Objectives

- More Assignments and Arithmetic
- Software development practices
  - Testing
  - Debugging
  - Iteration

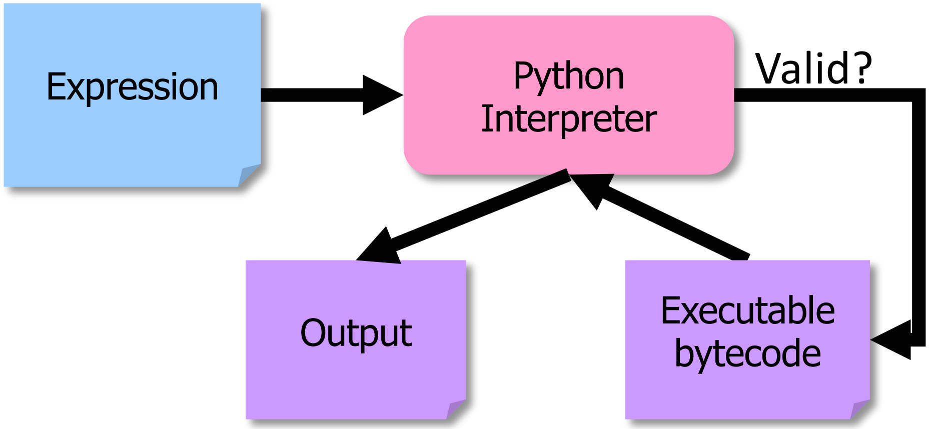
Review

Note: using slightly different terminology
Goal: comfort with terminology, synonyms

1. What is Python? (two things)
2. What are the two modes for running Python?
3. How can we store information?
   - What is the syntax to do that?
4. What are the rules and conventions for variable names?
   - What is another term for "variable names"?
   - Describe characteristics of *good* variable names
5. What are the primitive types of information in Python?
6. What are the arithmetic operators? Describe their syntax and semantics.

Review: 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

Recap: Programming Fundamentals

- Most important data types (for us, for now): `int, float, str, bool`
  - Use these types to represent various information
- Variables have identifiers, (implicit) types
  - Should have "good" names
  - Names: start with lowercase letter; can have numbers, underscores
- Assignments
  - `x = y` means "x set to value y" or "x is assigned value of y"
  - Only variable on LHS of statement changes

Review: Numeric Arithmetic Operations

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

Remember PEMDAS

Review: 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.

Assignment statements

- Assignment statements are NOT math equations!
  - Valid expression: `count = count + 1`

- These are commands!

```
x = 2
y = x
x = x + 3
```

After these 3 statements execute, what are the values of x, y?

What are the values?

- After executing the following statements, what are the values of each variable?

```
1. a = 5
2. y = a + -1 * a
3. z = a + y / 2
4. a = a + 3
5. y = (7+x)*z
6. x = z*2
```

What are the values?

- After executing the following statements, what are the values of each variable?

```
a = 5
y = a + -1 * a
z = a + y / 2
a = a + 3
y = (7+x)*z
x = z*2
```

Runtime error: x doesn’t have a value yet!

- We say “x was not initialized”
- Can’t use a variable on RHS until seen on LHS!*

Printing Output

- `print` is a special command or 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

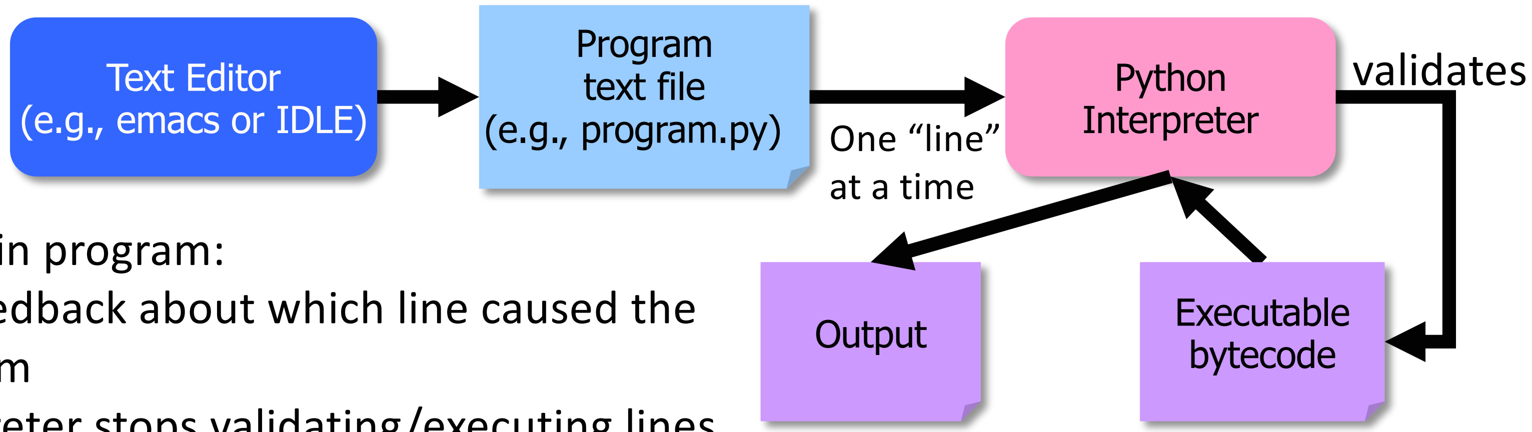
Syntax: a pair of double quotes
Semantics: represents text

