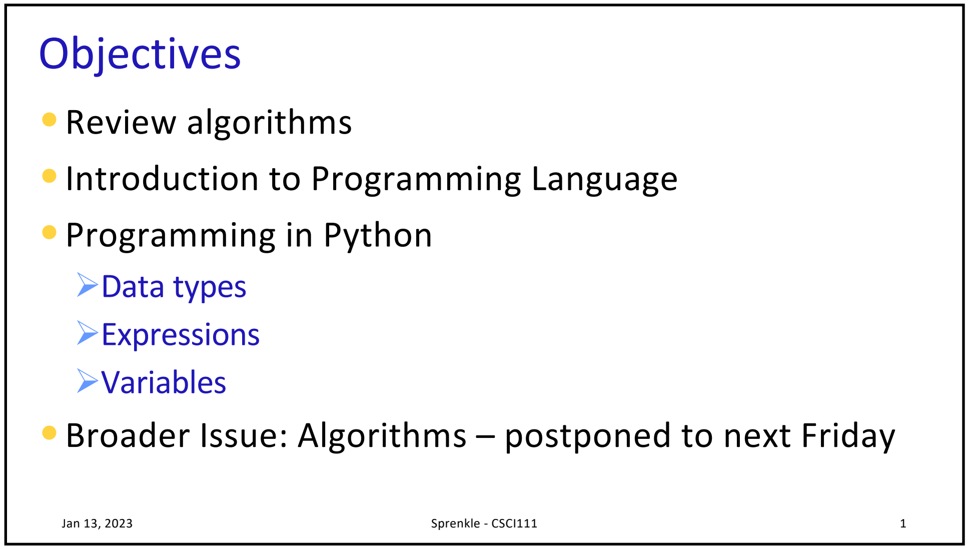
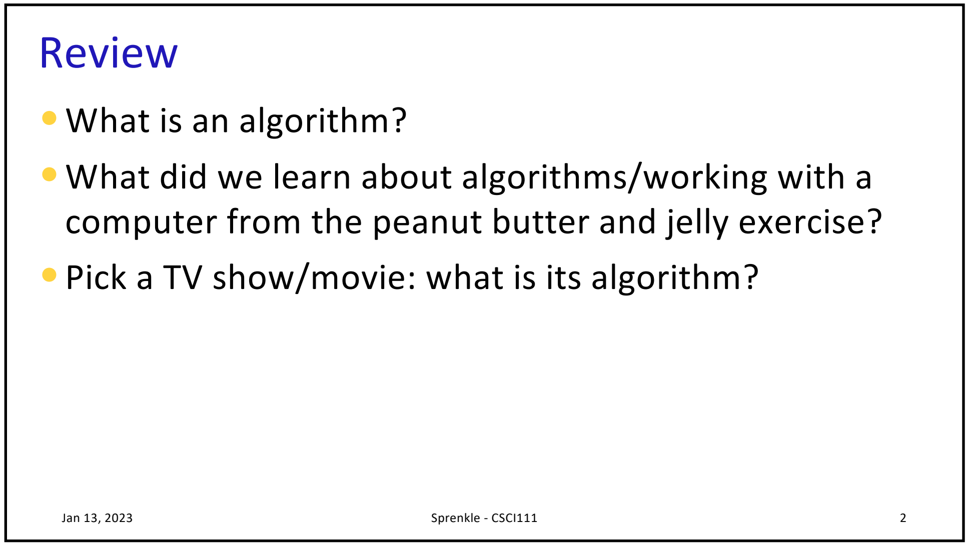


Objectives

- Review algorithms
- Introduction to Programming Language
- Programming in Python
  - Data types
  - Expressions
  - Variables
- Broader Issue: Algorithms – postponed to next Friday



Review

- What is an algorithm?
- What did we learn about algorithms/working with a computer from the peanut butter and jelly exercise?
- Pick a TV show/movie: what is its algorithm?

“Really?” with Professor Sprenkle

- In *TV Guide*, showrunners of *Once Upon a Time* were asked, “Give us an algorithm for your show.”

“Really?” with Professor Sprenkle

- In *TV Guide*, showrunners of *Once Upon a Time* were asked, “Give us an algorithm for your show.”
  - ➢Example (for first season): 1 part Snow White + 1 part *Lost* + .5 *Alias*
- They said, “We don’t understand math. That’s why we became writers.”

Review: Discussion of PB&J

- The computer: a blessing and a curse
  - ➢ Recognize and meet the challenge!
- Be unambiguous, descriptive
  - ➢ Must be clear for the computer to understand
  - ➢ “Do what I **meant**! Not what I said!”
    - Motivates programming languages
- Creating/Implementing an algorithm
  - ➢ Break down pieces
  - ➢ Try it out
  - ➢ Revise

Review: Discussion of PB&J

- Steps need to be done in a particular order
- Be prepared for special cases
  - ➢ Any other special cases we didn’t discuss?
- Aren’t necessarily spares in real life
  - ➢ Need to write correct algorithms!
- Reusing similar techniques
  - ➢ Do the same thing with a little twist
- Looping
  - ➢ For repeating the same action

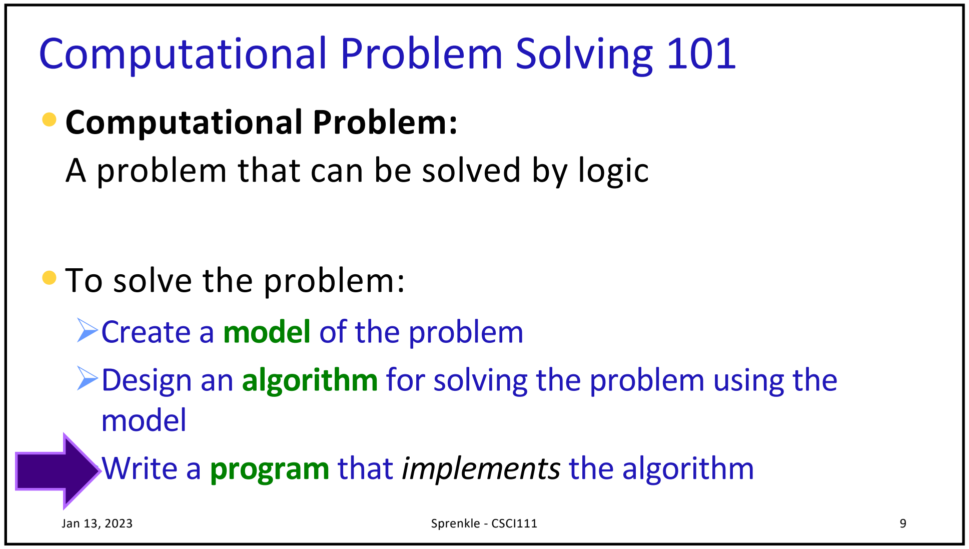
Other Lessons To Remember

- A cowboy’s wisdom: Good judgment comes from experience
  - How can you get experience?
  - Bad judgment works every time
- Program errors can have **bad** effects
  - Prevent the bad effects (that’s the thinking part)--especially before you turn in your assignment!

