National University of Singapore
School of Computing
CS1010S: Programming Methodology

Extra Practice 9

All the best for your finals!

Code Tracing

(a)
```python
# CS1010S AY17/18 Sem 2 Finals
def foo(x):
    return lambda y: bar(x) if y % 2 else x
def bar(y):
    return lambda x: foo(x) if y % 2 else y

print(foo(2)(3)(4))
```

(b)
```python
# CS1010S AY17/18 Sem 2 Finals
a = [0, 1, 2]
a.append(a)
b = [a[0] + a[1], a[1:2], a[3][3][2]]
print(b)
```

(c)
```python
# CS1010S AY17/18 Sem 1 Finals
s = 'Lollapalooza'
d = {}
for i in range(len(s)):
    d[s[i % 5]] = s[i]
print(d)
```

(d)
```python
# CS1010S AY17/18 Sem 2 Finals
def force(x):
    try:
        return int(x)
    except ValueError:
        return float(x)
    except Exception:
        return "NaN"

print(force("100"))
print(force("1.0"))
print(force("abc"))
```

(e)
```
# CS1010S AY19/20 Sem 1 Finals
def foo(x):
    def baz(y):
        return lambda z: (x, y)[z]
    return lambda x: baz(x)

print(foo(-1)(0)(1))
```

(f)
```
# CS1010S AY19/20 Sem 2 Finals
lst = [[1], [2, 2], [3, 3, 3]]

def f(lst):
    for i in lst.copy():
        if len(i) < 2:
            i.append(1)
        if sum(i) < 5:
            i.pop()
        else:
            lst.extend(i)
        print(lst)
    return lst

print(lst is f(lst))
print(lst)
```

(g)
```
# CS1010S AY20/21 Sem 1 Finals
def wow(n):
    print(n)
    return lambda m: n + m

def twice(t):
    print('yes')
    return lambda x: t(t(x))

once = twice(twice)(wow(2))
print(once(1))
```

(h)
```
# CS1010FC AY14/15 Special Term I Finals
a = {1: 2, 2: 4, 3: 6, 4: 7}
for k in a:
    if k % 2 == 1:
        del a[k]
print(a)
```

(i)
```
# CS1010X AY16/17 Special Term I Finals
a = {(1, 2): 3, (3, 4): 5}
for k, v in a.items():
    a[[v, k[0]]] = k[1]
b = list(a.values())
b.sort(reverse = True)
print(b)
```

(j)
```
# CS1010S AY17/18 Sem 2 Midterm
def foo(y):
    return lambda x: x(x(y))
def bar(x):
    return lambda y: x(y)
print((bar)(bar)(foo)(2)(lambda x: x + 1))
```

(k)
```
# CS1010S AY19/20 Sem 1 Midterm
def foo(x):
    return x(lambda a: a + 1)
def kung(x):
    return foo(lambda a: a(x))
print(kung(foo)(9000))
```

Robbing a House

A thief wants to rob a series of houses, but unfortunately, the houses are somehow linked by an alarm system such that he cannot rob two houses side-by-side. Given a list of houses containing the amount of money (in millions) in each house, return the maximum amount the thief can earn.

Sample Tests:

```
>>> rob([3, 1, 4, 10, 2, 2, 9, 8])
23  # 3 + 10 + 2 + 8 = 23
>>> rob([1, 100, 99, 1, 3])
103 # 1 + 99 + 3 = 103
```

Number Sum Mania

(CS1010FC AY14/15 Special Term I Finals)

A positive integer $n \geq 2$ can be expressed as the sum of a number of positive integers smaller than n. For example,

$$
\begin{aligned}
2 &= 1 + 1 \\
3 &= 1 + 2 \\
&= 1 + 1 + 1 \\
4 &= 1 + 3 \\
&= 2 + 2 \\
&= 1 + 1 + 2 \\
&= 1 + 1 + 1 + 1 \\
5 &= 1 + 4 \\
&= 1 + 1 + 3 \\
&= 2 + 3 \\
&= 1 + 2 + 2 \\
&= 1 + 1 + 1 + 2 \\
&= 1 + 1 + 1 + 1 + 1
\end{aligned}
$$

The function **`num_sum`** returns the number of ways that an integer can be expressed as the sum of a number of positive integers. From the above examples, it should be clear that:

```
num_sum(2) = 1
num_sum(3) = 2
num_sum(4) = 4
num_sum(5) = 6
```

(a) Write the function **`num_sum`**. **BIG HINT:** **`num_sum`** is extremely similar to **`count_change`** which was discussed in lecture.

(b) Write the function **`sum_set`** that will return a list of the lists of possible number combinations for the integer sums. **Hint:** Think about how to modify the answer for Part (a).
Sample Execution:

```
>>> sum_set(2)
[[1, 1]]

>>> sum_set(3)
[[1, 1, 1], [2, 1]]

>>> sum_set(4)
[[1, 1, 1, 1], [2, 1, 1], [2, 2], [3, 1]]

>>> sum_set(5)
[[1, 1, 1, 1, 1], [2, 1, 1, 1], [2, 2, 1], [3, 1, 1], [3, 2],
[4, 1]]
```

```
>>> sum_set(6)
[[1, 1, 1, 1, 1, 1], [2, 1, 1, 1, 1], [2, 2, 1, 1], [2, 2, 2],
[3, 1, 1, 1], [3, 2, 1], [3, 3], [4, 1, 1], [4, 2], [5, 1]]
```

(c) Write the function **`sum_set_product`** that will return a list of the products of the integer sums produced by **`sum_set`**, i.e. multiply together the components of each integer sum. You can assume that you have access to the function **`sum_set`** even if you cannot do Part (b).
Sample Execution:

```
>>> sum_set_product(2) # 1x1
[1]

>>> sum_set_product(3) # 1x1x1 and 2x1
[1, 2]

>>> sum_set_product(4)
[1, 2, 3, 4]

>>> sum_set_product(5) # Note that 4x1 = 2x2x1 so 5 elements, not 6
[1, 2, 3, 4, 6]

>>> sum_set_product(6)
[1, 2, 3, 4, 5, 6, 8, 9]
```

(d) Write the function **`has_prime_sum`** that will return **`True`** for an integer n if it can be expressed as a sum of **2** prime numbers, or **`False`** otherwise. Assume that you have access to the function **`is_prime`** that will return **`True`** if an integer is prime.
Sample Execution:

```
>>> has_prime_sum(2)
False

>>> has_prime_sum(3)
False

>>> has_prime_sum(4)    # 2+2
True

>>> has_prime_sum(5)    # 2+3
True

>>> has_prime_sum(6)    # 3+3
True

>>> has_prime_sum(11)   # Not possible!
False
```