Printing Multiple Things

- `print` is a special command or a ***function***
  - **Syntax**: `print(arg1, arg2, arg3, … )`
  - **Semantics**: display the arguments, in order separated by a space in the display; ends with a "\n"
- 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)`

Review: Batch Mode

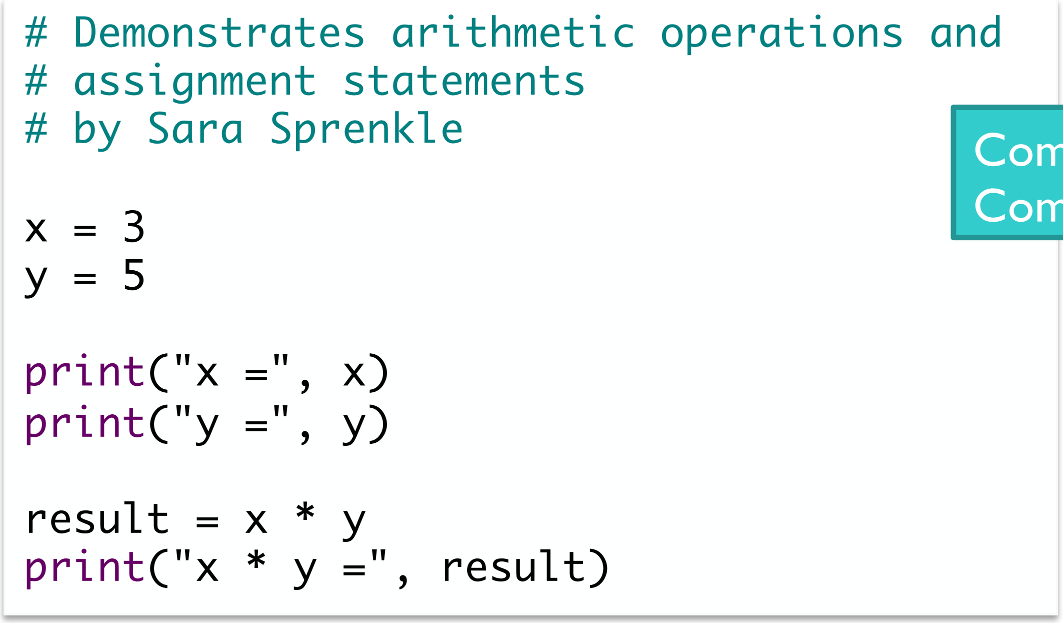
1. Programmer types a program/script into a **text editor**
2. An interpreter turns each expression into bytecode and then executes each expression



If errors in program:

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

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

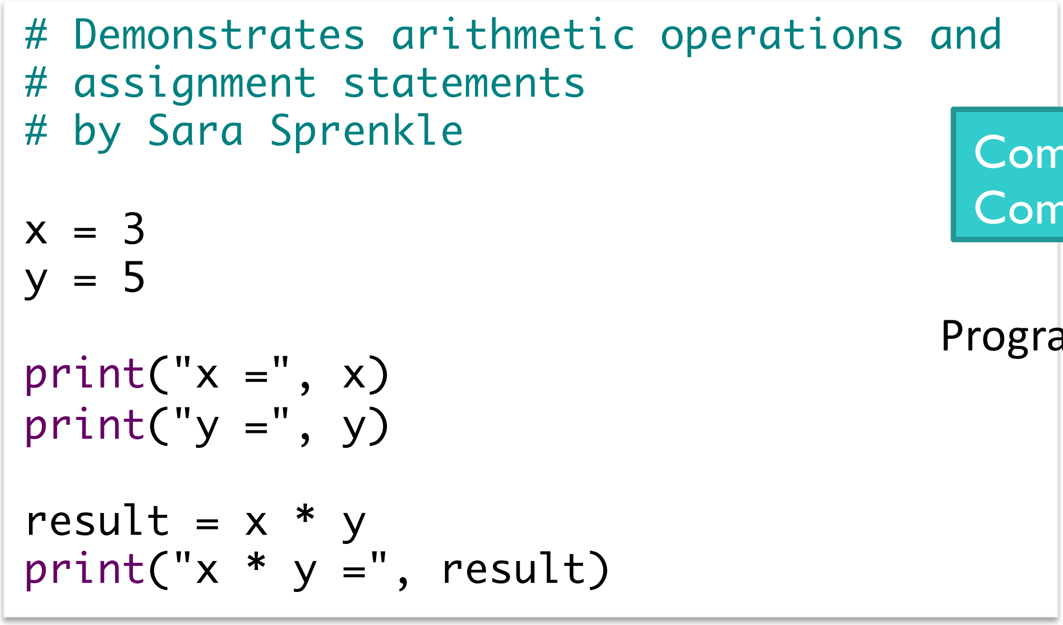


```
# Demonstrates arithmetic operations and
# assignment statements
# by Sara Sprenkle

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.

arith_and_assign.py

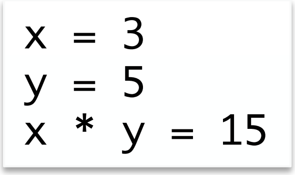
Bringing It All Together:
A simple *program* or *script*



```
# 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)
```

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

Program outputs/displays:

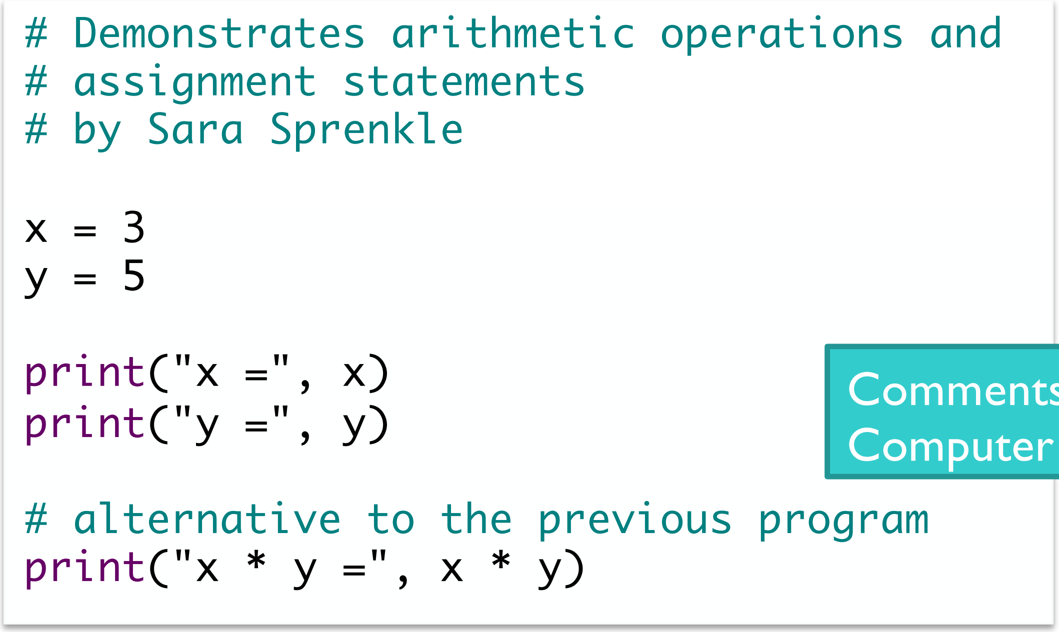


```
x = 3
y = 5
x * y = 15
```

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.

arith_and_assign2.py

Equivalent Output to Previous Example

```
# Demonstrates arithmetic operations and
# assignment statements
# by Sara Sprenkle
x = 3
y = 5