Parts of an Algorithm

- Input, Output
- Primitive operations
  - What data you have, what you can do to the data
- Naming
  - Identify things we’re using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques

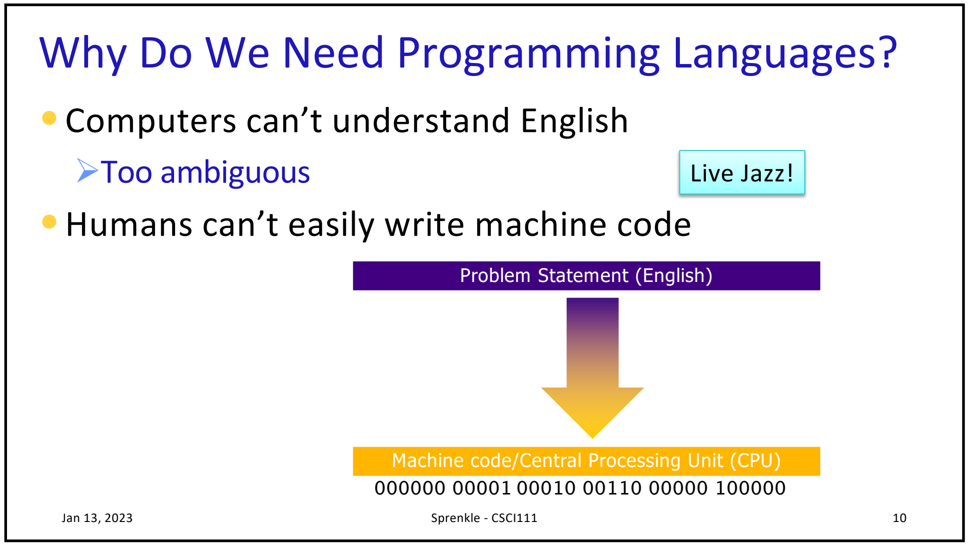
An overview for the semester!



Computational Problem Solving 101

- **Computational Problem:**
  A problem that can be solved by logic

- To solve the problem:
  - Create a **model** of the problem
  - Design an **algorithm** for solving the problem using the model
  - Write a **program** that *implements* the algorithm



Why Do We Need Programming Languages?

- Computers can’t understand English
  - Too ambiguous

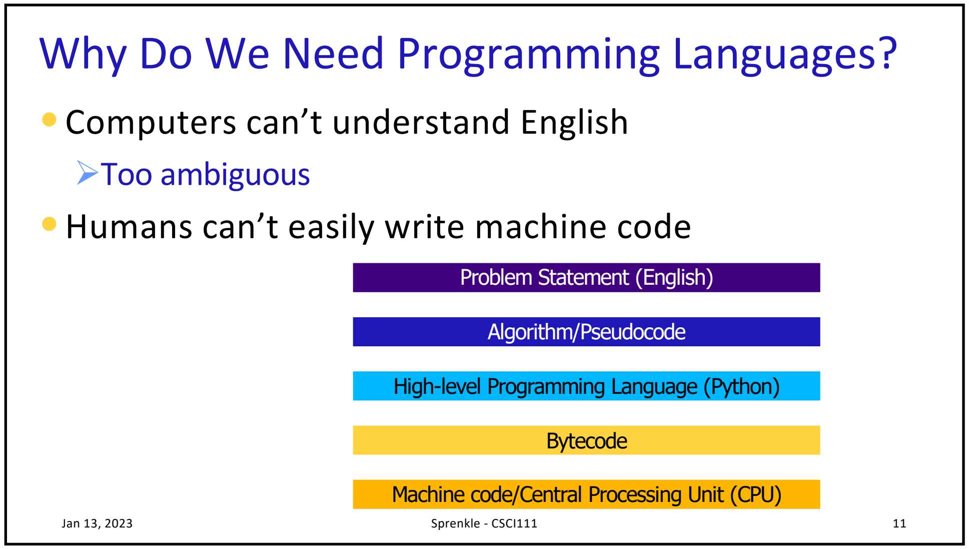
Live Jazz!

- Humans can’t easily write machine code

Problem Statement (English)

Machine code/Central Processing Unit (CPU)

000000 00001 00010 00110 00000 100000



Why Do We Need Programming Languages?

- Computers can’t understand English
  - ➢Too ambiguous
- Humans can’t easily write machine code

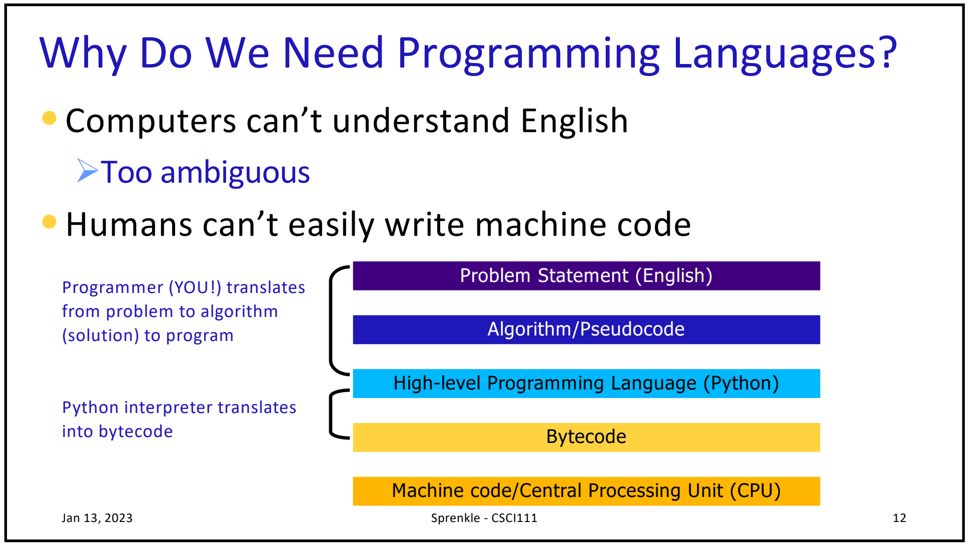
Problem Statement (English)

Algorithm/Pseudocode

High-level Programming Language (Python)

Bytecode

Machine code/Central Processing Unit (CPU)



Why Do We Need Programming Languages?

- Computers can’t understand English
  - ➢Too ambiguous
- Humans can’t easily write machine code

Programmer (YOU!) translates from problem to algorithm (solution) to program

Python interpreter translates into bytecode

Problem Statement (English)

Algorithm/Pseudocode

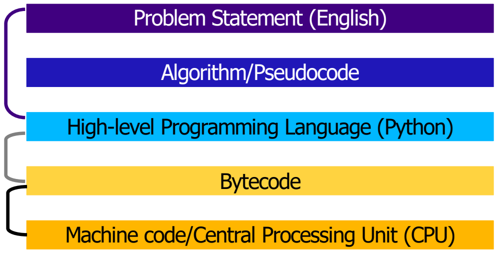
High-level Programming Language (Python)

Bytecode

Machine code/Central Processing Unit (CPU)

Why Do We Need Programming Languages?

