Objectives

- Assignments and Arithmetic
- Software development practices
  - Testing
  - Debugging
  - Iteration
- User input

Get handouts from Canvas

1

Review

- How do we tell our program to display output?
- How can we store information?
  - What is the syntax to do that?
- What are the rules and conventions for variable names?
  - What is another term for "variable names"?
  - Describe characteristics of *good* variable names
- What are the primitive types of information in Python?
- What are the arithmetic operators? Describe their syntax and semantics.

2

Review: Numeric Arithmetic Operations

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

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?

5

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`

Try these expressions out in interactive mode!

6

Think-Pair-Share

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`

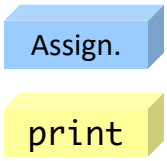
What are the values?

- 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!*

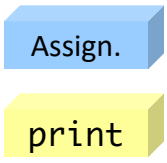
Programming Building Blocks

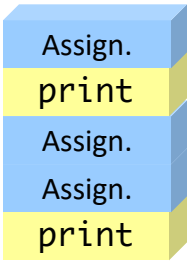
- Each type of statement is a building block
  - Initialization/Assignment
    - So far: Arithmetic
  - Print



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





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 output?

arith_and_assign.py

11

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)
```

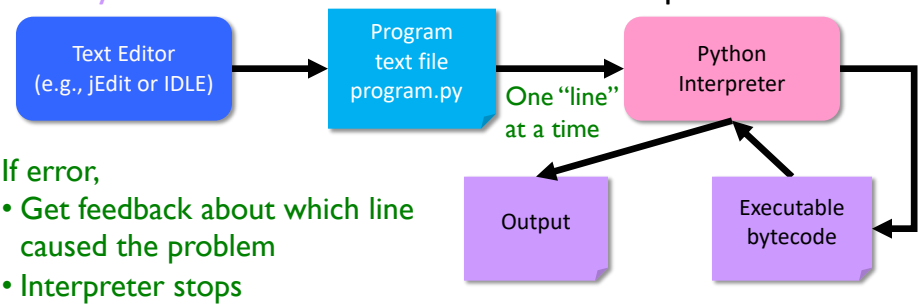
If no print statements, the program would not *output* anything!

arith_and_assign.py

12

Batch Mode: Execute Scripts

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



If error,
- 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)

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

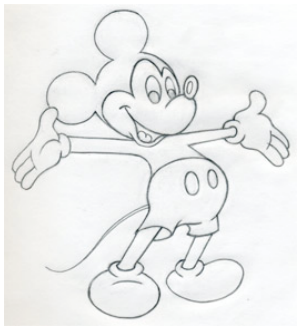
Comments: human-readable descriptions. Computer does not execute.

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

Formalizing Process of Developing Computational Solutions

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

Use comments to describe the steps



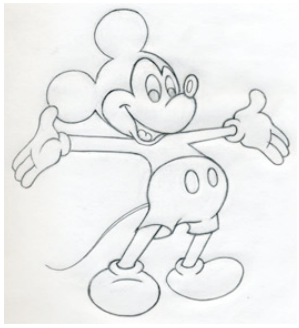
15

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



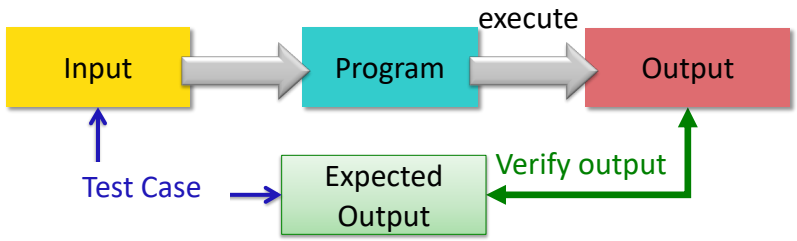
16

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

17

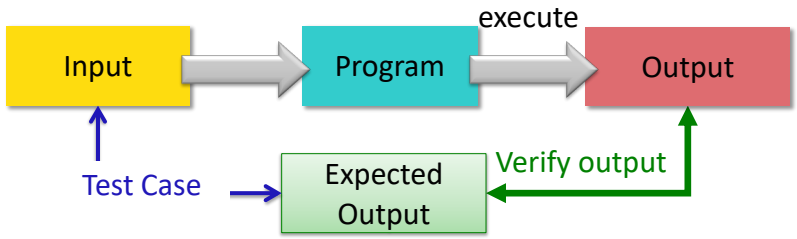
Testing Process



- Test case:
  - **input** used to test the program
  - **expected output** given that input
- Verify if output is what you expected

18

Testing Process



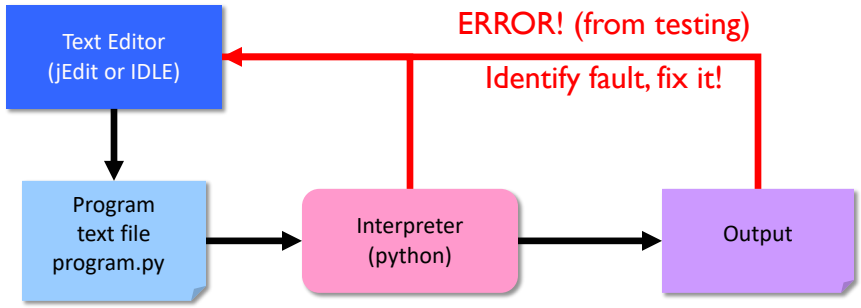
- Test case:
  - **input** used to test the program
  - **expected output** given that input
- Verify if output is what you expected
- Goal: create good test cases that will reveal if there is a problem in your code

