Info1 Midterm 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
def type_and_value(e):
    print("Type:  {}".format(type(e).__name__))
    print("Value: {}".format(e))
```

In [2]:

```
# a)
c = []
c.extend("abc")
type_and_value(c)
```

```
Type:  list
Value: ['a', 'b', 'c']
```

In [3]:

```
# b)
try:
    8 + 3.0 + "9.1"
except Exception as e:
    print(e)
```

```
unsupported operand type(s) for +: 'float' and 'str'
```

In [4]:

```
# c)
c1 = c2 = []
c1.append(1)
c2.append(2)
type_and_value(c1+c2)
```

```
Type:  list
Value: [1, 2, 1, 2]
```

In [5]:

```
# d)
d = [(1), (2,3), (4,5,6)]
type_and_value(d[0])
```

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

In [6]:

```
# e)
names = [
   ["adrian", "alina"],
   ["ben", "beat"],
   ["cornelius", "christoph"]
]
type_and_value(names[-1][-2][2:3])
```

```
Type:  str
Value: r
```

In [7]:

```
# f)
f = ["a", "x", "h"]
try:
    f[3] = "f"
except:
    print(sys.exc_info())
```

```
(<class 'IndexError'>, IndexError('list assignment index out of range'), <traceback object at 0x10a1
786c8>)
```

In [8]:

```
g = 0
for n in [1, 2, 3.0, 4.0]:
    if n%2:
        g += n
    break
type_and_value(g)
```

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

In [9]:

```
# h)
def fun_h():
    h=0
    for i in range(10):
        h += i
fun_h()

try:
    h
except:
    print(sys.exc_info())
```

```
(<class 'NameError'>, NameError("name 'h' is not defined"), <traceback object at 0x10a17e248>)
```

In [10]:

```
# i)
i = (lambda x: 0.5 * x**2, 4)
type_and_value(i[1])
```

```
Type:  int
Value: 4
```

In [11]:

```
# j)
a = [1,2,3]
b = [1,2,3]
c = b
if a == c:
    j = a is c
else:
    j = b == c
type_and_value(j)
```

```
Type:  bool
Value: False
```

Task 2: Iteration

In [12]:

```
def find_sum(l, target):
    for idx1, v1 in enumerate(l):
        for idx2, v2 in enumerate(l):
            if idx1 == idx2:
                continue
            if v1 + v2 == target:
                return (idx1, idx2)
    return None

assert find_sum([], 9) == None
assert find_sum([1], 2) == None
assert find_sum([1, 1], 2) == (0, 1)
assert find_sum([1, 7, 2, 11], 9) == (1, 2)
assert find_sum([1, 2, 3, 4], 5) == (0, 3)
```

Task 3: Working With Files

In [13]:

```
def read_fake():
    return ["Midway upon the journey of our life I found myself within a FOREST dark",
            "for the straightforward pathway had been lost",
            "",
            "Ah me How hard a thing it is to say what was this forest savage rough and",
            "stern which in the very thought renews the fear So bitter is it death is",
            "little more but of the good to treat, which there I found speak will I of"]

def read_real(path):
    with open(path, "r") as f:
        # find a way to get rid of '\n's
        return f.read().split("\n")

def count_keywords(path, keywords):
    res = {}
    for line in read_fake():
        for word in line.split(" "):
            word = word.lower()
            if word in keywords:
                n = res.get(word, 0) + 1
                res[word] = n
    return res

# file.txt contains the example above
assert count_keywords('file.txt', ['forest', 'the', 'found']) == {'the': 5, 'found': 2, 'forest': 2}
assert count_keywords('file.txt', ['black']) == {}
assert count_keywords('file.txt', []) == {}
```

Task 4: Binary Conversion

In [14]:

```
def to_binary(n):
    if n == 0:
        return "0"

    res = ""
    while n:
        res = ("1" if n%2 else "0") + res
        n=n//2

    return res

assert to_binary(0) == "0"
assert to_binary(1) == "1"
assert to_binary(2) == "10"
assert to_binary(19) == "10011"

def test(n):
    print("to_binary({}) = '{}'".format(n, to_binary(n)))

test(1)
test(2)
test(3)
test(19)
test(23187469187469187246918274698174691872469817246987126498712649873126498273)
```

```
to_binary(1) = '1'
to_binary(2) = '10'
to_binary(3) = '11'
to_binary(19) = '10011'
to_binary(23187469187469187246918274698174691872469817246987126498712649873126498273) = '1000110011101001111
111100110011010001010010010001000000100000100100000110110111101101011100011000011111011001111100000
110100110000110000001010100110101000010011100011010100000011001010110010110100111010000001111100001'
```

Task 5: Recursive Functions

In [15]:

```
def find_max(l):

    # anchor 1: handle numbers
    if type(l) == int: return l

    # anchor 2: handle empty lists
    if not l: return None

    mx = None
    for e in l:
        cur = find_max(e)
        if type(cur) == int:
            if mx == None or cur > mx:
                mx = cur
    return mx

assert find_max([]) == None
assert find_max([1, 12, -3]) == 12
assert find_max([2, [1]]) == 2
assert find_max([1, [-7, [13, [4]], [[[[5]]]] ]]) == 13
# additional cases
assert find_max([[]]) == None
assert find_max([[], 1]) == 1
assert find_max([2, []]) == 2
```