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

# alternative to the previ      ogram
print("x * y =", x * y)
```

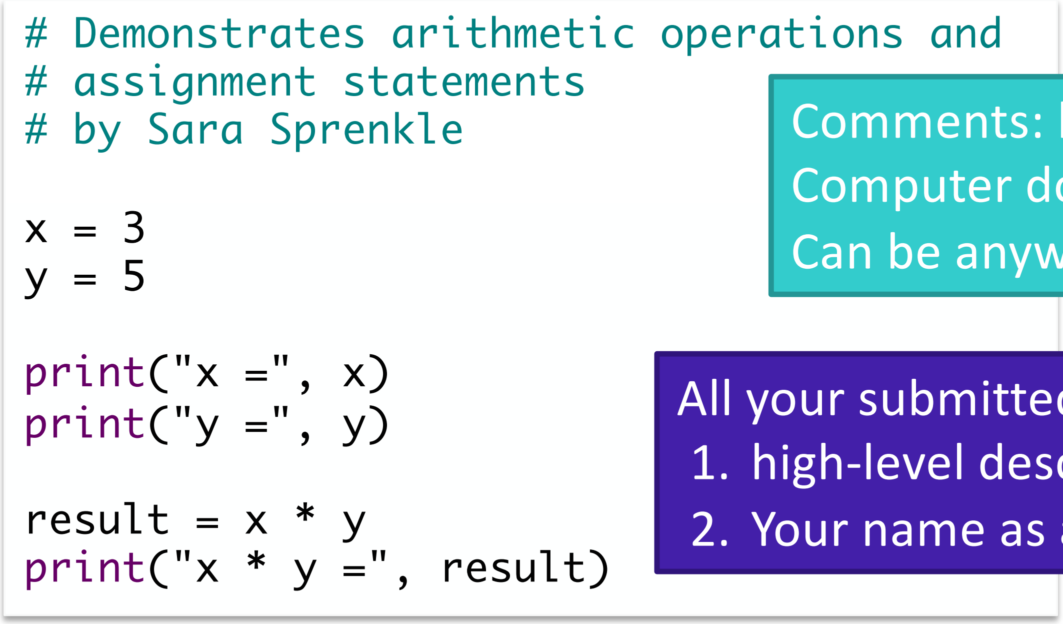
Program displays same output as previous example

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

A Documented Program



```
# 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.
Can be anywhere in code.

All your submitted programs ***must*** have

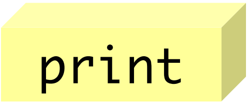
1. high-level description of what the program does
2. Your name as author and date you authored it