- Computers can’t understand English
  - Too ambiguous
- Humans can’t easily write machine code



Programmer (YOU!) translates from problem to algorithm (solution) to program

Python interpreter translates into bytecode

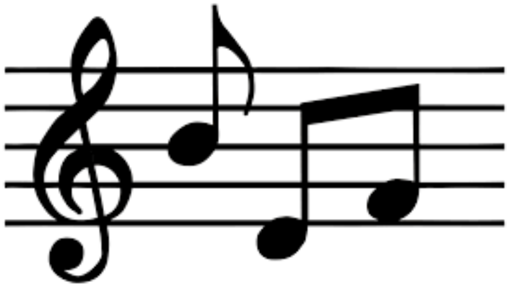
Python interpreter executes the bytecode in a “virtual machine”

Problem Statement (English)
Algorithm/Pseudocode
High-level Programming Language (Python)
Bytecode
Machine code/Central Processing Unit (CPU)

Programming Languages

- Programming language:
  - Specific rules for what is and isn’t allowed
  - Must be exact
  - Computer carries out commands as they are given
- **Syntax**: the symbols given
- **Semantics**: what it means
- Example:
  - III * IV means 3 × 4 which evaluates to 12
  - `cp src dest` means copy the file named `src` to `dest`
- Programming languages are unambiguous

Another Syntax and Semantics Example



What is the *syntax*? What is the *semantics*?

Python Is …

- A ***programming language***
  - The *most* popular programming language, according to the Tiobe index

    `http://www.tiobe.com/tiobe-index/`

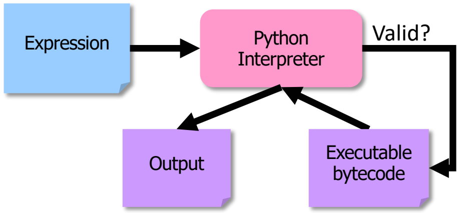
- An ***interpreter*** (which is a *program*) that understands and executes Python code

Python

- A common *interpreted* programming language
  - Runs on many operating systems
- First released by Guido van Rossum in 1991
- Named after *Monty Python's Flying Circus*
- Minimalist syntax, emphasizes readability
- Flexible, fast, useful language
- Used by scientists, engineers, systems programmers

Python Interpreter

1. Validates Python programming language expression(s)
   - Enforces Python **syntax**
   - Reports **syntax** errors
2. Executes expression(s)
   - Runtime errors (e.g., divide by 0)
   - **Semantic** errors (not what you *meant*)



Could be errors

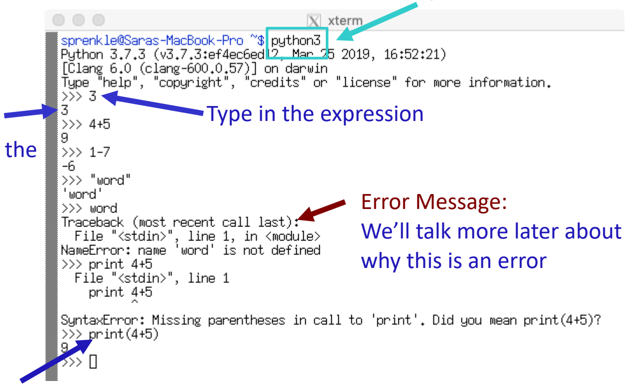
Two Modes to Execute Python Code

- **Interactive**: using the interpreter
  - Try out Python expressions

- **Batch**: execute *scripts* (i.e., files containing Python code)
  - What we'll usually write

Interactive Mode

Run by typing "python3" in terminal



```
sprenkle@Saras-MacBook-Pro ~$ python3
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 16:52:21)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 3
3
>>> 4+5
9
>>> 1-7
-6
>>> "word"
'word'
>>> word
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'word' is not defined
>>> print 4+5
  File "<stdin>", line 1
    print 4+5
          ^

SyntaxError: Missing parentheses in call to 'print'. Did you mean print(4+5)?
>>> print(4+5)
9
>>> 
```

Python displays the result

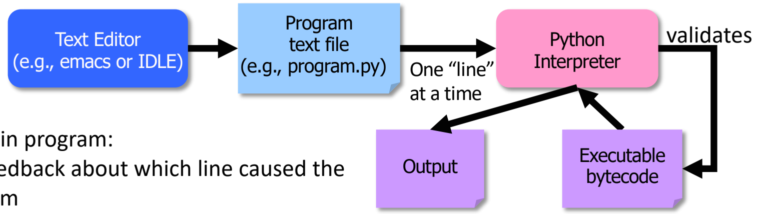
Type in the expression

Error Message:
We'll talk more later about why this is an error

`print`: Special function to display output

Batch Mode

1. Programmer types a program/script into a **text editor**
2. An interpreter turns each expression into bytecode and then executes each expression



Text Editor (e.g., emacs or IDLE)

Program text file (e.g., program.py)

One “line” at a time

Python Interpreter

validates

Executable bytecode

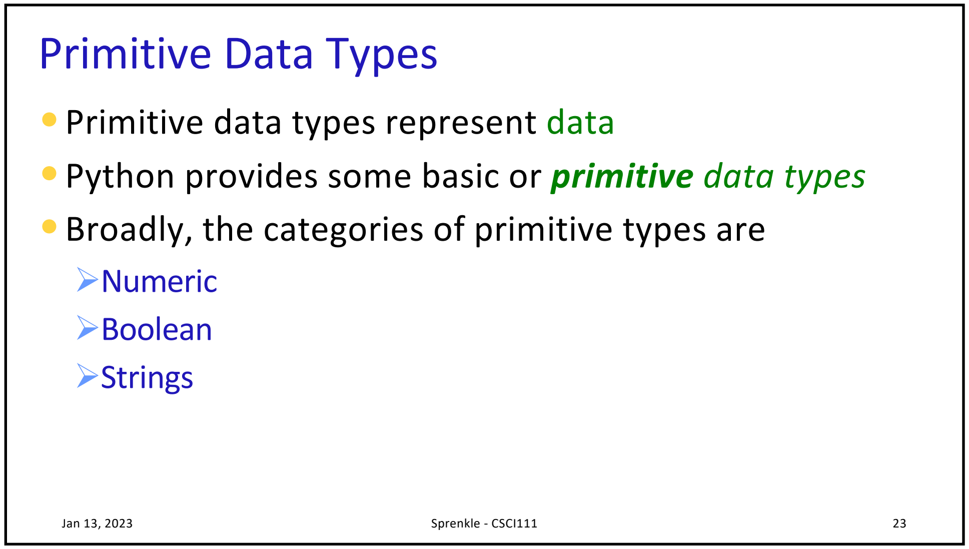
Output

If errors in program:

- Get feedback about which line caused the problem
- Interpreter stops validating/executing lines

