How to develop a program

CSE 160

University of Washington

Program development methodology: English first, then Python

1. Define the problem
2. Decide upon an algorithm
3. Translate it into code

Try to do these steps in order

Program development methodology: English first, then Python

1. **Define the problem**
   A. Write an English description of the input and output **for the whole program**. (Do not give details about *how you will compute* the output.)
   B. Create test cases **for the whole program**
      - Input *and* expected output
2. Decide upon an algorithm
3. Translate it into code

Try to do these steps in order

Program development methodology: English first, then Python

1. Define the problem
2. **Decide upon an algorithm**
   A. Implement it in English
      - Write the recipe or step-by-step instructions

   B. Test it using paper and pencil
      - Use small but not trivial test cases
      - Play computer, animating the algorithm
      - Be introspective
        - Notice what you really do
        - May be more or less than what you wrote down
        - Make the algorithm more precise
3. Translate it into code

Try to do these steps in order

Program development methodology: English first, then Python

1. Define the problem
2. Decide upon an algorithm
3. **Translate it into code**
    - A. Implement it in Python
        - Decompose it into logical units (functions)
        - For each function:
            - Name it (important and difficult!)
            - Write its documentation string (its specification)
            - Write tests
            - Write its code
            - Test the function
    - B. Test the whole program

Try to do these steps in order

Program development methodology: English first, then Python

1. Define the problem
2. Decide upon an algorithm
3. Translate it into code

Try to do these steps in order

- It’s OK (even common) to back up to a previous step when you notice a problem
- You are incrementally learning about the problem, the algorithm, and the code
- “Iterative development”

The *Wishful Thinking* approach to implementing a function

- If you are not sure how to implement one part of your function, define a **helper function** that does that task
  - "I wish I knew how to do task X"
  - Give it a name and assume that it works
  - Go ahead and complete the implementation of your function, *using* the helper function (and assuming it works)
  - Later, implement the **helper function**
  - The helper function should have a simpler/smaller task
- Can you test the original function?
  - Yes, by using a stub for the **helper function**
  - Often a lookup table: works for only 5 inputs, crashes otherwise, or maybe just returns the same value every time

ThinkPython 3.12 Why functions?

It may not be clear why it is worth the trouble to divide a program into functions. There are several reasons:

- Creating a new function gives you an opportunity to name a group of statements, which **makes your program easier to read and debug**.
- Functions **can make a program smaller** by eliminating repetitive code. Later, if you make a change, you only have to make it in one place.
- Dividing a long program into functions allows you to **debug the parts one at a time** and then assemble them into a working whole.
- Well-designed functions are often useful for many programs. Once you write and debug one, **you can reuse it**.