Objectives

- Continuing fundamentals of programming
- Numeric Operations
- Introduction to design patterns
- Software development practices
  - Testing
  - Debugging
  - Iteration

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

Review

- What are Python's primitive data types and what do they represent?

Recap of 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: Assignment statements

- Assignment statements are NOT math equations!

```
count = count + 1
```

- These are commands!

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

What is the value of y?

Review: 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!*

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 - DM% AS

negation

Associativity matters when you have the same operation multiple times

NOT Math Class

- Need to write out all operations explicitly
  - In math class, a (b+1) meant $\underbrace{\texttt{a * (b+1)}}_{\text{Write this way in Python}}$

Math Practice

```
5+3*2
2 * 3 ** 2
-3 ** 2
2 ** 3 ** 3
```

How should we verify our answers?

Two Division Operators

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

// Integer Division
- Result is an `int`
- Examples:
  - `6//3 → 2`
  - `10//3 → 3`
  - `3.0//6.0 → 0`

Integer division is the division used in most programming languages

Integer Division Practice

- What is the result?
- What is the **type** of the LHS variable?

```
x = 6//4
y = 4 // 6 * 5.0
a = 6/12.0
b = 6.0//12
z = x / a
z = x // a
```

What is integer division good for?

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

Review: Printing Output

- `print` is a special ***function***
  - Displays the result of expression(s) to the terminal
- `print("Hello, class")`

  string literal

  `print` automatically adds a `'\n'` (carriage return) after it's printed
- `print("Your answer is", 4*4)`

Syntax: comma
Semantics: print multiple "things" in one line

Interactive Programs

- Meaningful programs often need input from users
- Demo: `input_demo.py`

Getting Input From User

- `input` is a ***function***
  - **Function**: A command to do something
    - A "subroutine"
  - Prompts user for input, gets the user's input
  - `input`: reads input in as strings/text
- Syntax:
  - `input(<string_prompt>)`

Getting Input From User

- Typically used in assignments
- Examples:

  Prompt displayed to user

  - `name=input("What is your name? ")`
    - `name` is assigned the string the user enters
  - `width=eval(input("Enter the width:"))`
    - **What the user enters is evaluated (as a number) and assigned to `width`**
    - Use `eval` function because expect a number from user

What do you think the code looks like for `input_demo.py`?

Getting Input from User

```
color = input("What is your favorite color? ")
```

Semantics: Sets the variable `color` to the user's input

Terminal:

Grabs every character up to the user presses "enter"

```
> python3 input_demo.py
What is your favorite color? blue
Cool!  My favorite color is _light_ blue !
```

`input_demo.py`

Documenting Your Code

"Programs should be written for people to read, and only incidentally for machines to execute."
from "Structure and Interpretation of Computer Programs" by Abelson and Sussman

- Use English to describe what your program is doing in ***comments***
  - Everything after a **#** is a comment
    - Color-coded in IDLE, jEdit
  - Python does not execute comments
- Does not affect the correctness of your program
- Improves program's ***readability***
  - Easier for someone else to read and update your code

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

Identify the Parts of a Program

```
# Demonstrate numeric and string input
# by Sara Sprenkle for CS111
#

color = input("What is your favorite color? " )
print("Cool!  My favorite color is _light_", color, "!")

rating = eval(input("On a scale of 1 to 10, how much do
you like Ryan Gosling? "))
print("Cool!  I like him", rating*1.8, "much!")
```

Identify the comments, variables, functions, expressions, assignments, literals

input_demo.py

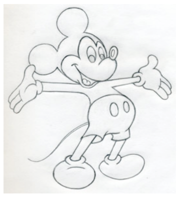
Identify the Parts of a Program

```
# Demonstrate numeric and string input
# by Sara Sprenkle for CS111
#

color = input( "What is your favorite color? " )
print("Cool!  My favorite color is _light_" , color, "!")

rating = eval(input( "On a scale of 1 to 10, how much do
you like Ryan Gosling? " )
print("Cool!  I like him" , rating*1.8, "much!")
                            expression
```

Identify the comments, variables, functions, expressions, assignments, literals

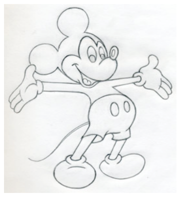
Formalizing Process of Developing Computational Solutions

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



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

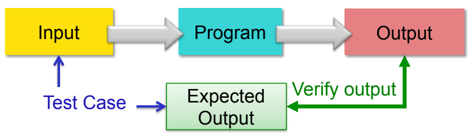


Errors

- 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

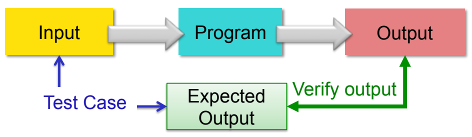
Expose errors when **Testing**

Testing Process



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

Testing Process

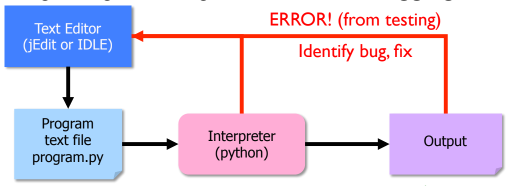


- Need ***good*** **test cases** to help determine if program is correct
  - Tester plays devil's advocate
  - Want to expose **all** errors!
  - Find before customer/professor!

If output is not what you expect…

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***



debug_practice.py

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 the Python program with *good* test cases
   a. If errors found, debug program
   b. Repeat step 3

Practice: Our First Computational Algorithm

- Find the area of a rectangle, which has a width and height
- Test cases:

Input	Expected Output

Our First Computational Algorithm

- Algorithm for finding the area of a rectangle:
  - Optional: get the width and height from user
    - Alternative: "hard-code" width and height
  - Calculate area
  - Print area
- Test cases for finding the area of a rectangle
  - Test both integers
  - Test with at least one float for width, height
  - Test numbers less than or equal to 0
    - Shouldn't compute area for those

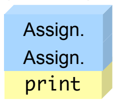
area.py

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)
```

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

This Week

- Tuesday: Lab 1
  - ➢ Bring your lecture notes and handouts!
  - ➢ Due Friday
- For Friday, read up to (but not including) "Themes" of Four Puzzles from Cyberspace
  - ➢ Post summary on Sakai
  - ➢ To paste from Word, click on the icon "Paste from Word"
    - Looks like a clipboard with Word's W