Informatik 1 2018: Solution of Midterm

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")

try:
    details(1.2 + 3 + "5")
except:
    error()
```

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

b)

In [2]:

```
i = [1, 2, 3]
i + [4,5]
i.append([6, 7])
i.extend([8, 9])
details("i")
```

```
expr:   i
value:  [1, 2, 3, [6, 7], 8, 9]
type:   list
```

c)

In [3]:

```
b1 = "4321"
b2 = "12343"
b1[3:4] + b2
```

Out[3]:

```
'112343'
```

d)

In [4]:

```
details("False or False and True or False")
```

```
expr:   False or False and True or False
value:  False
type:   bool
```

e)

In [5]:

```
tmp = (lambda x: x//2, 13)
details("tmp[0](tmp[1])")
```

```
expr:   tmp[0](tmp[1])
value:  6
type:   int
```

f)

In [6]:

```
def e():
    for i in range(0, 10, 2):
        if i == 10:
            return i
details("e()")
```

```
expr:   e()
value:  None
type:   NoneType
```

g)

In [7]:

```
try:
    name = "John doe"
    name[5] = "D"
    details("name")
except:
    error()
```

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

h)

In [8]:

```
h = {'a': 1, "b": 2.3}
tmp = None
for x in h:
    tmp = x
    break
details("tmp")
```

```
expr:   tmp
value:  a
type:   str
```

i)

In [9]:

```
i=0
while i <= 100:
    i+=1
details("i")
```

```
expr:   i
value:  101
type:   int
```

j)

In [10]:

```
j=0
def fun(n):
    j = n
fun(3)
details("j")
```

```
expr:   j
value:  0
type:   int
```

Task 2: Control Flow and Data Structures

a) Control Flow

In [11]:

```
def get_age_desc(age):
    if age < 3:
        return "toddler"
    if age >= 65:
        return "senior"
    if age >= 18:
        return "adult"
    return "child"

assert get_age_desc(1) == "toddler"
assert get_age_desc(3) == "child"
assert get_age_desc(14) == "child"
assert get_age_desc(18) == "adult"
assert get_age_desc(65) == "senior"
```

b) Iteration

In [12]:

```
def compute_max_diff(numbers):
    if len(numbers) < 2:
        return 0

    max = 0
    last= numbers[0]
    for cur in numbers[1:]:
        diff = abs( last - cur )
        last = cur
        if diff > max:
            max = diff

    return max

assert compute_max_diff([]) == 0
assert compute_max_diff([1]) == 0
assert compute_max_diff([2, -1]) == 3
assert compute_max_diff([2, 7, 5, 14, 12, 14]) == 9
```

c) Even/Odd Sum

In [13]:

```
def sum_even_and_odd(numbers):
    sum = 0
    for n in numbers:
        if n%2:
            sum -= n
        else:
            sum += n
    return (len(numbers), sum)

assert sum_even_and_odd([]) == (0, 0)
assert sum_even_and_odd([2,4,7]) == (3, -1) # 3 values, 2+4-7=-1
```

d) Functional Programming

In [14]:

```
def app(l, op):
    res = []
    for e in l:
        res.append(op(e))
    return res

l = [1, 2, 3]
def double(x): return x * 2
assert app([], double) == []
assert app(l, double) == [2, 4, 6]
assert app(l, lambda x: x * 3) == [3, 6, 9]
assert l == [1, 2, 3]
```

e) Counting

In [15]:

```
def count_chars(s):
    counts = {}
    for c in s:
        if not c in counts.keys():
            counts[c] = 1
        else:
            counts[c] += 1
    return counts

assert count_chars("") == {}
assert count_chars("aA .") == {"a": 1, "A": 1, " ": 1, ".": 1}
assert count_chars("abbCaabb") == {"a": 3, "b": 4, "C": 1}
```

f) Function Calls

In [16]:

```
def count_occurrences(s, c):
    count = 0
    for e in s:
        if e == c:
            count += 1
    return count

assert count_occurrences("", "a") == 0
assert count_occurrences("x", "b") == 0
assert count_occurrences("abbccc", "c") == 3
```

Task 3: Recursive Functions

In [17]:

```
def nested_sum(l, depth=1):
    if type(l) == int:
        return l

    sum = 0
    for e in l:
        sum += nested_sum(e, depth+1)
    return depth * sum


assert nested_sum([]) == 0
assert nested_sum([1, 2, 3]) == 6 # 1*(1+2+3)
assert nested_sum([1, [2]]) == 5 # 1*(1+2*(2))
assert nested_sum([1, [2, [3]]]) == 23 # 1*(1+2*(2+3*(3)))
assert nested_sum([[1, 2], 3, [4, 5]]) == 27 # 1*(2*(1+2)+3+2*(4+5))
```

Task 4: Files

In [18]:

```
FILE_NAME = "INFO1_HS18_MIDTERM_T4.csv"

# prepare file:
content = """Hans,Info1,5.5
Hans,Economics,5
Petra,Info1,5.25
Petra,Economics,4.75
Petra,Math,6
Martin,Info1,5.75
"""

file = open(FILE_NAME, "w")
file.write(content)
file.close()

try:
    # task:
    def read(path):
        users = {}
        with open(FILE_NAME, "r") as f:
            for l in f.readlines():
                (user, course, grade) = l[:-1].split(",")
                if not user in users.keys():
                    users[user] = []
                users[user].append((course, float(grade)))

        return users

    assert read("results.csv") == {"Hans": [("Info1", 5.5), ("Economics", 5)],
                                   "Petra": [("Info1", 5.25), ("Economics", 4.75), ("Math", 6)],
                                   "Martin": [("Info1", 5.75)]}
finally:
    # teardown
    import os
    os.remove(FILE_NAME)
```