National University of Singapore
School of Computing
CS1010S: Programming Methodology

Extra Practice 9 Solutions

All the best for your finals!

Code Tracing

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

# This means foo(x)(y) = bar(x) if y is odd
#                        x if y is even

def bar(y):
    return lambda x: foo(x) if y % 2 else y

# This means bar(y)(x) = foo(x) if y is odd
#                        y if y is even

print(foo(2)(3)(4))
# foo(2)(3)(4)
# = bar(2)(4) since 3 is odd
# = 2 since 2 is even
```

(b)
```
# 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) # [1, [1], 2]
```

Click here for full visualization.

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

# len(s) = 12
# i = 0 -> d = {'L': 'L'}
# i = 1 -> d = {'L': 'L', 'o': 'o'}
# i = 2 -> d = {'L': 'L', 'o': 'o', 'l': 'l'}
# i = 3 -> d = {'L': 'L', 'o': 'o', 'l': 'l'}
# i = 4 -> d = {'L': 'L', 'o': 'o', 'l': 'l', 'a': 'a'}
# i = 5 -> d = {'L': 'p', 'o': 'o', 'l': 'l', 'a': 'a'}
# i = 6 -> d = {'L': 'p', 'o': 'a', 'l': 'l', 'a': 'a'}
```

```
# i = 7 -> d = {'L': 'p', 'o': 'a', 'l': 'l', 'a': 'a'}
# i = 8 -> d = {'L': 'p', 'o': 'a', 'l': 'o', 'a': 'a'}
# i = 9 -> d = {'L': 'p', 'o': 'a', 'l': 'o', 'a': 'o'}
# i = 10 -> d = {'L': 'z', 'o': 'a', 'l': 'o', 'a': 'o'}
# i = 11 -> d = {'L': 'z', 'o': 'a', 'l': 'o', 'a': 'o'}
```

(d)
```
# 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")) # 100
print(force("1.0")) # 1.0
print(force("abc")) # ValueError: could not convert string to float: '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)

# foo(x) = lambda x: baz(x)
#        = lambda t: baz(t) -> dummy variable trick
# foo(x)(t) = baz(t) = lambda z: (x, t)[z]
# foo(x)(t)(z) = (x, t)[z]

print(foo(-1)(0)(1))
# foo(-1)(0)(1)
# = (-1, 0)[1] = 0
```

(f)
```
# CS1010S AY19/20 Sem 2 Finals
lst = [[1], [2, 2], [3, 3, 3]]
def f(lst):
    for i in lst.copy(): # will have 3 iterations
        if len(i) < 2:
            i.append(1)
        if sum(i) < 5:
            i.pop()
        else:
            lst.extend(i)
        print(lst)  # 1st iteration: [[1], [2, 2], [3, 3, 3]]
                    # 2nd iteration: [[1], [2], [3, 3, 3]]
                    # 3rd iteration: [[1], [2], [3, 3, 3], 3, 3, 3]
    return lst
print(lst is f(lst))     # True
print(lst)               # [[1], [2], [3, 3, 3], 3, 3, 3]
```

Click here for full visualization.

(g)
```python
# 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))
# We start from evaluating twice(twice).
# The moment twice is called the first time, it will print 'yes'.
# Then, twice(twice) will return lambda x: twice(twice(x)).
# After that, wow(2) is evaluated first, printing 2 and returning
# lambda m: m + 2.
# Now we are just to evaluate twice(twice(lambda m: m + 2)).
# Note that twice(lambda m: m + 2) will print another 'yes' and
# returns lambda m: m + 4.
# Then, twice(lambda m: m + 4) will print one more 'yes' and returns
# lambda m: m + 8, which is our final result.

# From beginning until now, we have printed 'yes', 2, 'yes', 'yes'.
print(once(1)) # finally this is (lambda m: m + 8)(1) = 9.
```

(h)
```python
# 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] # RuntimeError: dictionary changed size during iteration
print(a)
```

Click here for full visualization.

(i)
```python
# 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] # TypeError: unhashable type: 'list'
b = list(a.values())
b.sort(reverse = True)
print(b)
```

Click here for full visualization.

(j)
```python
# CS1010S AY17/18 Sem 2 Midterm
def foo(y):
    return lambda x: x(x(y))
# This means foo(y)(x) = x(x(y))

def bar(x):
    return lambda y: x(y)
# This means bar(x)(y) = x(y)
```

```
print((bar)(bar)(foo)(2)(lambda x: x + 1))
# (bar)(bar)(foo)(2)(lambda x: x + 1)
# = bar(foo)(2)(lambda x: x + 1)
# = foo(2)(lambda x: x + 1)
# = (lambda x: x + 1)((lambda x: x + 1)(2))
# = (lambda x: x + 1)(3) = 4
```

(k)
```
# CS1010S AY19/20 Sem 1 Midterm
def foo(x):
    return x(lambda a: a + 1) # rewrite (lambda a: a + 1) as add1

def kung(x):
    return foo(lambda a: a(x))

print(kung(foo)(9000))
# kung(foo)(9000)
# = foo(lambda a: a(foo))(900)           [using kung(x)]
# = (lambda a: a(foo))(add1)(9000)       [using foo(x)]
# = (add1(foo))(9000)                    [a is now replaced by add1]
# = TypeError                            [how can you add 1 to foo?]
```

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
```

Solution:

```
# DISCLAIMER:
# This question uses dynamic programmming, which may not be
# examinable on finals.
# Solution is taken entirely from the official solution.
def rob(nums):
    # helper function
    def calculate(values):
        if len(values) == 1:
            return values[0]

        # Use DP. This is the list to build
        lst = []
        # fill a zero to signify zero houses
        lst.append(0)
```

```python
        # value of the first house
        lst.append(values[0])
        for i in range(2, len(values) + 1):
            # given house i, there are 2 options:
            # 1) not chose this house and choose the previous
            #    house, or
            # 2) choose this house and add it to the optimal
            #    choice of choosing 2 houses back.
            lst.append(max(lst[i-1], lst[i-2] + values[i-1]))

        # the optimal value is in the last element
        return lst[-1]

    if not nums:
        return 0
    elif len(nums) == 1:
        return nums[0]
    else:
        # since its cirular I eliminate 1st element of the list
        # and then eliminate the last element of the list and
        # seperately calculate the max value I can get for each sub list
        # and then I take the maximum out of them
        return max(calculate(nums[1:]), calculate(nums[:len(nums)-1]))
```

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.
Solution:

```
def num_sum(n):
    def helper(n, k):
        if n == 0:
            return 1
        elif n < 0 or k == 0:
            return 0
        return helper(n, k-1) + helper(n-k, k)
    return helper(n, n-1)
```

(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]]
```

Solution:

```
def sum_set(n):
    result = []
    def helper(n, k, current):
        if n == 0:
            result.append(current)
        elif n < 0 or k == 0:
            return
        else:
            copy = list(current)
            copy.append(k)
            helper(n, k-1, current)
            helper(n-k, k, copy)
    helper(n, n-1, [])
    return result
```

(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]
```

Solution:

```
def sum_set_product(n):
    result = []
    for s in sum_set(n):
        product = 1
        for e in s:
            product *= e
        if product not in result:
            result.append(product)
    result.sort() # prettify - not necessary
    return result
```

(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
```

Solution:

```
def has_prime_sum(n):
    for i in range(2, n//2):
        if is_prime(i) and is_prime(n-i):
            return True
    return False
```

And that's the end of the Extra Practice series! Pat on the back!
Solution compiled by Russell Saerang.