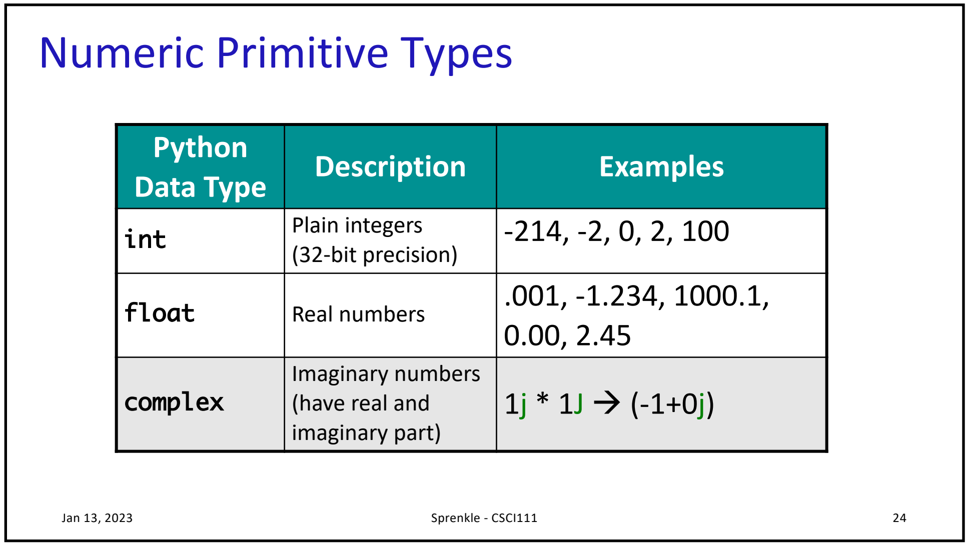
Parts of an Algorithm

- Input, **Output**
- Primitive operations
  - What data you have, what you can do to the data
- Naming
  - Identify things we’re using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques



Primitive Data Types

- Primitive data types represent data
- Python provides some basic or ***primitive*** *data types*
- Broadly, the categories of primitive types are
  - Numeric
  - Boolean
  - Strings



Numeric Primitive Types

Python Data Type	Description	Examples
`int`	Plain integers (32-bit precision)	-214, -2, 0, 2, 100
`float`	Real numbers	.001, -1.234, 1000.1, 0.00, 2.45
`complex`	Imaginary numbers (have real and imaginary part)	1j * 1J → (-1+0j)

How big (or small or precise) can we get?

- Computer cannot represent all values
- Problem: Computer has a **finite** capacity
  - The computer only has so much memory that it can devote to one value.
  - Eventually, reach a cutoff
    - Limits size of value
    - Limits precision of value

PI has more decimals, but we're out of space!

0	0	0	0	0	3	.	1	4	1	5	9	2	6	5

Example: in Python interpreter, .1 + .1 + .1 yields 0.30000000000000004.
* In reality, computers represent data in binary.

Strings: `str`

- Indicated by double quotes " " or single quotes ' '
- Treat what is in the " " or ' ' literally
  - Known as ***string literals***
- Examples:
  - "Hello, world!"
  - 'c'
  - "That is Buddy's dog."

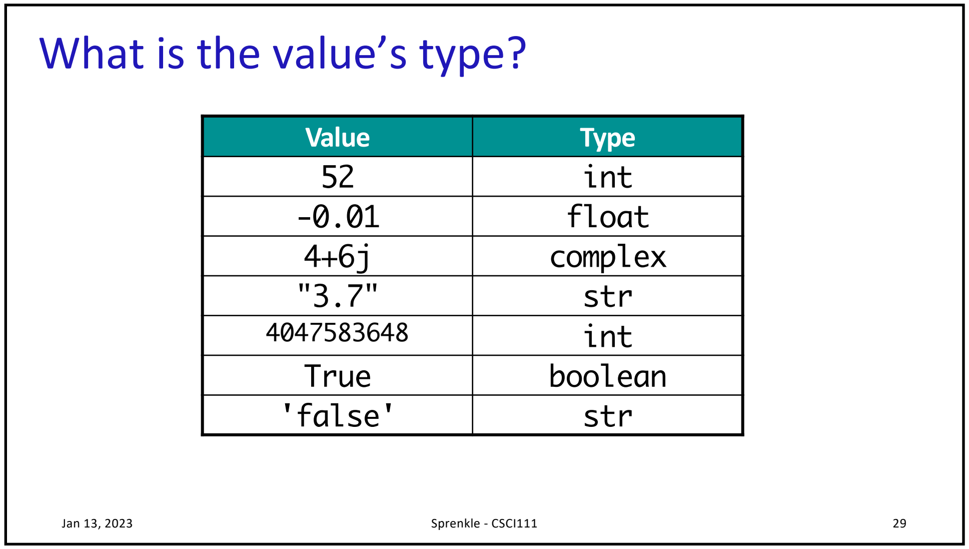
Single quote must be inside double quotes*
*Exception later

Booleans: `bool`

- 2 values
  - `True`
  - `False`
- Much more on these later…

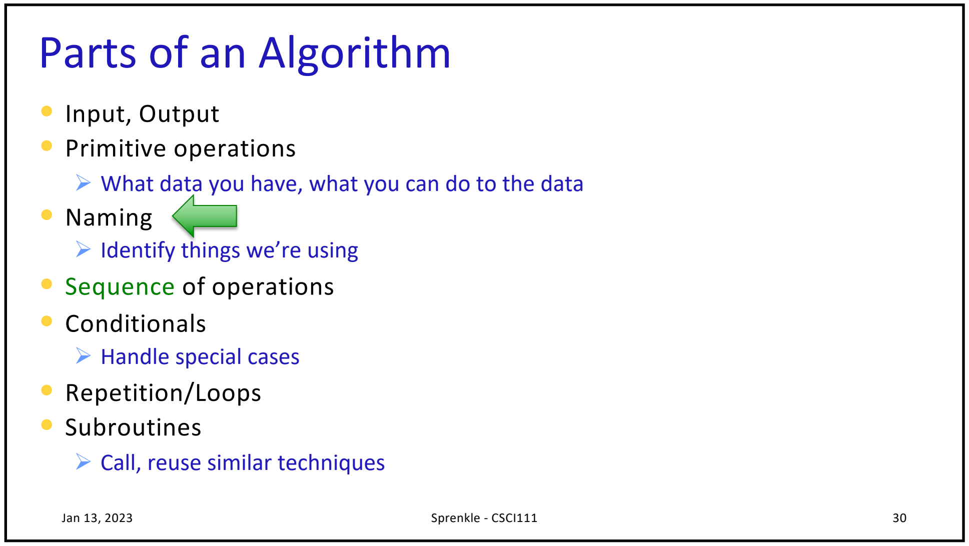
What is the value’s type?

