Name student:

Instructions: The pseudocode being graded should contain each of the lines contained in the left column of the rubric below in order to receive the score contained in the middle column. There is not partial credit, and the score in the middle column is either all or nothing. Note\* - The pseudocode does not have to be an exact match to the pseudocode listed in the left column. As long as the logic is the same, and the overall design is consistent with the expectations, full credit may be given. For example, ` currentPop = 307357870 ` is equivalent to

` populationToday = 307357870 ` and ` secondsPerBirth = 7 ` would be the same as `birthsEachSec = 1/7 `, etc.

**Problem 1: 10p**

|  |  |  |
| --- | --- | --- |
| **Definition of constants** |  | Your score |
| currentPop = 307357870 | 0.5 |  |
| secondsPerYear = 31536000 # (60s/min)\*(60min/hr)\*(24hr/day)\*(365days/yr) | 0.5 |  |
| secondsPerBirth = 7 | 0.5 |  |
| secondsPerDeath = 13 | 0.5 |  |
| secondsPerImmigration = 35 | 0.5 |  |
| **Perform math** |  |  |
| annualBirthRate = secondsPerYear / secondsPerBirth | 0.5 |  |
| annualDeathRate = secondsPerYear / secondsPerDeath | 0.5 |  |
| annualImmigrantRate =secondsPerYear / secondsPerImmigration | 0.5 |  |
| annualPop = annualBirthRate ­ annualDeathRate + annualImmigrantRate | 1 |  |
| projectedPop = annualPop + currentPop | 1 |  |
| **Print** |  |  |
| print("The population is”, projectedPop) | 1 |  |
| Total | 7 |  |

**Problem 2: 10p**

|  |  |  |
| --- | --- | --- |
|  |  | Your score |
| **Getting the input from the user** |  |  |
| number = input("Enter a number:") | 2 |  |
| **Conditionals** |  |  |
| if number > 84000 or number < 1:  exit | 1 |  |
| **Math and Variables** |  |  |
| hours = floor(number/3600)  number -= hours\*3600  minutes = floor(number/60)  number -= minutes\*60  seconds = number  # Or equivalent logic to the pseudocode above | 5 |  |
| **Print** |  |  |
| print (“The time is” + hours + “hours,” + minutes + “minutes, and” + seconds + “seconds”) | 2 |  |
| Total | 10 |  |

**Problem 3: 3p**

|  |  |  |
| --- | --- | --- |
|  |  | Your score |
| **Variables and Math** |  |  |
| fahrenheit = #some value  celsius = (5/9)\*(fahrenheit - 32) | 2 |  |
| **Return or Print** |  |  |
| return celsius  # OR  print celsius | 1 |  |
| Total | 3 |  |

**Problem 4: 10p**

|  |  |  |
| --- | --- | --- |
| # A variety of different valid implementations exist |  | Your score |
| **While loop** (true): #or while input is invalid, etc. | 2 |  |
| number=input("Enter a number") | 2 |  |
| **Correct conditionals** | 5 |  |
| if number greater than 10 or number less than 1:  print number  return # exit |  |  |
| **print statement** |  |  |
| print “Please Try Again” | 1 |  |
| Total | 10 |  |

**Problem 5: 20p**

|  |  |  |
| --- | --- | --- |
| **Getting the input from the user** |  | Your score |
| milespergallon= input("Enter the no. of miles per gallon:") | 5 |  |
| **3 cases** |  |  |
| if milespergallon >= 30:  print “nice job.” | 5 |  |
| else if milespergallon >= 15 and milespergallon <= 29:  print “not great, but okay.” | 5 |  |
| else:  print “so bad, so very, very bad.” | 5 |  |
| Total | 20 |  |

**Problem 6: 20 p**

|  |  |  |
| --- | --- | --- |
| **Defining variables and getting the input** |  | Your score |
| a = ”Fight the Dragon” b = ”Go Home” c = ”Save the Princess” choice = input(“Enter your choice: a) Fight the dragon b) Go Home c) Save the princess”) | 5 |  |
| **Loop to repeatedly ask the user for an option with right conditionals** |  |  |
| While (choice != b): | 3 |  |
| if choice equals a:  print “you win” | 5 |  |
| else: # choice == c  print “You saved the princess”  choice = input(“Enter another choice:”) # Not  under the else. | 5 |  |
| print “Wimp” | 2 |  |
| Total | 20 |  |

**Problem 71: 30p**

|  |  |  |
| --- | --- | --- |
| **Variables** |  | Your score |
| # Given gameboard as an argument to the algorithm, or you could define your own…  # You may have implemented your own boundary function, or you may assume the function was provided for you, or that another function provided to you such as move(direction) checked if a move was valid  # You may assume move functions exist, or you may have defined your own  # You may assume that your move functions keep track of the forward direction for you based on your current position and the direction from whence you moved from.  WHITE = 1  BLUE = 2  GREEN = 3  BLACK = 4  YELLOW = 5 | 5 |  |
| cur\_pos = (0, 0) # or equivalent  forward\_tile = (0, 1) # or equivalent  cur\_tile = gameboard[cur\_pos[0] ][cur\_pos[1] ] # or equivalent | 5 |  |
| **Looping until treasure is found** |  |  |
| while(cur\_tile != YELLOW): # or equivalent | 5 |  |
| **Logic** |  |  |
| if cur\_tile == WHITE:  cur\_pos, forward\_tile = moveForward(cur\_pos, forward\_tile)  else if cur\_tile == BLUE:  cur\_pos, forward\_tile = moveLeft(cur\_pos, forward\_tile)  else if cur\_tile == GREEN:  cur\_pos, forward\_tile = moveRight(cur\_pos, forward\_tile)  else if cur\_tile == BLACK:  cur\_pos, forward\_tile = moveBackTwo(cur\_pos, forward\_tile)  cur\_tile = gameboard[cur\_pos[0] ][cur\_pos[1] ] # update cur\_tile | 10 |  |
| **end while** |  |  |
| print “Congratulations, you found the treasure!” | **5** |  |
| Total | 30 |  |