arith_and_assign.py

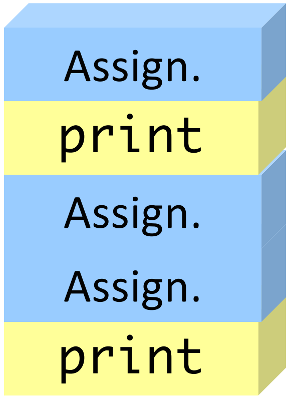
Programming Building Blocks

- Each type of statement is a building block
  - Initialization/Assignment

    - So far: Arithmetic
  - Print



- We can combine them to create more complex programs
  - Solutions to problems

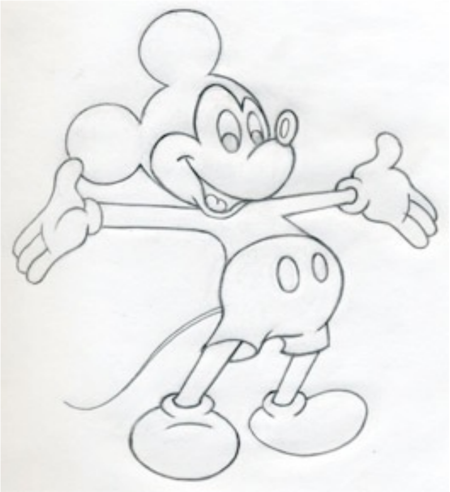


DEVELOPMENT PROCESS

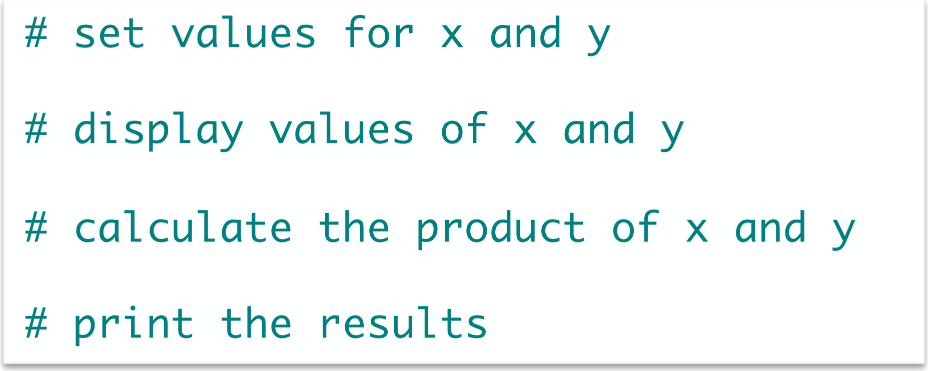
Formalizing Process of Developing Computational Solutions

1. Create a sketch of how to solve the problem (the algorithm)

Use comments to describe the steps



Example sketch for previous Python program:



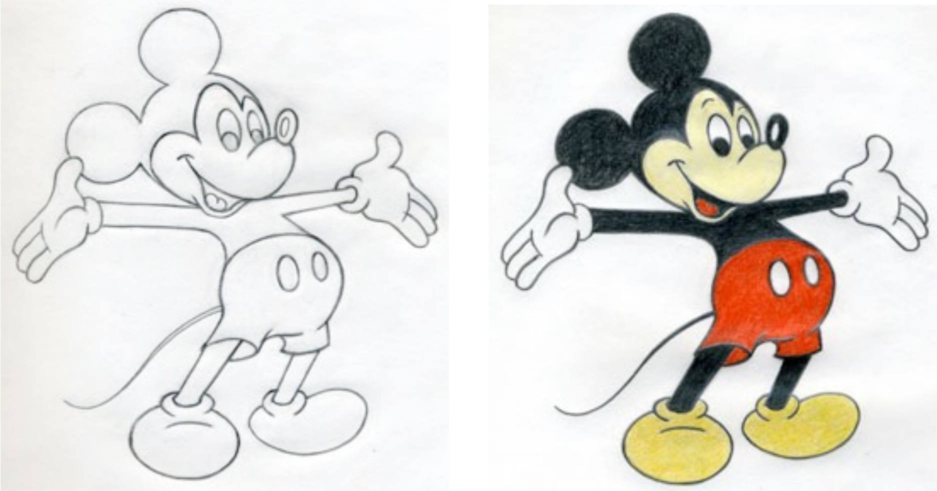
```
# set values for x and y

# display values of x and y

# calculate the product of x and y

# print the results
```

Formalizing Process of Developing Computational Solutions

1. Create a sketch of how to solve the problem (the algorithm)

Use comments to describe the steps

2. Fill in the details in Python



```
# set values for x and y
x = 3
y = 5

# display values of x and y
print("x =", x)
print("y =", y)

# calculate the product of x and y
…
```

Formalizing Process of Developing Computational Solutions

1. Create a sketch of how to solve the problem (the algorithm)
2. Fill in the details in Python
3. Execute the program
   - ➢ Test: does the program's output match your expectation?

