Post Diagnostic Report

The purpose of the Diagnostic was to help you self assess where you can improve. Every single person in this class has some area of the material where they could use some extra practice. In this document we are going to go over each of the problems in the diagnostic.

Here is a link to your answers:
https://us.edstem.org/courses/490/lessons/1198/slides/6432

Can I get extra help?
Yes! We are hosting two live sessions to discuss the midterm where folks from the teaching team go over the problems and you can ask questions live. They will be recorded!
Thursday, 10-11pm PDT: zoom link
Friday, 9-10am PDT: zoom link
If those are in high demand we are happy to host more.

How can I check my answers?
With this document you can check your answer. For each problem we include

1. Several common solutions
2. Comments which explain the solutions
3. The major concepts that the problem covers
4. Which readings and which lecture videos relate to the problem
5. Related problems to try if you want more practice.

I wasn't able to solve many of the problems on the diagnostic. Should I panic?
Absolutely not! It is normal to find these problems hard. If you didn't do as well as you would have liked you have a two step procedure. Step 1: make sure you know the underlying concepts (for example, using the i variable in a for loop). Then if you know the concepts but you weren't able to come up with solutions in time the simple solution is practice and review! Many people incorrectly think that they are not good at this. False! We all can code. You just need more experience. Check out the review session.

My answer is different from the provided solution. Is my answer wrong?
Not necessarily. You will learn a lot by looking at the difference between the teaching team solution and your work. Your answer might be great. There are many solutions.

Can I discuss my solution on ed?
Yes! You are free to share your solutions

Problem 1: Debugging & Tracing

```
Part A: The divide_and_round function correctly divides n by 2 and
rounds up to the nearest whole number, but does not return or print n.
Therefore, the value of n in main’s print statement is still 42, and
so 42 gets printed.
```

```
"""
Part B: Here is the fixed program
"""
def divide_and_round(n):
   """
   Divides an integer n by 2 and rounds
   up to the nearest whole number
   """
   if n % 2 == 0:
       n = n / 2
   else:
       n = (n + 1) / 2
   return n

def main():
   n = 42
   n = divide_and_round(n)
   print(n)
```

Concepts
This problem focused on functions are return values. If you found it confusing, here are a few resources: This chapter covers functions and return values. Return values are notoriously one of the hardest concepts in the first half of CS106A.

These lecture videos cover functions and return values:

Problem 2a: Print Odd Numbers

```
"""
There are many good ways to approach this problem! Here are a few.
"""

def main():
 # this approach loops 100 times
 for i in range(100):
   print(2 * i + 1)   # each time i goes up by 1, this term goes up by 2

#------------------------------------------------------------#

 # this approach loops 200 times and looks for odd numbers
 for i in range(200):
   if i % 2 == 1:     # is i odd? If you divide by 2 is 1 left over?
     print(i)         # if i is odd, print it!

#------------------------------------------------------------#

 # this approach uses a while loop
 v = 1             # this number keeps track of the current oddnum
 while v < 200:    # keep going while its less than 200
   print(v)        # dont forget to actually print it
   V += 2          # change the value to increase by 2
```

There are many other strategies that are just as valid!

Concepts
The main concepts that this problem requires is looping and variables. The most mind boggling part is that in a for loop, you are allowed to use the counter variable (often i). If you had trouble with this here is the chapter on for loops in the book. Here is the lecture where we covered it.

Details
If you have a structure that largely works, the most important detail to watch out for is: do you have an off by one issue? A common bug would be to only print the first 99 odd numbers. Which is a very small bug to have.

Related Problems
There are many challenges you could give yourself for more practice. Print the first 100 even numbers backwards. Print multiples of 3 under 1000. Try your solutions here.

Problem 2b: Ride the Rollercoaster

```
"""
Write your answer for odd numbers below here:
"""

def main():
 height = float(input("Enter height in meters: "))
 if height < 1 or height > 2:
   print("You can't ride the roller coaster")
 else:
   print("You can ride the roller coaster ")

if __name__ == "__main__:
 main()
```

Concepts
This problem combines a few concepts. The major ones are: conditionals (like if/else statements) and getting input from a user.

Here are the chapters on if-statements and input.
Here is the lecture where we covered if statements and input.

If you got the main idea, the details to watch out for are: did you include quotes around your strings? Did you remember to convert your input into a number? We are not very concerned about whether you accepted someone who was *exactly* 1m or 2m tall. That is a tiny detail!

