UNIVERSITY OF WATERLOO

UW: CS 115
Introduction to Computer Science 1

Assignment 01
Assignment 01

Due date: Wednesday, January 24th, 5:00 pm Waterloo time

Coverage: Module 01

Files to submit: `a01q1.rkt`, `a01q2.rkt`, `a01q3.rkt`, `a01q4.rkt`

Assignment Guidelines

Language Level

- Beginning Student

Allowable Racket Functions and Special Forms

- `define`
- any functions on numbers found in Section 1.6 of the documentation for Beginning Student Racket

General Expectations

- Remember to use defined constants when appropriate.
- All test data for correctness will always meet the stated assumptions for consumed values: you do not need to handle function arguments that do not conform to the function's specification.

Question 0: Style Guide

Unlike concept checks and quiz questions, assignment questions are graded partly by hand, to check for good coding style. Go to the Coding Style page and read Sections 1 and 2 of the Style Guide. In all the code you submit for this and subsequent assignments, you must adhere to the recommendations in that document.

For this assignment, good style means including a header comment at the top of every file, using appropriate spacing and indentation, and choosing appropriate names for constants and functions when specific names are not requested. Poorly styled code may lose marks, even if it's correct.

If you read ahead in the style guide, you'll find a lot of information about the Design Recipe. In this assignment, you are *not* required to include any Design Recipe components in your submission (apart from function definitions).

There is nothing to submit for this question.

Question 1: Basketball Points

A basketball player's points per game is based on the number of field goals fg, 3-point baskets $3pt$, free throws ft, and a positive number of games played gp, and is calculated as follows:

$$points-per-game\,(fg, 3pt, ft, gp) = \frac{2 \cdot fg + 3 \cdot 3pt + ft}{gp}$$

Write a function `points-per-game` that consumes four non-negative integers: `fg`, `3pt`, `ft` and `gp`, giving the number of field goals, 3-point baskets, free throws, and a positive number of games played, respectively, and produces the points per game for that player as described above.

For example, if you define `points-per-game` in the Definitions window of DrRacket, you could then evaluate the following expressions in the Interactions window:

```
> (points-per-game 0 0 0 1)
0
> (points-per-game 1 3 5 5)
3.2
> (points-per-game 15 6 2 2)
25
> (points-per-game 104 35 8 10)
32.1
```

Save your code in a file called `a01q1.rkt` and submit it to MarkUs.

Question 2: Fibonacci Numbers

The famous Fibonacci numbers are a sequence beginning $0, 1, 1, 2, 3, 5, 8, 13, 21, 34...$, etc. Named after the Italian mathematician Leonardo Fibonacci, this intriguing sequence has widespread applications in mathematics, nature, and computer science.

The nth number in the Fibonacci sequence, F_n, can be calculated by adding the two numbers F_{n-1} and F_{n-2} preceding it in the sequence. However, there is a closed-form mathematical expression that can be used to approximate F_n without knowing the values of F_{n-1} and F_{n-2}:

$$F_n = \frac{\phi^n - ((1-\phi)^n)}{\sqrt{5}}$$

where ϕ is the golden ratio with value $\frac{1+\sqrt{5}}{2}$.

Write a function `fib-approx` that consumes a positive integer n and produces a number equal to F_n using the formula given above. Because of the use of the irrational number, we accept that Racket might answer with an inexact number starting with #i and not a natural number, even though we know that mathematically, the answer should always be an integer.

For example, if you define `fib-approx` in the Definitions window of DrRacket, you could then evaluate the following expressions in the Interactions window:

```
> (fib-approx 0)
0
```

```
> (fib-approx 1)
#i1.0
> (fib-approx 2)
#i1.0
> (fib-approx 3)
#i2.0
> (fib-approx 5)
#i5.000000000000001
> (fib-approx 27)
#i196418.00000000017
```

Save your code in a file called `a01q2.rkt` and submit it to MarkUs.

Question 3: CS 115 marking scheme

Your final CS 115 grade will be computed based on three components: concepts checks and quizzes, assignments, and the final exam. Review the grading page in the course syllabus to remind yourself of the relative weights of these three components. One of the conditions for passing the course is passing the weighted average of the final exam (worth 40%).

For example, if you get 65% on the final exam, you earn 65·0.4=26 marks towards your final grade. To pass the weighted exam grade, you need to earn at least 20 marks (of the available 40) on the exam component of the course. In this example, you earned 26 marks out of the possible 40, and have passed the weighted exam average.

Write a function `min-exam-grade` that consumes two numbers `assignments` and `concept-checks`, both numbers between 0 and 100 (inclusive), giving your final grades on the concept checks and assignments portions of the course, respectively. The function produces a non-negative number giving the minimum grade you need to achieve on the final exam in order to pass the course with a final grade of at least 50.

You don't need to assume that the consumed or produced numbers are integers.

Note that there can be cases where you've earned so few marks in concept checks and assignments that it's *impossible* to pass the course, because you need to score more than 100 on the final exam. Don't perform any special adjustments in that case—just allow the answer to exceed 100.

For example, if you define min-final-exam-grade in the Definitions window of DrRacket, you could then evaluate the following expressions in the Interactions window:

```
> (min-exam-grade 0 0)
125
> (min-exam-grade 100 100)
50
> (min-exam-grade 40 40)
65
> (min-exam-grade 10.1 80)
74.9
```

Save your code in a file called `a01q3.rkt` and submit it to MarkUs.

Question 4: Cheese Ball Counters

A fun contest is to guess the number of cheese balls in a closed jar (where the person whose guess is closest to correct wins!). If you've got the right mathematical tools, you can calculate a pretty good guess for the number of cheese balls based on the dimensions of the jar, the average volume of a cheese ball, and the "packing efficiency" of the cheese balls (the average proportion of space that can be occupied by a bunch of cheese balls packed together). Let's assume that all cheese balls are ideally sphere shapes and have the same size, and the jar is cylindrical.

Write a function `count-cheeseballs` that consumes four numbers (in this order):

- `radius`, a positive number giving the radius of the cylindrical jar (in cm);
- `height`, a positive number giving the height of the jar (in cm);
- `c-radius`, a positive number giving the radius of a single cheese ball (in cm); and
- `efficiency`, a number strictly between 0 and 1 that gives the average fraction of space that can be occupied by a dense packing of cheese balls.

The function produces a natural number giving an ideal estimate for the largest number of cheese balls that can fit in the jar. You may need to look up the formula for the volume of a cylinder (you can assume the jar's glass is infinitely thin) and a spherical cheese ball. You'll probably use the built-in constant `pi`, meaning that your calculations will produce inexact intermediate values. Use the built-in function `floor` and `inexact->exact` to ensure your final result is a natural number.

For example, if you define `count-cheeseballs` in the Definitions window of DrRacket, you could then evaluate the following expressions in the Interactions window:

```
> (count-cheeseballs 1 1 0.1 0.1)
75
> (count-cheeseballs 130 10 10 0.9)
114
```

Save your code in a file called `a01q4.rkt` and submit it to MarkUs.