CSCI-B 505 Applied Algorithms (3 cr.) № 3

Dr. H. Kurban

Computer Science

School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

September 17, 2021

Contents

Problem 1: Variable Arguments

Python allows variable arguments to functions which makes coding easier. Assume we have two functions, one that sums a list of numbers and one that finds the product.

```
1  def sum1(x):
2      s = 0
3      if x:
4          for i in x:
5              s += i
6      return s
7
8  def prod1(x):
9      s = 1
10     if x:
11         for i in x:
12             s *= i
13     return s
14
15 if __name__ == '__main__':
16
17     xlst = [1,2,3,4,5]
18
19     print("sum " + str(sum1(xlst)))
20     print("product " + str(prod1(xlst)))
```

If we execute this program, we have:

```
1  PS D:\505> py .\Assignment_3\mult_arg_1.py
2  sum 15
3  product 120
4  PS D:\505>
```

It's clear from inspecting the code that the spirit of the two functions are nearly indentical. We'd like to write one function that finds either the sum or product given some key value. In Python this is denoted with * and **. Prefixing a single asterisk means the variable will be bound to a list. Prefixing a double asterisk means the variable will be a key value into the dictionary named as the bound variable. Examine the code below and its execution.

```
1  def f(*x, **y):
2
3      def s1(x):
4          s = 0
5          if x:
6              for i in x:
7                  s += i
8          return s
9
10     def p1(x):
11         s = 1
12         if x:
13             for i in x:
14                 s *= i
15         return s
16
17     if y["action"] == "sum":
18         return s1(*x)
19     elif y["action"] == "prod":
20         return p1(*x)
21     else:
22         return f"bad argument: {y}"
23
24 if __name__ == '__main__':
25
26     xlst = [1,2,3,4,5]
27
28     print(f(xlst, action = "sum"))
29     print(f(xlst, action = "prod"))
30     print(f(xlst, action = "reciprocal sum"))
```

when run gives:

```
1  15
2  120
3  bad argument: {'action': 'reciprocal sum'}
```

Line 1 uses *. The variable x creates a list object of all the values up to, but not including the expression key = “value”. We moved the two functions to be locally defined. Lines 17-22 show how we’re using the **. Lines 28-30 show the values sent. At run-time, the string action is made into a key in the dictionary y. The value assigned is its value. We check in code 17-22 if we invoke the appropriate functions given the values–since we haven’t defined the reciprocal sum, we let the user know it’s an error. You should experiment with removing the asterisk on lines 18 and 20 to observe the error and figure out why the asterisk is required.

Problem 2: Console I/O

There will be many times when you'll want to run your program from the console with various arguments. Python promotes using the argparse module, now standard for some time, as the means of writing console argument scripts. Please read this documentation before continuing: `https://docs.python.org/3/howto/argparse.html`

Write this script and execute it. Please read about the elements that make it work.

```
1  import argparse
2
3  parser = argparse.ArgumentParser()
4
5  parser.add_argument('-lst', nargs='+', default = ['1'], help="List of ↩
       numbers")
6  parser.add_argument('-op', default = 'sum', help = "sum, prod, rec")
7
8  args = parser.parse_args()
9
10 print(args.lst)
11 print(args.op)
```

when runs gives:

```
1  PS D:\505> py .\Assignment_3\goo.py -lst 1 2 3 4 5 -op sum
2  ['1', '2', '3', '4', '5']
3  sum
4  PS D:\505>
```

Problem 3: OO

Like most contemporary languages, Python is object-oriented though differing in some minor ways. This in no way is meant to teach you OO in Python; it's meant to be illustrative of significant elements to help you when you need to build a class.

Python has as a type, complex numbers that use j instead of the standard i. Here is a session:

```
>>> (1+3j) + (3+4j)
(4+7j)
>>> (1+3j) - (3+4j)
(-2-1j)
>>> (1+3j) * (3+4j)
(-9+13j)
>>> complex(1,3)
(1+3j)
```