Related Problems
This worked example is very similar. If you want more practice, try and solve that problem!

Problem 3: Ramp Climbing Karel

Here are two example solutions

```
from karel.stanfordkarel import *

def main():
   while front_is_clear():
       put_beeper()             # one beeper per step
       move_to_next_step()      # what a nice decomposition!
   put_beeper()                 # this is needed for the fencepost

def move_to_next_step():        # there are three ways to write this
   move()
   move()
   turn_left()
   move()
   turn_right()

def turn_right():
   for i in range(3):
       turn_left()

if __name__ == "__main__":
   run_karel_program()
```

```
from karel.stanfordkarel import *

def main():
   put_beeper()                # another way to solve the fencepost
   while front_is_clear():
       turn_left()             # this is move_to_next_step with no fn
       move()
       turn_right()
       move()
       move()
       put_beeper()            # you need at least one put in the loop

def turn_right():
   for i in range(3):
       turn_left()

if __name__ == "__main__":
   run_karel_program()
```

Continued on the next page...

Strategies
There were many ways to solve this problem. It was necessary to use a while loop to solve this problem.

There are three correct ways to move karel to the next beeper spot:

> *left, move, right, move, move*
> *move, left, move, right, move*
> *move, move, left, move, right*

Inside the while loop either the first or the last statement should be a put beeper. This problem has a "fencepost" challenge. You either need a put beeper before or after the while loop.

Many people put if statements around their moves to make sure they wouldn't walk into a wall. Not a bad idea, but it is not necessary since you are promised the world is even.

Concepts
The main concept here is while loops. Here is the reader chapter:
https://compedu.stanford.edu/karel-reader/docs/python/en/chapter6.html

Here are the video where we cover while loops:
https://www.youtube.com/watch?v=S5y2u7VITMo&feature=emb_logo

Decomposition?
We found it helpful to define a function "move to next step" which took Karel from one beeper location to the next. It wasn't necessary but it makes the problem much easier to solve!

Related Problems
You can use the Karel Playground in ed to try solving related problems.

Can you make Karel place a diagonal line of slope 1?

This problem becomes much more difficult if you try and make it work for odd sized worlds as well!

Problem 4: Nondecreasing

This problem is complex because you need to keep track of several variables. There are so many ways to solve this problem! They generally fall into two main categories.

Those who used a **while loop which goes while curr >= last**

```python
def main():
   print("Enter a sequence of non-decreasing numbers. ")
   count = 0                      # essential to keep track of length
   last_num = float(input('Enter num: '))
   curr_num = last_num            # can have another enter num here
   while curr_num >= last_num:    # there are many ways to write this!
       count += 1                 # this line can go anywhere in body
       last_num = curr_num        # update the last num first!
       curr_num = float(input('Enter num: '))
   print("Thanks for playing!")   # watch for string
   print("Sequence length: " + str(count))

if __name__ == "__main__":
  main()
```

Those who used a **while True loop** where they only needed to have a single last_value declared before the loop. This required use of the break keyword.

```python
def main():
   print("Enter a sequence of non-decreasing numbers. ")
   count = 1                       # length is at least 1
   last_num = float(input('Enter num: '))
   while True:
       curr_num = float(input('Enter num: '))
       if curr_num < last_num:
           # some people optionally put the thanks for playing here!
           break   # this makes python exit the loop immediately
       count += 1
   print("Thanks for playing!")
   print("Sequence length: " + str(count))  # convert to string

if __name__ == "__main__":
  main()
```

Concepts
This problem mixes variables with control flow. In class we went over the guess number program which was our introduction to variables with control flow. Here is the same example written up as a worked example.

This problem also required you to use variables in new ways, on the fly. If you were able to solve it, that is certainly a good sign that you have mastered the core concepts. If you knew all of the concepts (loops, variables, input and print) and your challenge was putting it together I have good news. That comes with repetitions. Keep doing

Details
Here were some of the details to look out for:

1. Did you handle the case where the input was equal to the last number? Check if you have an equals sign or not in your condition. Should you?
2. Did you convert your input to a float? This is a real nitpick. But lots of people forgot.
3. Did you cast your count variable to a string before printing it?

Related Problems
If you want to try a problem of similar complexity, I would recommend the hailstone problem which was written up as one of the extension options for assignment 2

That's all folks! Thanks for your hard work.