Vantage 140 CPSC 110

HtDF critique

Week 2

Lesson Plan:
Learn why the recipe is important to use when designing functions

Intended Learning Outcomes:
- Develop understanding of the importance of readability
- Develop understanding of the different parts of the recipe

Required Materials:
1) Week 2 handout (HtDF critique)
2) Week 2 notes for Discussion Talking Points

Lesson Procedure

	Resource(s)	Time
Main Task 1 – Discussion of the Recipe	Notes: Discussion Talking Points	10 mins
As a class, discuss each aspect of the recipe: its form, its function, how it contributes to problem solving, and how it contributes to readability. Drive this discussion with three questions for each part of the recipe: 1) what is it? 2) Where do I start? 3) Why do we do this?		
Main Task 2 – Critiquing Programs	Week 2 handouts Versions A through E	15-20 mins
Split student into 6 groups of at least 3 students per group. Hand out one problem handout for each group to critique. Have the groups identify the problems in the solution provided and a create a corresponding corrected solution to each problem. Questions are provided on the handout to guide this critique.		
Main Task 3 – Discussion	Week 2 handouts Versions A through E	20-25 mins
Have each group make a lesson plan to teach one of the concepts that is related to one of the mistakes they identified in the solution they were given. They will present this lesson in their VANT 140 Language Class this week.		

Discussion Talking points…

Signature

- What is it…
  - Specify the type of data consumed and produced
  - Can consume multiple arguments but always produces one
  - Type names are capitalized
  - Commented with 2 ;'s
- Where do I start
  - Look at problem description to decide how many arguments are coming into the function and what the result needs to be
- Why do we do this?
  - This is the first step to understanding what kind of data our function has to work with and what it needs to produce.
  - When someone reads your function later they quickly understand what it takes as input and what they can expect out

Purpose

- What is it…
  - short description of what the function does
  - Expands on the signature to describe what is produced in terms of what the function consumes.
  - Can use the parameter names in this description – this can help to make it short
  - Commented with 2 ;'s
- Where do I start
  - Look to the signature for help in writing this. The signature will tell you what the function is going to produce and the type of the arguments you have to work with
  - produces… *the thing to be produced , how it uses the arguments to produce it*
    - ie. produces a number two times n
- Why do we do this?
  - To expand our understanding of how our function will use the data coming in
  - When someone reads your function later they get a quick plain English description of what it does

Stub

- What is it…
  - A dummy function that produces a value of the right type
  - A function definition with the function name + parameter names and a dummy result in the body.
  - Commented with ; stub at the end
  - left commented out with 1 ; once function body is written
- Where do I start
  - Look to the signature and the purpose for help writing this.
    - The signature will tell you how many parameters your function needs.
    - The purpose will help you pick a descriptive function name.
    - The purpose may have also specified the parameter names you should use.
    - The signature will also tell you what type your dummy result should be.
- Why do we do this?

- Don't have to think hard yet about how to write the function body but…
- Can run examples that are going to be written to ensure they are well-formed (no syntax errors)

E**xamples/Tests**

- What are they
  - A comparison of an example function call to what we expect to be the output of that call
  - Do not have to write examples for ALL possible inputs but you should have examples for each category of input – we will talk more about this as we get into more complex functions
- Where do I start?
  - Each example will be of the form:
    - (check-expect (FUNCTION CALL) (EXPECTED RESULT))
    - use the signature, purpose and stub to help write the function call
      - the stub will tell you what the function name is
      - the signature will tell you the type of input to pass to the function and the type of output you should expect
      - the purpose will tell you what to do to the input to get the expected result
- Why do we do this?
  - To help us clarify what it is the function will do with the input
  - To let us start to think about how the function will do what we need it to do
  - To give us tests that can be run after our function is complete ensure correctness

Template

- What is it…
  - The outline of the function that shows us what we have to work with when we write the function body
  - Not runnable
  - Comment out with #; and leave a copy
- Where do I start
  - When consuming atomic data (week 1 HtDF problems) copy the stub and replace the dummy result in the body of the function with (… param-name)
- Why do we do this?
  - Gives us a start to the function and lets us know our function must do something (…) with the incoming data (param-name)

Code the function body

- What is it…
  - Fill in the …
- Where do I start
  - Look back at the result in the examples to understand what it is you need to do with the incoming data (param-name)
- Why do we do this?
  - To complete the function so it is runnable

Test and debug until correct

- What is it…
  - Comment out your stub and run your check-expects
  - If check-expects all pass, you are done

- If check-expects don’t all pass, you will need to ensure your tests are correct and make changes to your function until the test pass

❖ Where do I start
- Click on the run button ☺

❖ Why do we do this?
- To ensure the function we wrote is correct before we move onto to writing more functions