Info1 Final Fall 2019

This document illustrates sample solutions for the different tasks. Please not that there is not only this one correct colution, many alternatives exist for how to approach the various tasks.

Task 1: General Questions

In [1]:

```
import sys
import unittest
from unittest import TestCase

def type_and_value(e):
    print("Type:  {}".format(type(e).__name__))
    print("Value: {}".format(e))
```

In [2]:

```
# a)
type_and_value(not ())
```

```
Type:  bool
Value: True
```

In [3]:

```
# b)
type_and_value(print("Hello World!"))
```

```
Hello World!
Type:  NoneType
Value: None
```

In [4]:

```
# c)
def input(q): # replace with fixed implementation for this notebook (mimicks default behavior)
    return "1.83"
type_and_value(input("How tall are you?"))
```

```
Type:  str
Value: 1.83
```

In [5]:

```
# d)
l = [1, 2, 3, 4]
l[1:3] = []
type_and_value(l)
```

```
Type:  list
Value: [1, 4]
```

In [6]:

```
# e)
def fun(l):
    if len(l) > 0:
        return fun(l[0])
    else:
        return 42
l = []
l.append(l)
try:
    type_and_value(fun(l))
except RecursionError:
    print("Uncontrolled recursion! (--> Exception / error)")
```

```
Uncontrolled recursion! (--> Exception / error)
```

In [7]:

```
# f)
class Person:
    def get_name(self):
        return self.name
try:
    p1 = Person("Adam")
    p2 = Person("Bran")
    type_and_value(p1.get_name())
except TypeError:
    print("The constructor does not have a parameter! (--> Exception / error)")
```

```
The constructor does not have a parameter! (--> Exception / error)
```

In [8]:

```
# g)
class X: pass
class Y(X): pass
g = 1 if isinstance(Y(), object) else 2.3
type_and_value(g)
```

```
Type:  int
Value: 1
```

In [9]:

```
# h)
a=2
b = 3.0
assert a < b
h = a*b
type_and_value(h)
```

```
Type:  float
Value: 6.0
```

In [10]:

```
# i)
try:
    x = None
    raise IndexError()
    x=1
except IndexError:
    x = 2.0
except:
    x = True
else:
    x = -1+0j
finally:
    x = "fin"
type_and_value(x)
```

```
Type:  str
Value: fin
```

In [11]:

```
# j)
try:
    try:
        x = 42 / 0
    finally:
        x = 1
    type_and_value(x)
except ZeroDivisionError:
    print("x is successfully set to {} in the finally block, but the error is never caught! (--> Exception, error
)".format(x))
```

```
x is successfully set to 1 in the finally block, but the error is never caught! (--> Exception, erro
r)
```

Task 2: Hailstone Sequence

In [12]:

```
def hailstone(n):
    result = [n]
    el = n
    while el != 1:
        if el % 2 == 0:
            el = el // 2
        else:
            el = el * 3 + 1
        result.append(el)
    return result
```

In [13]:

```
assert hailstone(1) == [1]
assert hailstone(3) == [3, 10, 5, 16, 8, 4, 2, 1]
assert hailstone(7) == [7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
```

Task 3: Recursion

In [14]:

```
class Node:
    def __init__(self, v, l=None, r=None):
        self.v = v
        self.l = l
        self.r = r

root = Node(10, \
   Node(5, Node(3), Node(18)), \
   Node(15, Node(8)))

def range_sum(node, lower, upper):
    sum = 0
    if lower <= node.v < upper:
        sum += node.v
    if node.l != None:
        sum += range_sum(node.l, lower, upper)
    if node.r != None:
        sum += range_sum(node.r, lower, upper)
    return sum

assert range_sum(Node(7), 1, 100) == 7
assert range_sum(Node(2, Node(3, Node(4))), 2, 4) == 5
assert range_sum(root, 4, 10) == 13 # see example above
```

