# Assignment 04

**Due Wednesday, June 17 at 10:00 am (no late submissions)**

**Assignment Guidelines.**

- This assignment covers material in Module 5.
- Submission details:
  - Solutions to these questions must be placed in files `a04q1.rkt`, `a04q2.rkt`, `a04q3.rkt`, and `a04q4.rkt`, respectively, and must be completed using Racket *Intermediate Student*.
  - Unless otherwise indicated in the question you may use only the built-in functions and special forms introduced in the lecture slides from CS115 up to and including the modules covered by this assignment.
  - Download the interface file from the course Web page to ensure that all function names are spelled correctly and each function has the correct number and order of parameters.
  - All solutions must be submitted to MarkUs. No solutions will be accepted through email, even if you are having issues with MarkUs.
  - Verify using MarkUs and your basic test results that your files were properly submitted and are readable on MarkUs.
  - For full style marks, your program must follow the CS115 Style Guide.
  - Be sure to review the Academic Integrity policy on the Assignments page.
  - For the design recipe, helper functions only require a purpose, a contract and an example.
- Restrictions:
  - Unless the question specifically describes exceptions, you are restricted to using the functions and special forms covered in or before Module 5.
  - Read each question carefully for additional restrictions.

    ! Do not use `lambda` on this assignment.

- **The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.**

1. **Enough Vowels.** For our purposes we define a *vowel* as one of the characters `#\a`, `#\e`, `#\i`, `#\o`, or `#\u`. (So here we are *not* counting `#\y`, `#\A`, `#\E`, `#\I`, `#\O`, `#\U`, or `#\Y` as vowels.)

Exercise

Write a function `(vowel-heavies L)` that consumes a `(listof Str)` and returns a list containing all the values from `L` where at least half the characters are vowels.
For example,

```
(check-expect (vowel-heavies (list "a" "banana" "is" "tasty" "and" "good"))
              (list "a" "banana" "is" "good"))
(check-expect (vowel-heavies (list "aa!!" "ee!!!" "WOO")) (list "aa!!"))
```

2. **The Sum of All Ph34rz.**

Exercise

Write a function `(sum-digits s)` that consumes a `Str` and returns a `Nat` which is the sum of all the numeric characters in `s`. For example,

```
(check-expect (sum-digits "31337 H4X0R") 21)
(check-expect (sum-digits "ph34r my 1337 hax0rz ski11z") 23)
(check-expect (sum-digits "What?") 0)
```

Read through the extra documentation on Strings and characters.

3. **A Function for Finding Few Factors.**

Exercise

Write a function `(fff n)` that consumes a `Nat` and returns a `(listof Nat)` containing all the numbers between 1 and $n$ (inclusive) that are divisible by *exactly one* of 2, 3, and 7.
With $n = 15$, numbers divisible by at least one of these values are `(list 2 3 4 6 7 8 9 10 12 14 15)`.
But 6, 12, and 14 are divisible by two of these numbers. So

```
(check-expect (fff 15) (list 2 3 4 7 8 9 10 15))
```

**4. First Glimpse at Databases.** We can store a lot of information by making a list that contains lists.

We are going to think about lists of length exactly 3, representing an author, title, and number of pages. Each such list we will call a `Book`, and write:

```
;; a Book is a (list Str Str Nat)
```

Here are a few examples of a `Book`:

```
(define watney (list "Weir" "The Martian" 369))
(define potter (list "Rowling" "Harry Potter and the Philosopher's Stone" 223))
```

We can extract individual values from a `Book`:

```
(first watney) => "Weir"
(second watney) => "The Martian"
(third watney) => 369
```

We can then make a `(listof Book)`, which can store a lot of information. For example:

```
(define booklist
  (list
   (list "Liu" "The Three Body Problem" 302)
   (list "Nawaz" "Songs for the End of the World" 400)
   (list "Heinlein" "The Moon Is a Harsh Mistress" 382)
   (list "Weir" "The Martian" 369)
   (list "Rowling" "Harry Potter and the Philosopher's Stone" 223)
   ))
```

We can make another `(listof Book)` with one more item in it:

```
(define longer-booklist
  (cons (list "Austen" "Sense and Sensibility" 400) booklist))
```

Exercise

Write a function `(count-pages L)`. It consumes a `(listof Book)` and returns the total number of pages.

```
(check-expect (count-pages booklist) 1676)
```

Exercise

Write a function `(longest-book L)`. It consumes a non-empty `(listof Book)` and returns the `Book` with the largest number of pages. For example:

```
(check-expect (longest-book booklist)
              (list "Nawaz" "Songs for the End of the World" 400))
```

If more than one book has the largest number of pages, return the one closer to the front of the list:

```
(check-expect (longest-book longer-booklist)
              (list "Austen" "Sense and Sensibility" 400))
```

!

Do not use `sort`!
(To sort a list, the computer needs to look through the list many times. But to find the largest, it need only look through only once; this is faster. Do it the fast way!)