If output is not what you expect…

19

Debugging

- After identifying errors during *testing*
- 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***



20

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
3. Test code using *good, varied* test cases to try to find errors in code
4. If program's output does not match the expected output, debug to find the problem and fix it
   - Repeat testing and debugging until no more faults

Practice: A Computational Algorithm

- Find the average of two numbers
- Start the Process:
  1. Create a sketch of how to solve the problem (the algorithm)
  2. Fill in the details in Python
  3. Come up with good test cases for the problem

Practice: A Computational Algorithm

- Find the average of two numbers
- Test Cases

Input		
num1	num2	Expected Output

A Computational Algorithm

- Algorithm for finding the average of two numbers:
  1. "Hard-code" two numbers
     - Later: get the two numbers from user
  2. Calculate average
  3. Print average
- Test cases for finding the average
  - Test both integers
  - Test with at least one float
  - Test numbers less than or equal to 0

 `average2.py`

Good Development Practices

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

`average2.py`

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

Formalizing Process of Developing Computational Solutions

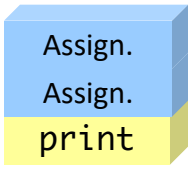
1. Create a sketch of how to solve the problem (the algorithm)
2. Fill in the details in Python
3. Test code using *good, varied* test cases to try to find errors in code
4. If program's output does not match the expected output, debug to find the problem and fix it
   - Repeat testing and debugging until no more faults
5. Make code "better", test again
   - Better variable names, output, comments

Design Patterns

- General, repeatable solution to a commonly occurring problem in software design
  - Template for solution

Design Patterns

- General, repeatable solution to a commonly occurring problem in software design
  - Template for solution
- Example (Standard Algorithm)
  - Get input from user
  - Do some computation
  - Display output




```
x = input("…")
ans = …
print(ans)
```


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

31

More on Arithmetic Operations

Symbol	Meaning	Associativity
+	Addition	Left
-	Subtraction	Left
*	Multiplication	Left
/	Division	Left
%	Remainder ("mod")	Left
**	Exponentiation (powe	

Precedence rules: P E - MD% AS

negation

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

32

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

- `a = 12//5`
- `12 // 4 * 5.0`
- `b = 6/12`
- `6.0//12 * 5.0`
- `z = a / b`

Looking Ahead

- Prelab 1 due tomorrow before lab
- Lab 1 due Friday
- Broader Issue due Friday