

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
 Department of Electrical Engineering and Computer Science
 6.090—Building Programming Experience
 IAP 2006

Problem Set 3

Non-collaborative problems

Please attempt these problems by yourself. If you get stuck and need assistance, talk to the course staff.

1. Write a procedure, `with-tax`, that given a dollar amount returns the amount with a 5

```
(with-tax 5.00)
;Value: 5.25
(with-tax 259.99)
;Value: 272.9895
```

Remember that you can't actually use the `$` in your code or it won't work.

2. Write a procedure, `num-digits`, that takes in a number and returns the number of decimal digits in the number. For example:

```
(num-digits 5)
;Value: 1
(num-digits 21)
;Value: 2
(num-digits 3987423)
;Value: 7
```

You will find `quotient` useful, as it works like `/`, except it throws away anything after the decimal point (ie integer division). For example:

```
(quotient 4 2)
;Value: 2
(quotient 5 2)
;Value: 2
```

First write out a plan (base case and recursive case), then implement the procedure in scheme. Finally, test your procedure to verify that it works.

3. Write a procedure, `add-digit`, which takes a number and a digit and returns a new number with the digit as the least significant digit and the rest of the digits like the original number. You may assume that the second input is always less than 10 (ie a digit from 0-9). For example:

```
(add-digit 123 4)
;Value: 1234
```

```
(add-digit 7 1)
;Value: 71
```

Reasonable Problems

You may collaborate on these problems.

1. Implement `divisible?`, which returns true if the first argument is divisible by the second. Using `remainder` is a good idea.

```
(divisible? 6 3)
;Value: #t
(divisible? 37 4)
;Value: #f
```

2. Implement `slow-add`, which adds two non-negative numbers together. The catch is that you may not use any arithmetic procedures other than `inc` (increment by 1), `dec` (decrement by 1), and `zero?` (is the number 0):

```
(inc 5)
;Value: 6
(dec 3)
;Value: 2
(zero? 0)
;Value: #t

(slow-add 5 3)
;Value: 8
```

Sticky Problems

These problems are sticky. Make sure you have a good plan before you go for implementation.

1. Write a procedure to find the smallest factor of a number. As a suggestion, have the procedure take in both the number `n`, and a factor `f`. The procedure should first test to see if `f` is a factor, then check larger factors if `f` is not a factor. Some numbers may not have any factors in this range, if so, the procedure should return false. You should use `divisible?` from the previous problem.

```
(smallest-factor 8 3)
;Value: 4 (4 divides 8 and is between 3 and 8)
(smallest-factor 12 7)
;Value: #f (12 has no factors larger than 7)
```

2. Building on the previous exercise, write `prime?`, which returns true if the number is prime. Remember that a prime number has no factors other than 1 and itself.

```
(prime? 4)
;Value: #f
(prime? 17)
;Value: #t
(prime? 569)
;Value: #t
```