CS 160: Exploring Computer Science

Algorithm Design Document

Make a copy before you begin (File -> Make a copy). Add the Assignment # above and complete the sections below BEFORE you begin to code. The sections will expand as you type. When you are finished, download this document as a PDF (File -> Download -> PDF) and submit it to D2L.

This document contains an interactive checklist. To mark an item as complete, click on the box.

Planning your program before you start coding is part of the development process. In this document, you will:

- ☑ ~~Step 1: Write a detailed description of your program, at least two complete sentences~~
- ☑ ~~Step 2: If applicable, design a sample run with test input and output~~
- ☑ ~~Step 3: Algorithm design~~
  - ☑ ~~Identify the program inputs and their data types~~
  - ☑ ~~Identify the program outputs and their data types~~
  - ☑ ~~Identify any calculations or formulas needed~~
  - ☑ ~~Write the algorithmic steps as pseudocode or a flowchart. Look at the Pseudocode syntax at the bottom of this document. Tools for flowchart - Draw.io - Diagrams.net~~

1. Program Description

In the box below, describe the purpose of the program. You must include a detailed description with at least two complete sentences.

Program description:
This program allows the user to play a game of “Rock, Paper, Scissors.” Paper covers rock, scissors cut paper, rock smashes scissors. The player’s goal is to beat the computer by making a winning choice.

2. Sample Run

If you are designing your own program, you will start with a sample run. Imagine a user is running your program - what will they see? What inputs do you expect, and what will be the outputs from the given inputs? Choose test data you will use to test your program. Calculate and show the expected outputs. Use the sample run to test your program.

Sample run:
Welcome to “Rock, Paper, Scissor: Python Edition”. The goal of this game is to choose between rock, paper, and scissors and defeat the computer player. Player, it is now your turn. Please make a selection by typing “Rock, Paper, or Scissors”:

-‘Rock’

The computer picked scissors, and rock smashes scissors. You win!

OR

The computer picked rock, and paper covers rock. You win!

OR

The computer picked paper, and scissors covers paper. You win!

OR

The computer picked rock, and rock smashes scissors. You lose 😕

OR

The computer picked scissors, and scissors cuts paper. You lose 😕

OR

The computer picked paper, and paper covers rock. You lose :(

OR

You both selected [player_choice], its a tie! Try again!

Would you like to try again?

-’No’

Thanks for playing!

3. Algorithmic Design

Before you begin coding, **you must first plan out the logic** and think about what data you will use to test your program for correctness. All programmers plan before coding - this saves a lot of time and frustration! Use the steps below to identify the inputs and outputs, calculations, and steps needed to solve the problem.

Use the pseudocode syntax shown in the document, supplemented with English phrases if necessary. **Do not include any implementation details (e.g. source code file names, or language syntax)**. Do not include any Python specific syntax or data types.

Algorithmic design:
a. Identify and list all of the user input and their data types. Include a variable name, data type, and description. Simple data types include string, integer, floating point, (single)

character, and boolean. Complex Data structures like lists should be referenced by name, e.g. “array of integer” or “array of string”.

Player_move – string

Do they want to play again? –string

b. Identify and list all of the user output and their data types. Include a variable name, data type, and description. Data types include string, integer, floating point, (single) character, and boolean. Complex Data structures like lists should be referenced by name, e.g. “array of integer” or “array of string”.

Winning message – string

Losing message – string

Tie – string

Updated score – integer

Exit message – string

Is player_move == Computer_move? – boolean

c. What calculations do you need to do to transform inputs into outputs? List all formulas needed, if applicable. If there are no calculations needed, state there are no calculations for this algorithm.If there are no calculations needed, state there are no calculations for this algorithm. Formulae should reference the variable names from step a and step b as applicable.

Does Player_move == Computer_move?