Value	Type
`52`	
`-0.01`	
`4+6j`	
`"3.7"`	
4047583648	
`True`	
`'false'`	



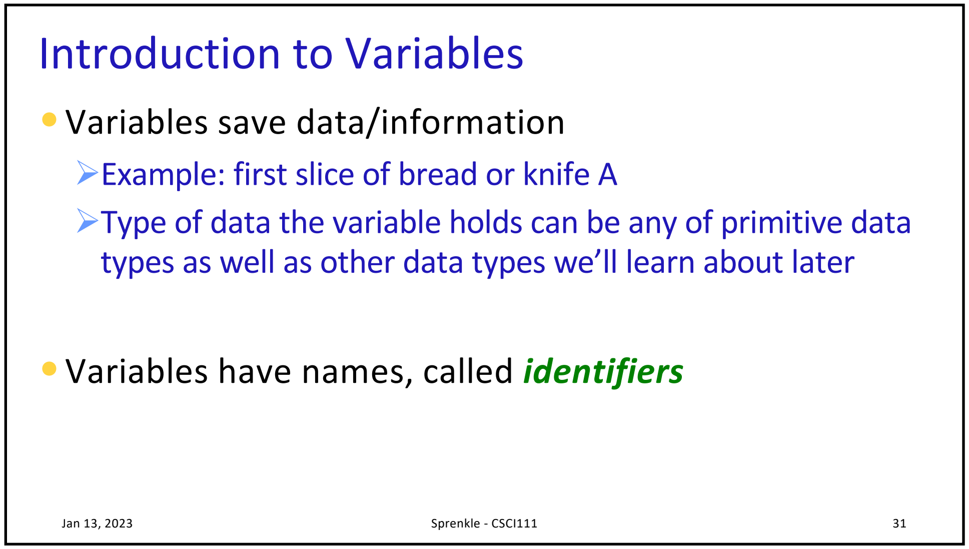
What is the value's type?

Value	Type
52	int
-0.01	float
4+6j	complex
"3.7"	str
4047583648	int
True	boolean
'false'	str



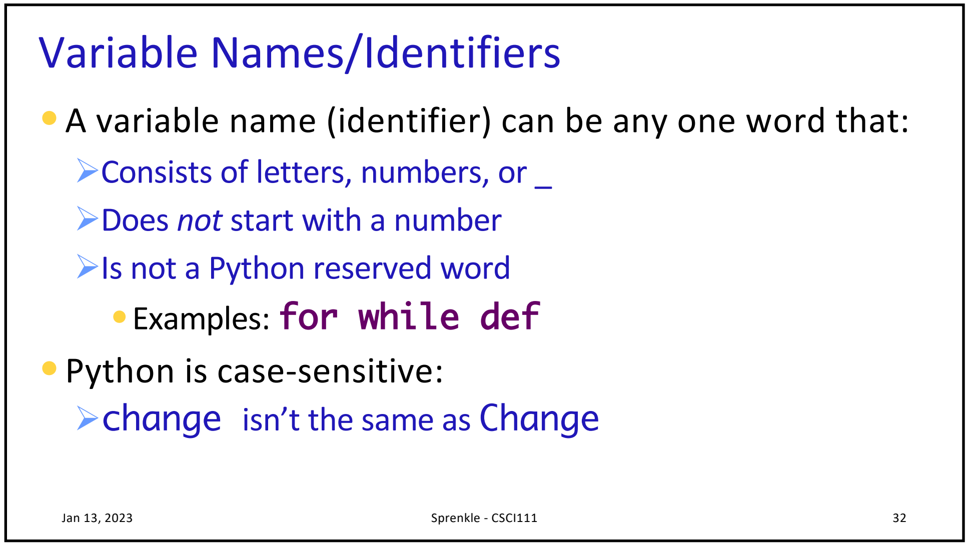
Parts of an Algorithm

- Input, Output
- Primitive operations
  - What data you have, what you can do to the data
- Naming
  - Identify things we're using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques



Introduction to Variables

- Variables save data/information
  - ➢Example: first slice of bread or knife A
  - ➢Type of data the variable holds can be any of primitive data types as well as other data types we'll learn about later
- Variables have names, called ***identifiers***



Variable Names/Identifiers

- A variable name (identifier) can be any one word that:
  - ➢Consists of letters, numbers, or _
  - ➢Does *not* start with a number
  - ➢Is not a Python reserved word
    - Examples: **`for while def`**
- Python is case-sensitive:
  - ➢`change` isn't the same as `Change`

Variable Name Conventions

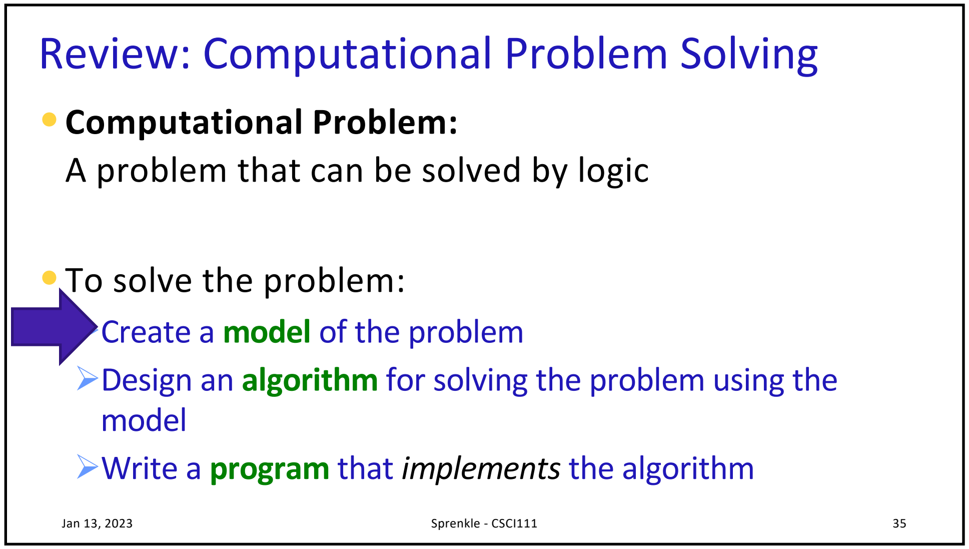
- **Variables** start with a lowercase letter
- Convention: **Constants** (values that won't change) are all capitals
  - (more on this later…)
- Example: Variable for the current year
  - `currentYear`
  - `current_year`
  - `CURRENT_YEAR`
  - ~~`currentyear`~~ Harder to read
  - ~~`current year`~~ No spaces allowed

Naming doesn't matter to computer, matters to *humans*

Importance of Variable Naming

- Helps you *remember* what the variable represents
- Easier for others to *understand* your program
- Examples:

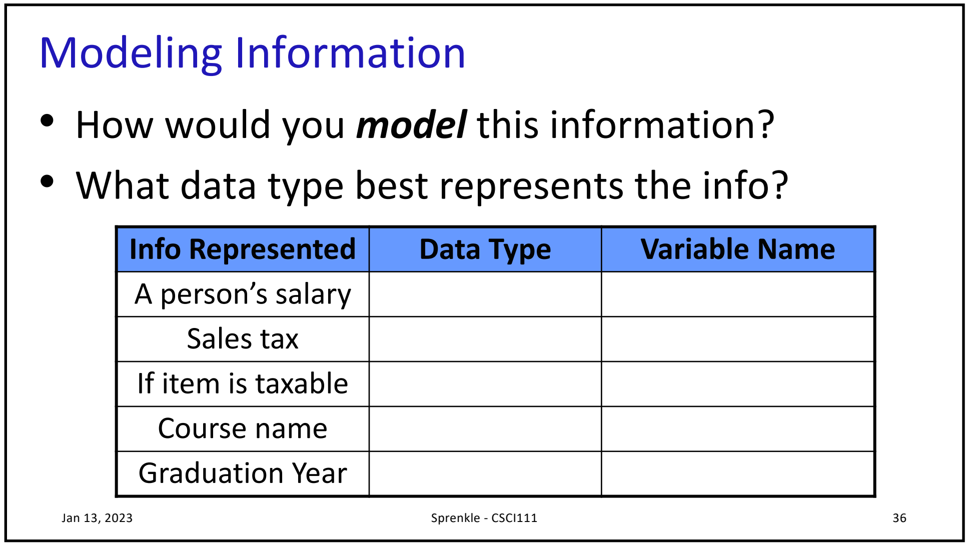
Info Represented	Good Variable Name
A person's first name	`firstName, first_name`
Radius of a circle	`radius`
If someone is employed or not	`isEmployed`



