Informatik 1 2018: Solution of Final

Task 1: General Questions

a)

In [1]:

```
def details(exprStr):
    expr = eval(exprStr)
    print ("expr:\t{}".format(exprStr))
    print ("value:\t{}".format(expr))
    print ("type:\t{}".format(type(expr).__name__))

def error():
    print("value:\terror")
    print("type:\tNoneType")

details("5/2 + 3")
```

```
expr:   5/2 + 3
value:  5.5
type:   float
```

b)

In [2]:

```
b1 = "13579"
b2 = "02468"
details("b1[5:] + b2[-5]")
```

```
expr:   b1[5:] + b2[-5]
value:  0
type:   str
```

c)

In [3]:

```
details("(lambda x: x%2 == 0)(2)")
```

```
expr:   (lambda x: x%2 == 0)(2)
value:  True
type:   bool
```

d)

In [4]:

```
d = [[[1, 1], [2, 2]], [[3, 3], [4, 4]]]
try:
    details("d[0][2]")
except:
    error()
```

```
value:  error
type:   NoneType
```

e)

In [5]:

```
class X: pass
class Y(X): pass
class Z(Y): pass
if isinstance(Z(), X):
    e = 1
else:
    e = 2.3

details("e")
```

```
expr:   e
value:  1
type:   int
```

f)

In [6]:

```
f = sorted({ 'a':1, 'b':2, 'c':3 }.items())
details("f[0]")
```

```
expr:   f[0]
value:  ('a', 1)
type:   tuple
```

g)

In [7]:

```
def g():
    return False
details('"x" if not g else {}')
```

```
expr:   "x" if not g else {}
value:  {}
type:   dict
```

h)

In [8]:

```
def addition(arr):
    s = 0
    for el in arr:
        if el % 2 == 0:
            s += el
    return s
details("addition([1, 2, 3, 4])")
```

```
expr:   addition([1, 2, 3, 4])
value:  6
type:   int
```

i)

In [9]:

```
class Animal:
    def talk(self):
        return "Moo!"

class Dog(Animal):
    pass

dog = Dog()
details("dog.talk()")
```

```
expr:   dog.talk()
value:  Moo!
type:   str
```

j)

In [10]:

```
class Employee:
    id = 0
    def __init__(self, name):
        self.name = name
        self.id = Employee.id
        Employee.id += 1

emp = Employee("Marc")
details("emp.id")
```

```
expr:   emp.id
value:  0
type:   int
```

Task 2: Functions

a) Implementation of split

In [11]:

```
def split(text):
    words = []
    cur = ""
    for c in text:
        if c == " " and not cur == "":
            words.append(cur)
            cur = ""
        else:
            cur += c
    if not cur == "":
        words.append(cur)
    return words

assert split("") == []
assert split("aaa") == ["aaa"]
assert split("a bbb cc") == ["a", "bbb", "cc"]
```

b) Reverse Index

In [12]:

```
def rev_idx(words):
    d = {}
    idx = 0
    for c in words:
        c = c.lower()
        if not c in d.keys():
            d[c] = []
        d[c].append(idx)
        idx += 1
    return d

assert rev_idx([]) == {}
assert rev_idx(["a","b"]) == {"a": [0], "b": [1]}
assert rev_idx(["a","B","A","aa"]) == {"a": [0, 2], "aa": [3], "b": [1]}
```

Task 3: Recursion

a) Product of two numbers

In [13]:

```
def prod(x, y):
    if x == 1:
        return y
    return y + prod(x-1, y)

assert prod(2, 0) == 0
assert prod(5, 2) == 10
```

b) Reverse List

In [23]:

```
def reverse(l):
    if len(l) < 2:
        return l
    return [l[-1]] + reverse(l[:-1])

assert reverse([]) == []
assert reverse([2]) == [2]
assert reverse([2, 6, 5]) == [5, 6, 2]
```

Task 4: Object-Oriented Programming & Testing

a) Accounting

In [15]:

```
class BankAccount:
    def __init__(self, limit):
        assert limit >= 0
        self.__balance = 0
        self.__limit = limit

    def balance(self):
        return self.__balance

    def available(self):
        return self.__balance + self.__limit

    def deposit(self, amount):
        assert amount > 0
        self.__balance += amount

    def withdraw(self, amount):
        assert amount > 0
        assert amount <= self.available()
        self.__balance -= amount
```

b) Black-Box Unit Tests

In [24]:

```
import unittest
from unittest import TestCase

class BankAccountTest(TestCase):

    def test_balance(self):
        acc = BankAccount(100)
        self.assertEqual(acc.balance(), 0)

    def test_available(self):
        acc = BankAccount(12)
        self.assertEqual(acc.available(), 12)

    def test_deposit(self):
        acc = BankAccount(12)
        acc.deposit(23)
        self.assertEqual(acc.balance(), 23)

    def test_withdraw(self):
        acc = BankAccount(12)
        acc.withdraw(3)
        self.assertEqual(acc.balance(), -3)

    def test__fail_negative_limit(self):
        with self.assertRaises(AssertionError):
            BankAccount(-1)

    def test__fail_negative_deposit(self):
        acc = BankAccount(0)
        with self.assertRaises(AssertionError):
            acc.deposit(-1)

    def test__fail_negative_withdraw(self):
        acc = BankAccount(0)
        with self.assertRaises(AssertionError):
            acc.withdraw(-1)

    def test__fail_too_big_withdraw(self):
        acc = BankAccount(10)
        with self.assertRaises(AssertionError):
            acc.withdraw(11)


unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
........
----------------------------------------------------------------------
Ran 8 tests in 0.010s

OK
```

Out[24]:

```
<unittest.main.TestProgram at 0x10dabf5f8>
```

Task 5: Inheritance & Composition

a) Implement the abstract base class FileSystemItem

In [17]:

```
from abc import ABC
from abc import abstractmethod

class FileSystemItem(ABC):
    @abstractmethod
    def size(self):
        pass
```

b) Implement File according to the specification.

In [18]:

```
class File(FileSystemItem):
    def __init__(self, size):
        self.__size = size
    def size(self):
        return self.__size
```

c) Implement Folder according to the specification.

In [19]:

```
class Folder(FileSystemItem):
    def __init__(self, children):
        self.__children = children
    def size(self):
        sum = 0
        for c in self.__children:
            sum += c.size()
        return sum

assert File(1).size() == 1
assert Folder([]).size() == 0
assert Folder([File(2), File(3)]).size() == 5
assert Folder([File(4), Folder([File(5)])]).size() == 9
```

Task 6: Working With Modules

In [20]:

```
# content of file "stats.py"
def get_system_stats():
    return {
        "cpu_temp": 329.4,
        "fan_speed": 1234,
        # ...
    }

# content of file "TempReader.py"
# required: from stats import get_system_stats

class TempReader:
    def __kelvin(self):
        return get_system_stats()["cpu_temp"]
    def celsius(self):
        c = self.__kelvin() - 273.15
        return "{}C".format(round(c,1))
    def fahrenheit(self):
        f = 1.8 * self.__kelvin() - 459.67
        return "{}F".format(round(f,1))

# expected behavior with example output
tr = TempReader()
assert tr.celsius() == "56.2C"
assert tr.fahrenheit()== "133.2F"
```