+1 to score after each round

d. Design the logic of your program using pseudocode or flowcharts. Here is where you would use conditionals, loops, or functions (if applicable) and list the steps in transforming inputs into outputs. Walk through your logic steps with the test data from the assignment document or the sample run above. **Use the syntax shown at the bottom of this document and plain English phrases. Do not include any implementation details (e.g. file names) or Python or any language specific syntax.**

DISPLAY welcome message and rules

DECLARE score

SET Score = 0 (int)

```
DECLARE player_move (string) as variable: Rock/paper/scissors

DECLARE comp_move (string) as variable: SET Random: Rock/paper/scissors

WHILE TRUE set a loop point

INPUT player_move (.lower)

Is player_move == “rock”, “paper”, or “scissors”?

THEN continue

ELSE print “Invalid Entry”

IF player_move == comp_move DISPLAY “Tie, try again”

ELIF player_move = rock AND comp_move = scissors

    OR player_move = scissors AND comp_move = paper

    OR player_move = paper AND comp_move = rock

DISPLAY “You Win”

And SET score = +1

ELSE DISPLAY “You Lose”

And SET score = -1

DISPLAY “Play again? Enter Y or N”

INPUT

IF Y, restart at WHILE TRUE

ELIF N, break the loop and DISPLAY “Thanks for playing!”

ELSE anything else

PRINT “Invalid choice, choose Y or N”
```

Pseudocode Syntax

Think about each step in your algorithm as an action and use the verbs below:

To do this:	Use this verb:	Example:
Create a variable	DECLARE	`DECLARE integer num_dogs`
Print to the console window	DISPLAY	`DISPLAY "Hello!"`
Read input from the user into a variable	INPUT	`INPUT num_dogs`
Update the contents of a variable	SET	`SET num_dogs = num_dogs + 1`
Conditionals		

Use a single alternative conditional	IF *condition* THEN
	  statement
	  statement
	END IF
		`IF num_dogs > 10 THEN`
		`  DISPLAY "That is a lot of dogs!"`
		`END IF`

Use a dual alternative conditional	IF *condition* THEN
	  statement
	  statement
	ELSE
	  statement
	  statement
	END IF
		`IF num_dogs > 10 THEN`
		`  DISPLAY "You have more than 10 dogs!"`
		`ELSE`
		`  DISPLAY "You have ten or fewer dogs!"`
		`END IF`

Use a switch/case statement	SELECT *variable or expression*
	  CASE *value_1:*
	  statement
	  statement
	  CASE *value_2:*
	  statement
	  statement
	  CASE *value_2:*
	  statement
	  statement
	  DEFAULT:
	  statement
	  statement
	END SELECT
		`SELECT num_dogs`
		`  CASE 0: DISPLAY "No dogs!"`
		`  CASE 1: DISPLAY "One dog.."`
		`  CASE 2: DISPLAY "Two dogs.."`
		`  CASE 3: DISPLAY "Three dogs.."`
		`  DEFAULT: DISPLAY "Lots of dogs!"`
		`END SELECT`

Loops		

Loop while a condition is true - the loop body will execute 0 or more times.	WHILE *condition*
	  statement
	  statement
	END WHILE
		`SET num_dogs = 1`
		`WHILE num_dogs < 10`
		`  DISPLAY num_dogs, " dogs!"`
		`  SET num_dogs = num_dogs + 1`
		`END WHILE`

Loop while a condition is true - the loop body will execute 1 or more times.	DO
	  statement
	  statement
	WHILE *condition*
		`SET num_dogs = 1`
		`DO`
		`  DISPLAY num_dogs, " dogs!"`
		`  SET num_dogs = num_dogs + 1`
		`WHILE num_dogs < 10`

Loop a specific number	FOR *counter* = *start* TO *end*	`FOR count = 1 TO 10`

of times.	*statement* *statement* END FOR	DISPLAY num_dogs, " dogs!" END FOR
Functions		
Create a function	FUNCTION *return_type name (parameters)* *statement* *statement* END FUNCTION	FUNCTION Integer add(Integer num1, Integer num2) DECLARE Integer sum SET sum = num1 + num2 RETURN sum END FUNCTION
Call a function	CALL *function_name*	CALL add(2, 3)
Return data from a function	RETURN *value*	RETURN 2 + 3