Review: Computational Problem Solving

- **Computational Problem:**
  A problem that can be solved by logic

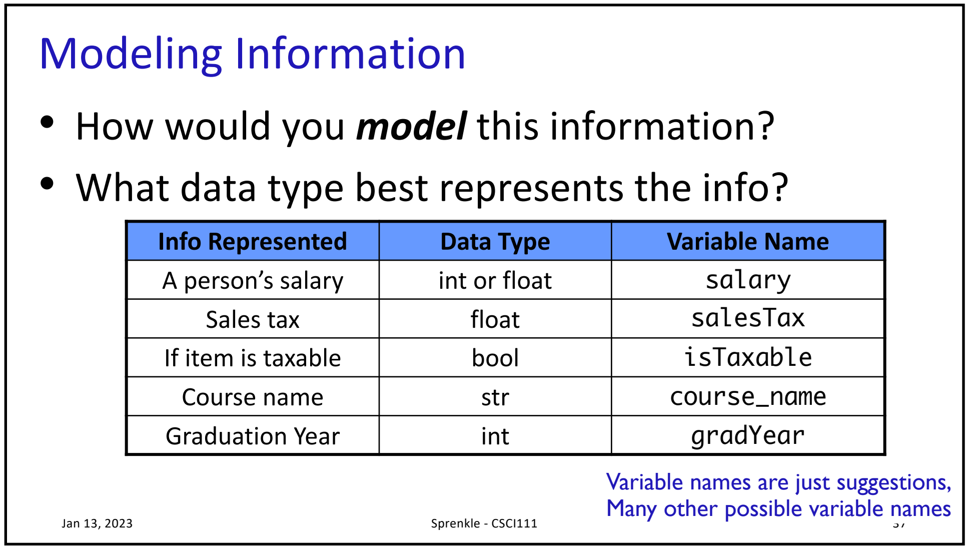
- To solve the problem:
  - Create a **model** of the problem
  - Design an **algorithm** for solving the problem using the model
  - Write a **program** that *implements* the algorithm



Modeling Information

- How would you ***model*** this information?
- What data type best represents the info?

Info Represented	Data Type	Variable Name
A person’s salary		
Sales tax		
If item is taxable		
Course name		
Graduation Year		

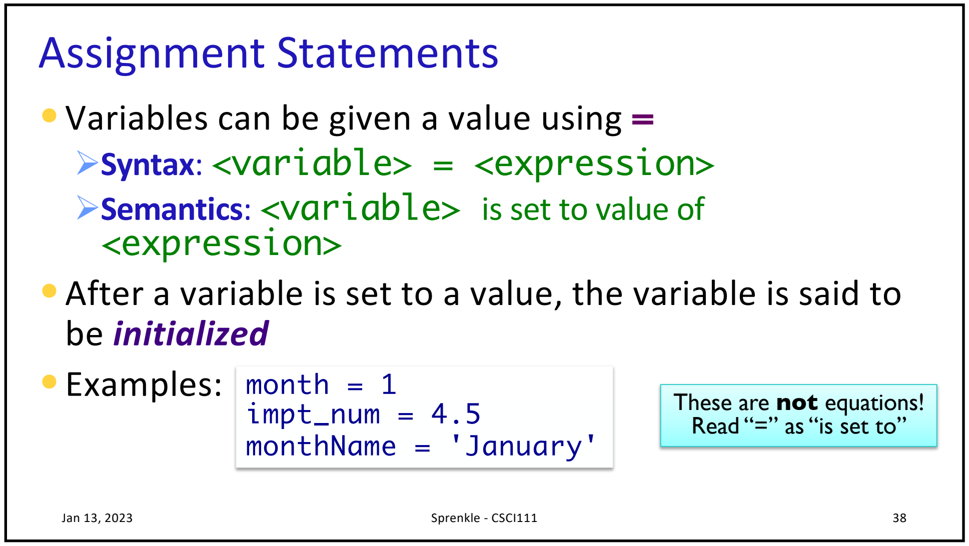


Modeling Information

- How would you ***model*** this information?
- What data type best represents the info?

Info Represented	Data Type	Variable Name
A person's salary	int or float	`salary`
Sales tax	float	`salesTax`
If item is taxable	bool	`isTaxable`
Course name	str	`course_name`
Graduation Year	int	`gradYear`

Variable names are just suggestions,
Many other possible variable names



Assignment Statements

- Variables can be given a value using **=**
  - **Syntax**: `<variable> = <expression>`
  - **Semantics**: `<variable>` is set to value of `<expression>`
- After a variable is set to a value, the variable is said to be ***initialized***
- Examples:

```
month = 1
impt_num = 4.5
monthName = 'January'
```

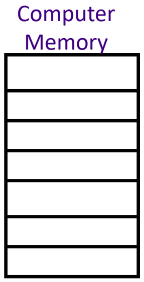
These are **not** equations!
Read "=" as "is set to"

Variables: The Rules

- Only the variable(s) to **left** of the = in the current statement change
  - We'll only have one variable on the left
- Order of operations
  1. Evaluate the expression on the right
  2. Assign the variable on the left to the evaluated expression
- **Initialize** a variable **before** using it on the right-hand side (rhs) of a statement

