Lab 2 – Conditional and Logical Expressions

1. In class this week, we wrote the function `(divides? smaller larger)`, which determines if the *smaller* number divides exactly into the *larger* number, with a remainder of zero. Rewrite this definition using a `cond` expression instead of an `if` expression. Make sure to test your new version on several input values.

```
(define divides?
  (lambda (smaller larger)
    (if (= (remainder larger smaller) 0)
        #t
        #f)))
```

2. We also wrote the function `(classify n)`, which returns the symbol `negative`, `positive`, or `zero` depending on the input value *n*. Rewrite this function using a `cond` expression instead of nested `if` expressions. Make sure to test your new version on several input values.

```
(define classify
  (lambda (n)
    (if (< n 0)
        'negative
        (if (> n 0)
            'positive
            'zero))))
```

3. Rewrite the function `both-zero?` using a single `cond` expression instead of nested `if` expressions. For this exercise, it may help to modify one or both of the if-conditions using the `not` function.

```
(define both-zero?
  (lambda (a b)
    (if (= a 0)
        (if (= b 0)
            #t
            #f)
        #f)))
```

4. Write the function `(teen? age)` that takes an age in years as input and returns `#t` if the age is between 13 and 19, inclusive. Hint: use an `and` expression in your definition. For example:

```
(teen? 13) → #t
(teen? 14) → #t
(teen? 21) → #f
```

5. Write the function `(exactly-one-zero? a b)`, which takes two numbers as input and returns `#t` if exactly one of the numbers is zero, or `#f` otherwise. For this problem, you are not allowed to use multiplication or addition in your solution. For example:

```
(exactly-one-zero? 0 0) → #f
(exactly-one-zero? 0 3) → #t
(exactly-one-zero? 2 0) → #t
(exactly-one-zero? 1 2) → #f
```

6. Write the function `(equal-dates? month1 day1 year1 month2 day2 year2)`, which takes two dates as input, each in the form of three numbers representing the month, day, and year, and returns `#t` if the dates are the same. Hint: use an `and` expression in your definition. Test your function on the following inputs, and make up a few other tests of your own:

```
(equal-dates? 2 12 2021 2 12 2021) → #t
(equal-dates? 7 4 1776 7 4 1889) → #f
(equal-dates? 10 31 1999 12 31 1999) → #f
```

7. Write the function `(different-dates? month1 day1 year1 month2 day2 year2)`, which returns `#t` if the dates are different. If you wish, you may use your `equal-dates?` function as a helper. Examples:

```
(different-dates? 2 12 2021 2 12 2021) → #f
(different-dates? 7 4 1776 7 4 1889) → #t
(different-dates? 10 31 1999 12 31 1999) → #t
```

8. Write the function `(earlier-date month1 day1 year1 month2 day2 year2)`, which takes two dates as input, each in the form of three numbers representing the month, day, and year, and returns one of three possible symbols: if the first date occurs earlier than the second date, the symbol `first` is returned; if the second date occurs earlier than the first, the symbol `second` is returned. Otherwise both dates are the same, so the symbol `neither` is returned. You should use a `cond` expression in your definition. Test your function on all of the examples below, and make up a few other tests of your own:

```
(earlier-date 1 1 1963 1 1 2000) → first
(earlier-date 1 1 1964 12 31 1963) → second
(earlier-date 6 15 1999 10 15 1999) → first
(earlier-date 6 30 1999 6 1 1999) → second
(earlier-date 6 30 1999 6 30 1999) → neither
```

9. Write the function `(has-30-days? month)`, which takes a month number from 1 to 12 as input, and returns `#t` if the month has exactly 30 days, or `#f` otherwise. The months with 30 days are April, June, September, and November. Your definition should use an `or` expression.

```
(has-30-days? 4) → #t
(has-30-days? 1) → #f
```

10. Write the function `(leap-year? year)`, which takes a year as input and returns `#t` or `#f`, indicating whether the year is a leap year, according to the rules below. Test your function on all of the examples shown. Hint: define your function as a `cond` expression, and use `divides?` as a helper function.

- If the year is divisible by 400, the year <u>is</u> a leap year. Examples: 1600, 2000
- Otherwise, if the year is divisible by 100, the year <u>is not</u> a leap year. Examples: 1700, 1800, 1900
- Otherwise, if the year is divisible by 4, the year <u>is</u> a leap year. Examples: 2004, 2020
- Otherwise, the year <u>is not</u> a leap year. Examples: 2021, 2022

11. Write the function `(days-in-month month year)`, which takes a month number and year as input and returns the exact number of days in that month, taking into account leap years. You should use your functions `has-30-days?` and `leap-year?` as helper functions. Test your function on the examples below, and make up a few more tests of your own:

```
(days-in-month 10 2021) → 31
(days-in-month 2 2021) → 28
(days-in-month 2 2020) → 29
(days-in-month 11 1950) → 30
```

12. Write the function `(valid-date? month day year)`, which takes a date represented as three numbers as input, and returns `#t` if the date is valid, or `#f` otherwise. To be considered a valid date, *month* must be in the range 1 to 12, *year* must be greater than 0, and *day* must be in the range 1 to the number of days in the month, taking into account leap years. Test your function on all of the dates below, and make up a few more tests of your own:

Valid dates: 2 29 2016, 2 29 2000, 2 28 1900, 1 30 2016, 10 5 1963, 7 4 1776
Invalid dates: 2 29 2018, 2 29 1900, 2 30 2021, 13 5 1963, 10 32 1963, 11 31 2000, 7 -4 0000