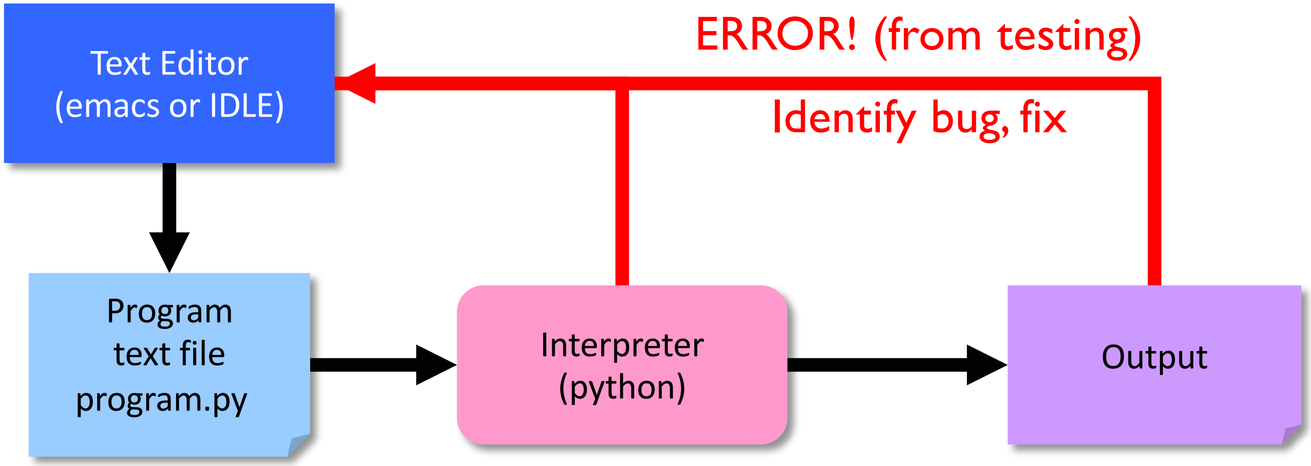
May not have everything filled

It worked! ☺ Or, it didn't ☹

- Sometimes the program doesn't work
- Types of programming errors:
  - Syntax error
    - Interpreter shows where the problem is
  - Logic/semantic error
    - answer = 2+3
    - No, answer should be *2*3*
  - Exceptions/Runtime errors
    - answer = 2/0
    - Undefined variable name

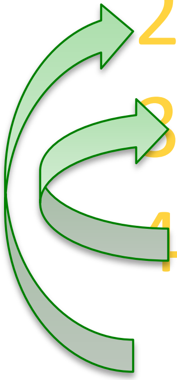
Debugging

- After executing program and output did not match what you expected
- Identify the problems in your code
  - Edit the program to fix the problem
  - Re-execute/test until all test cases pass
- The error is called a “bug” or a “fault”
- Diagnosing and fixing error is called ***debugging***



Formalizing Process of Developing Computational Solutions

1. Create a sketch of how to solve the problem (the algorithm)
2. Fill in the details in Python
3. Execute the program
4. If output doesn't match your expectation
   - Debug the program (Where is the problem? How do I fix it?)



Not necessarily complete program at first

Our development process will evolve over time

Good Development Practices

- Design the algorithm
  - Break into pieces
- Write comments FIRST for each step
  - Elaborate on what you’re doing in comments when necessary
- **Implement** *and* **Test** each piece *separately*
  - Identify the best pieces to make progress
  - Iterate over each step to improve it

When to Use Comments

- Document the author, high-level description of the program at the top of the program

- Provide an outline of an algorithm
  - Separates the steps of the algorithm

- Describe difficult-to-understand code

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

More on Arithmetic Operations

Symbol	Meaning	Associativity
+	Addition	Left
–	Subtraction	Left
*	Multiplication	Left
/	Division	Left
%	Remainder (“mod”)	Left
**	Exponentiation (power)	Right

Precedence rules: P E - MD% AS

negation

More on Arithmetic Operations

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

Associativity matters when you have the same operation multiple times. It tells you where you should start computing.

Precedence rules: P E - MD% AS

negation

Two Division Operators

/ Float Division

- Result is a `float`
- Examples:
  - 6/3 → 2.0
  - 10/3 → 3.3333333333333335
  - 3.0/6.0 → 0.5
  - 19/10 → 1.9

// Integer Division

- Result is an `int`
- Examples:
  - 6//3 → 2
  - 10//3 → 3
  - 3.0//6.0 → 0.0
  - 19//10 → 1

Integer division is the *default* division used in many programming languages

Python Division Practice

1. `6.0//12 * 5.0`
2. `12 // 4 * 5.2`
3. `a = 12//5`
4. `b = 6/12`
5. `z = a / b`

Showing a mix of expressions
(just expression and within assignment statements;
integers and floats)

Python Math Practice

1. `5 + 3 * 2`
2. `2 * 3 ** 2`
3. `-3 ** 2`
4. `2 ** 3 ** 3`

Modulo Operator: %

- Modular Arithmetic: Remainder from division
  - x % y means the remainder of x//y
  - Read as "x mod y"
- Example: 6 % 4
  - Read as "six mod four"
  - 6//4 is 1 with a remainder of 2, so 6%4 evaluates to 2
- Typical use: only with positive integers
- Precedence rules: P E - MD% AS

Modulo Practice

1. `7 % 2`
2. `3 % 6`
3. `6 % 2`
4. `7 % 14`
5. `14 % 7`
6. `6 % 0`

Looking Ahead

- Pre Lab 1 due tomorrow *before* lab
- Our first broader issue is due Thursday at 11:59 p.m.
- Lab 1 will be due on Friday