The following code is an attempt to write complex numbers as a class like Python. In this class we add a method cadd which allows us to add an arbitrary number of complex numbers, but we must chain the method call. Python allows you to overload most symbols (+ for instance) which we can overload to make our complex addition similar to what would be expect as a native type. We decided to use i instead of the j that Python uses.

```
class complex_:
    def __init__(self, re=0, im=0):
        self.re = re
        self.im = im

    def get_re(self):
        return self.re

    def get_im(self):
        return self.im

    def __str__(self):
        g = lambda x:"+" if x >= 0 else ""
        return f"({self.re}{g(self.im)}{self.im}i)"

    def cadd(self, other):
        new_re = self.get_re() + other.get_re()
        new_im = self.get_im() + other.get_im()
        return complex_(new_re, new_im)

    def __add__(self, other):
        new_re = self.get_re() + other.get_re()
```

```
23          new_im = self.get_im() + other.get_im()
24          return complex_(new_re, new_im)
25
26  if __name__ == '__main__':
27
28      w = complex_(1,-3)
29      x = complex_(-1,3)
30      y = complex_(1,3)
31      z = complex_(-1,-3)
32      print(w)
33      print(x)
34      print(y)
35      print(z)
36
37      print(w.cadd(x).cadd(y).cadd(z))
38
39      print((1-3j) + (-1+3j) + (1+3j) + (-1-3j))
40
41      print(w + x + y + z)
```

When run we get:

```
1  (1-3i)
2  (-1+3i)
3  (1+3i)
4  (-1-3i)
5  (0+0i)
6  0j
7  (0+0i)
```

The __init__ works like a constructor. You don't have to worry about destructors. Python does not have real private or protected data unlike Java or C# for example. We overload print in lines 12-14 using __str__ which uses the string we construct when print is called with our object. We overload + in lines 21-24 using __add__. Observe that we must return an object of the same type. The last three lines of output are all equivalent to zero:

```
1  \addcontentsline{toc}{subsection}{Interactive Python Session}
2  >>> 0 == (0+0j)
3  True
4  >>>
```

Problem 4: Homework

1. Modify the second program to allow for the sum of reciprocals. For example, when given a list of numbers 1,2,3,4, 5 then the sum would be $1/1 + 1/2 + 1/3 + 1/4 + 1/5 = 2.28\overline{3}$. When adding this function, add it locally and use the other sum function and use an inline lambda function that maps x to $1/x$. You can assume zero is never in the list.

2. Using how to read in console information in problem 2, modify the first program so that it can be executed from the console using a list of numbers and the operation. You should *not* modify the original function–simply add script to read in data from the console.

3. Add multiplication to the Python class overload asterisk.

4. Perform an experimental analysis over the three algorithms, average1, average2, average3. In a log-log chart, visualize their running times as a function of the input size.

```
1  def average1(S):
2    #S:sequence
3    n = len(S)
4    my_average = [0] * n
5    for j in range(n):
6      total = 0
7      for i in range(j + 1):
8         total += S[i]
9      my_average[j] = total / (j+1)
10   return my_average
```

```
1  def average2(S):
2    #S:sequence
3    n = len(S)
4    my_average= [0] * n
5    for j in range(n):
6      my_average[j] = sum(S[0:j+1]) / (j+1)
7    return my_average
```

```
1  def average3(S):
2   #S:sequence
3    n = len(S)
4    my_average = [0] * n
5    total = 0
6    for j in range(n):
7      total += S[j]
8      my_average[j] = total / (j+1)
9    return my_average
```

5. In this question, you will work with the three algorithms, algorithm1, algorithm2, algorithm3, which check whether a given sequence is unique. Perform an experimental analysis to determine the largest value of input size such that the given algorithm runs in less than 45 seconds.

```
def algorithm1( S):
    #S:sequence
    for j in range(len(S)):
        for k in range(j+1, len(S)):
             if S[j] == S[k]:
                return False
    return True
```

```
def algorithm2(S):
   #S:sequence
    S = sorted(S)
    for j in range(1, len(S)):
        if S[j-1] == S[j]:
            return False
    return True
```

```
def algorithm3(S, start, stop):
    #slice S[start:stop],  S:sequence
    if stop - start <= 1: return True
    elif not algorithm3(S, start, stop-1): return False
    elif not algorithm3(S, start+1, stop): return False
    else: return S[start] != S[stop-1]
```

Directions

Please follow the syllabus guidelines in turning in your homework. Each question is worth 20 points. This homework is due Tuesday, Sep 28, 2021 10:00p.m. **OBSERVE THE TIME**. Absolutely no homework will be accepted after that time. All the work should be your own.