Task 4: Object-Oriented Programming & Testing

In [15]:

```
import unittest
from unittest import  TestCase


class Item:
    def __init__(self, name, volume):
        self.name = name
        self.volume = volume

    # not required!! just for debugging...
    def __repr__(self):
        return "Item({}, {})".format(self.name, self.volume)


class Backpack:

    def __init__(self, max_volume):
        self.max_volume = max_volume
        self.content = []

    def pack(self, item):
        assert self.current_volume() + item.volume <= self.max_volume
        self.content.append(item)

    def unpack(self):
        if self.content:
            return self.content.pop()
        else:
            return None
```

```
            return None

    def current_volume(self):
        cur_volume = 0
        for item in self.content:
            cur_volume += item.volume
        return cur_volume

# example picked from exam (slightly adopted)
bp = Backpack(4.0)
bp.pack(Item("water bottle", 0.75))
bp.pack(Item("lighter", 0.005)) # exam had a typo in this line (0.05 vs. 0.005)
print("Current Volume: {}".format(bp.current_volume()))
i = bp.unpack()
print("Unpacked: {}".format(i))
try:
    bp.pack(Item("camping tent", 20.0))
except:
    print("Cannot pack a large tent!")



class BackpackTest(TestCase):

    def test_defaults(self):
        sut = Backpack(123)
        self.assertAlmostEqual(123.0, sut.max_volume)
        self.assertEqual([], sut.content)

    def test_no_volume_without_items(self):
        sut = Backpack(123)
        self.assertAlmostEqual(0.0, sut.current_volume())

    def test_adding_item_adds_volume(self):
        sut = Backpack(123)
        sut.pack(Item("Pillow", 10))
        self.assertAlmostEqual(10.0, sut.current_volume())

    def test_adding__large_item_fails(self):
        sut = Backpack(8)
        with self.assertRaises(AssertionError):
            sut.pack(Item("Pillow", 10))

    def test_added_item_can_be_retrieved(self):
        i = Item("Pillow", 10)
        sut = Backpack(123)
        sut.pack(i)
        self.assertIs(i, sut.unpack())

    def test_retrieved_items_get_removed(self):
        sut = Backpack(123)
        sut.pack(Item("Pillow", 10))
        sut.unpack()
        self.assertIsNone(sut.unpack())


unittest.main(argv=[''], verbosity=2, exit=False)
```

```
test_added_item_can_be_retrieved (__main__.BackpackTest) ... ok
test_adding__large_item_fails (__main__.BackpackTest) ... ok
test_adding_item_adds_volume (__main__.BackpackTest) ... ok
test_defaults (__main__.BackpackTest) ... ok
test_no_volume_without_items (__main__.BackpackTest) ... ok
test_retrieved_items_get_removed (__main__.BackpackTest) ...

Current Volume: 0.755
Unpacked: Item(lighter, 0.005)
Cannot pack a large tent!

ok

----------------------------------------------------------------------
Ran 6 tests in 0.007s

OK
```

Out[15]:

```
<unittest.main.TestProgram at 0x104201cc0>
```

Task 5: Inheritance & Composition

In [16]:

```
from abc import ABC, abstractmethod

class TableSerializer(ABC):
    def to_string(self, table):
        res = ""
        for r_idx, row in enumerate(table):

            # new line from second entry onwards
            if r_idx != 0:
                res += "\n"

            for c_idx, col in enumerate(row):
                # no separator on first column
                if c_idx != 0:
                    res += self._get_delimiter()

                # distinguish the three cases
                if type(col) is str:
                    res += self._frmt_str(col)
                elif type(col) is int:
                    res += self._frmt_int(col)
                else:
                    res += self._frmt_float(col)

        return res


    @abstractmethod
    def _get_delimiter(self):
        pass

    @abstractmethod
    def _frmt_str(self, s):
        pass

    @abstractmethod
    def _frmt_int(self, i):
        pass

    @abstractmethod
    def _frmt_float(self, f):
        pass

class CsvSerializer(TableSerializer):
    def _get_delimiter(self):
        return ","
    def _frmt_str(self, s):
        return "\"{}\"".format(s)
    def _frmt_int(self, i):
        return "{}".format(i)
    def _frmt_float(self, f):
        return "{}".format(f)

table = (("Name","Age","Size"), ("Hans",53,1.78), ("Frieda",27,1.63))
res = CsvSerializer().to_string(table)

print("## Input Table:")
print(table)
print("## Resulting CSV:")
print(res)
```

```
## Input Table:
(('Name', 'Age', 'Size'), ('Hans', 53, 1.78), ('Frieda', 27, 1.63))
## Resulting CSV:
"Name","Age","Size"
"Hans",53,1.78
"Frieda",27,1.63
```

Task 6: Working With Modules

In [17]:

```
# imports do not work in this Jupyter notebook
use_mock_methods = True
if use_mock_methods:
    # define the methods with hard-coded values
    def get_current_position(): return "0.123,0.345"
    def find_train_stations(pos):
        assert isinstance(pos, tuple)
        return [("Bahnhof Oerlikon", (0.123, 0.345))]
else:
    # correct solution for exam
    from navigation import find_train_stations, get_current_position

def find_next_station():
    # request position
    pos = get_current_position()
    # split string
    pos = pos.split(",")
    # convert values to floats and store them in tuple
    pos = (float(pos[0]), float(pos[0]))
    # find stations that are close by
    stations = find_train_stations(pos)
    if stations:
        # access closest station
        correct_station = stations[0]
        # access its name
        station_name = correct_station[0]
        return station_name
    # handle case, in which no station is close by
    return None

print(find_next_station())
```

```
Bahnhof Oerlikon
```