Assignment Statements

```
x = 5
y = x
```



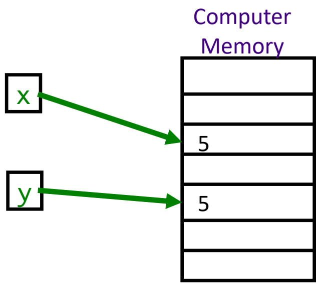
- Statements execute in order, from top to bottom
- Value of **x** does not change because of second assignment statement

https://pythontutor.com/visualize.html

Assignment Statements

```
x = 5
y = x
```

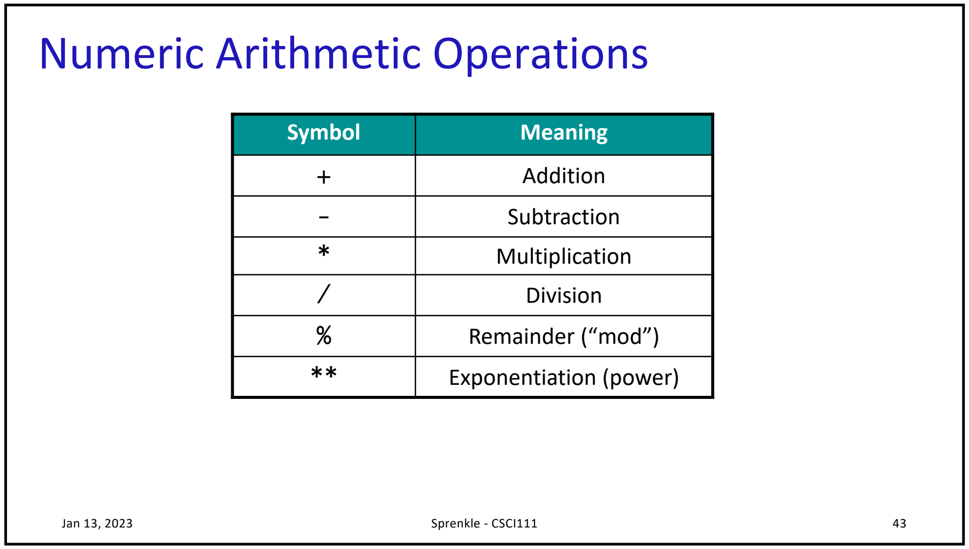
Does a "lookup" in memory to find value of x



- Statements execute in order, from top to bottom
- Value of **x** does not change because of second assignment statement

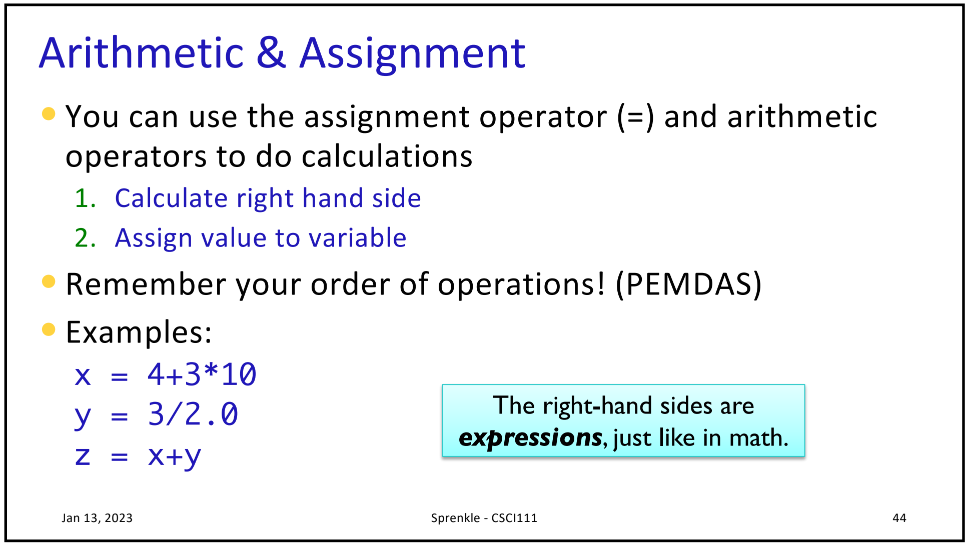
Literals

- Pieces of data that are not variables are called ***literals***
  - We've been using these a lot
- Examples:
  - 4
  - `3.2`
  - `'q'`
  - `"books"`



Numeric Arithmetic Operations

Symbol	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder (“mod”)
**	Exponentiation (power)

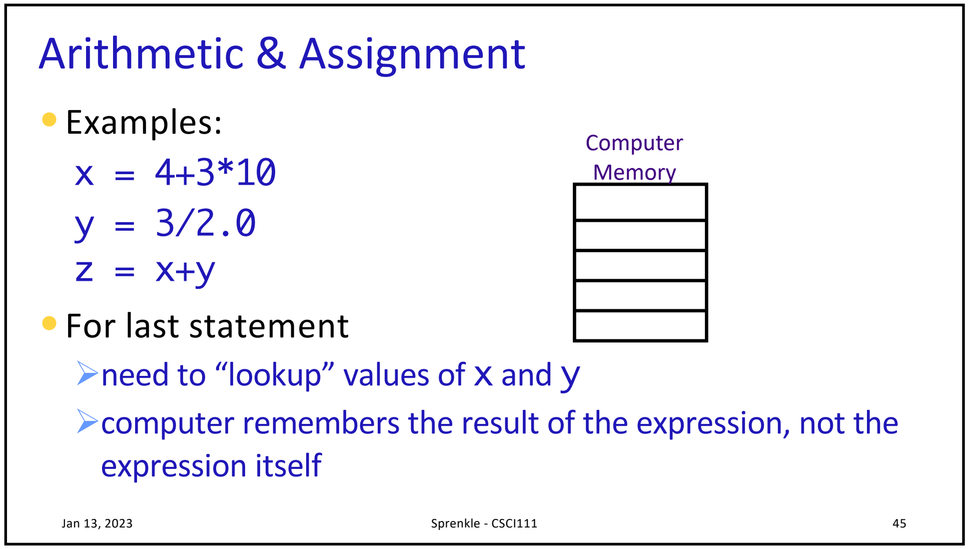


Arithmetic & Assignment

- You can use the assignment operator (=) and arithmetic operators to do calculations
  1. Calculate right hand side
  2. Assign value to variable
- Remember your order of operations! (PEMDAS)
- Examples:

```
x = 4+3*10
y = 3/2.0
z = x+y
```

The right-hand sides are ***expressions***, just like in math.



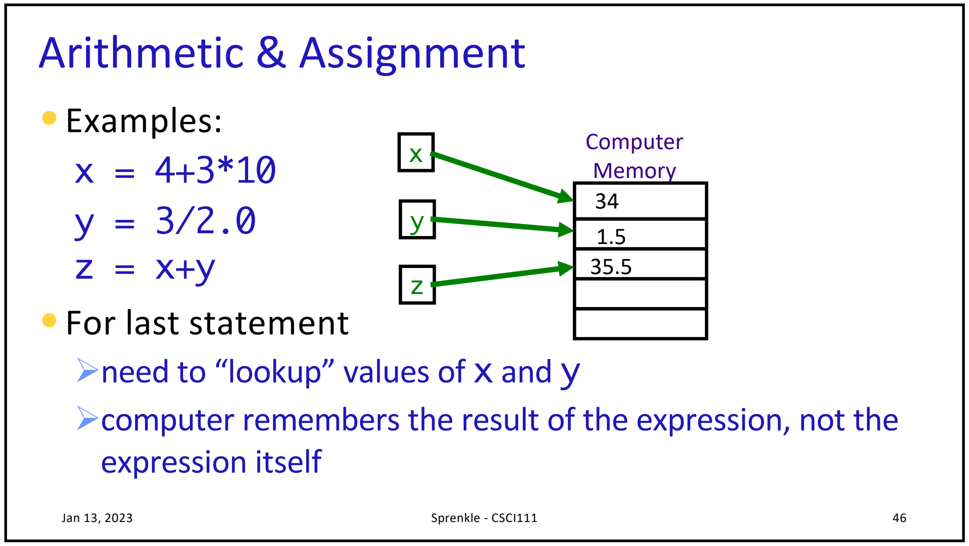
Arithmetic & Assignment

- Examples:

```
x = 4+3*10
y = 3/2.0
z = x+y
```

Computer
Memory

- For last statement
  - need to “lookup” values of x and y
  - computer remembers the result of the expression, not the expression itself



Arithmetic & Assignment

- Examples:

```
x = 4+3*10
y = 3/2.0
z = x+y
```

x
y
z
Computer
Memory
34
1.5
35.5

- For last statement
  - need to “lookup” values of x and y
  - computer remembers the result of the expression, not the expression itself

NOT Math Class

- Need to write out all operations explicitly
  - In math class, a (b+1) meant `a*(b+1)`

Write this way in Python

What are the values?

