LING/COMP 445, LING 645
Problem Set 1

Due before 10:05 AM on Thursday, September 16, 2021

There are several types of questions below.

- For questions involving answers in English or mathematics or a combination of the two, put your answers to the question in the **Answer** section provided, like in the example below. You can find more information about LaTeX here https://www.latex-project.org/.
- For programming questions, please put your answers into a file called `ps1-lastname-firstname.clj`. Be careful to follow the instructions exactly and be sure that all of your function definitions use the precise names, number of inputs and input types, and output types as requested in each question.

  For the code portion of the assignment, **it is crucial to submit a standalone file that runs**. Before you submit `ps1-lastname-firstname.clj`, make sure that your code executes correctly without any errors when run at the command line by typing `clojure ps1-lastname-firtname.clj` at a terminal prompt. We cannot grade any code that does not run correctly as a standalone file, and if the preceding command produces an error, the code portion of the assignment will receive a 0.

  To do the computational problems, we recommend that you install Clojure on your local machine and write and debug the answers to each problem in a local copy of `ps1-lastname-firstname.clj`. You can find information about installing and using Clojure here https://clojure.org/.

Once you have entered your answers, please compile your copy of this LaTeX document into a PDF and submit

(i) the compiled PDF renamed to `ps1-lastname-firstname.pdf`
(ii) the raw LaTeX file renamed to `ps1-lastname-firstname.tex` and
(iii) your `ps1-lastname-firstname.clj`

to the Problem Set 1 folder under 'Assignments' on MyCourses.

Example Problem: This is an example question using some fake math like this $L = \sum_0^\infty \mathcal{G}\delta_x$.

Example Answer: Put your answer right here like this $L = \sum_0^\infty \mathcal{G}\delta_x$.

Problem 1: Write an expression which defines a variable `year` with the integer value `2021`.

Answer 1: Please put your answer in `ps1-lastname-firstname.clj`.

Problem 2: The following is:

A. an expression
B. a list
C. both
D. neither

```
(= 4 (+ 1 2))
```

Answer 2: C

Problem 3: Which of the following evaluates to a value (A, B, both, or neither)?

A. `'(2 2 2)`
B. `(2 2 2)`

Answer 3: neither

Problem 4: The following expression contains:

A. a string
B. quoted material
C. both
D. neither

```
(= "4" (+ 1 3))
```

Answer 4: A

Problem 5: Write a function `add-up` that takes two numbers returns their sum.

Answer 5: Please put your answer in `ps1-lastname-firstname.clj`.

Problem 6: Write a function `is-it-four?` that returns `true` when given the number `4`, and returns `false` otherwise.

Note: Don't forget the question mark in the function name! This is a convention in Clojure for the names of predicate functions (functions that return a boolean value—`true` or `false`). Also, an incorrectly named function won't be seen by the grader script!

Answer 6: Please put your answer in `ps1-lastname-firstname.clj`.

Problem 7: Fill in the blank, so the following expression evaluates to `true`:

```
(= (quote ___) 'platypus)
```

Answer 7: Please put your answer in `ps1-lastname-firstname.clj`.

Problem 8: Define a function `func` and an expression `expr` such that the following evaluates to `true`.

```
(= 3 (apply func expr))
```

Hint: be sure you understand what kinds of arguments `apply` expects.

Answer 8: Please put your answer in `ps1-lastname-firstname.clj`.

Problem 9: The built-in function `type` is useful for checking what kind of object an expression evaluates to.[1] Write a function `both-same-type?` that takes two arguments, and returns `true` when they both have the same type, and `false` otherwise.

Answer 9: Please put your answer in `ps1-lastname-firstname.clj`.

Problem 10: Write a function `list-longer-than?` which takes two arguments: an integer `n`, and a list `lst` and returns `true` if `lst` has more than `n` elements, and `false` otherwise. For example, `(list-longer-than? 3 '(1 2 3))` should return false, and `(list-longer-than? 2 '(1 2 3))` should return true.

Hint: you may find built-in clojure function `count` useful.

Answer 10: Please put your answer in `ps1-lastname-firstname.clj`.

Problem 11: In linear algebra, if $\mathbf{x}, \mathbf{y}$ are two vectors each with n components, their dot product is $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i$. Write a function `dot-product` that takes two lists of numbers as arguments, and returns the dot product. So for example, if the list `x` is `'(0 2 4)` and the list `y` is `'(1 3 5)`, the expression `(dot-product x y)` should return `26` $= 0 \cdot 1 + 2 \cdot 3 + 4 \cdot 5$.

You may assume that the two input lists are of equal length and contain only numbers as elements.

Hint: you may find built-in clojure functions `apply` and `map` useful.

Answer 11: Please put your answer in `ps1-lastname-firstname.clj`.

---

[1] Though note that the names of the types that the function `type` returns are different for JVM Clojure and ClojureScript in the textbook. Don't worry about this.

Problem 12: In Clojure (like other functional programming languages) functions and variables are treated identically. This means a function may easily take another function as an argument, and/or return a function.[2] Write a function `swap-arg-order` which takes a *function* (of two arguments) as an argument returns *another function* that does the same thing, but expects its two arguments in the opposite order.

That is, for example

- given the division function `/` which divides the first argument by the second (so `(/ 3 6)` returns the number `1/2`), the expression `((swap-arg-order /) 3 6)` should return `2`
- given the function `list-longer-than?` from above, the expression `((swap-arg-order list-longer-than?) '(1 2 3) 2)` should return `true`

Answer 12: Please put your answer in `ps1-lastname-firstname.clj`.

Problem 13: Define a higher order function `g` so the following expression evaluates to `true`:

```
(= 100 (g
      (fn [n] (* n n))))
```

Answer 13: Please put your answer in `ps1-lastname-firstname.clj`.

---

[2] Functions which take other functions as arguments are called 'higher order functions'. Built-in functions `map` and `apply` are higher order functions.