- After executing the following statements, what are the values of each variable?
  - `r = 5`
  - `s = -1 + r`
  - `t = r + s`
  - `s = 2`
  - `r = -7`

How can we confirm that we’re right?

Parts of an Algorithm

- ➔Input, **Output**
- Primitive operations
  - What data you have, what you can do to the data
- Naming
  - Identify things we’re using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques

Printing Output

- `print` is a ***function***
  - Displays the result of expression(s) to the terminal
  - Automatically adds a ‘\n’ (carriage return) after it’s printed
    - Relevant when have multiple print statements
- `print("Hello, class")`

string literal

Syntax: a pair of double quotes
Semantics: represents text

Printing Multiple Things

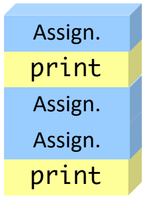
- `print` is a a ***function***
- To display multiple things on the same line, separate them with commas

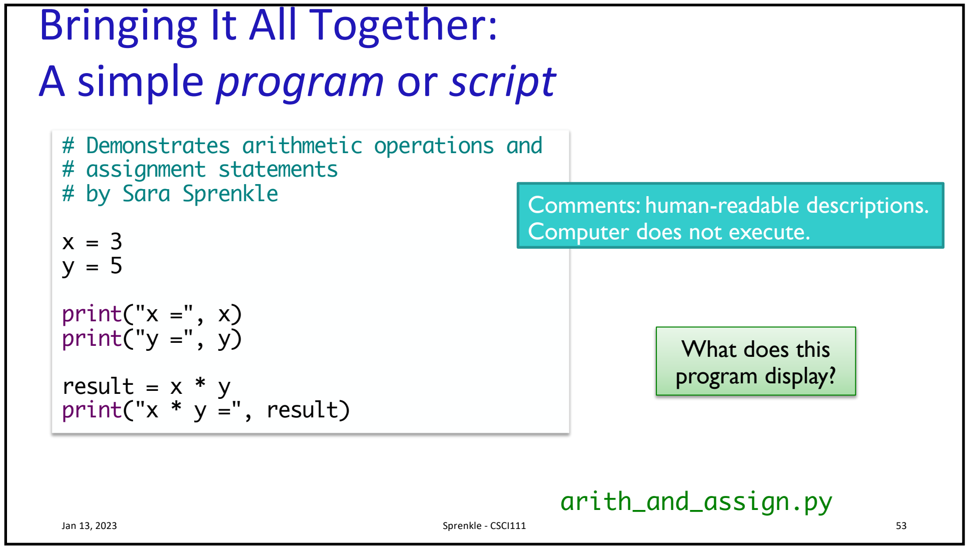
```
➢print("Hello,", "class")
➢print("x =", 5)
➢print(x*y, "is the magic number")
➢print(r, s, t)
```

Syntax: ,
Semantics: display this too, separated by a space in the display

Programming Building Blocks

- Each type of statement is a building block
  - Initialization/Assignment Assign.
    - So far: Arithmetic
  - Print print
- We can combine them to create more complex programs
  - Solutions to problems





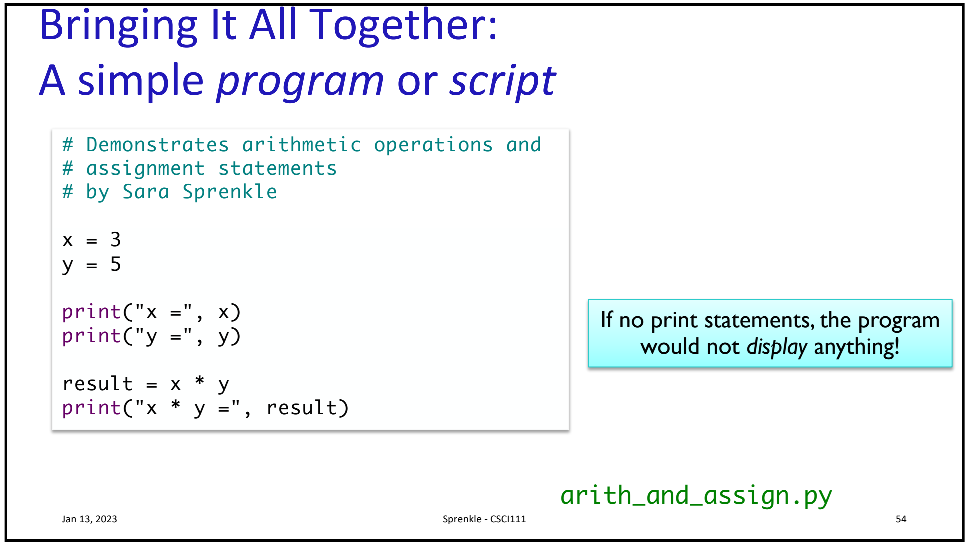
Bringing It All Together:
A simple *program* or *script*

```
# Demonstrates arithmetic operations and
# assignment statements
# by Sara Sprenkle

x = 3
y = 5

print("x =", x)
print("y =", y)

result = x * y
print("x * y =", result)
```

Comments: human-readable descriptions. Computer does not execute.

What does this program display?

arith_and_assign.py



Bringing It All Together:
A simple *program* or *script*

```
# Demonstrates arithmetic operations and
# assignment statements
# by Sara Sprenkle

x = 3
y = 5

print("x =", x)
print("y =", y)

result = x * y
print("x * y =", result)
```

If no print statements, the program would not *display* anything!

arith_and_assign.py

Bringing It All Together: A simple *program* or *script*

```
# Demonstrates arithmetic operations and
# assignment statements
# by Sara Sprenkle

x = 3
y = 5

print("x =", x)
print("y =", y)

# alternative to the previous program
print("x * y =", x * y)
```

Comments: human-readable descriptions. Computer does not execute.

This `print` statement is slightly more complicated than previous example. Goal: keep each statement simple so that it's easier to find errors.

`arith_and_assign2.py`

Looking Ahead

- Textbook Pre Lab 1 assignment due before lab on Tuesday
    - Covers some things we haven't yet covered in class; we'll review on Tuesday
- Extra Credit Opportunity:
    - Read an article that relates to CS
    - Summarize it on the discussions under "Extra Credit"
        - 5 pts extra credit added to lab grade