STA 141B
Exam

Oct 19, 2023

Name: Student ID:

The exam takes 50 minutes. It consists of three problems, covering the basics, memory handling as well as numpy and pandas. You are asked to explain the behavior and output of blocks of code. None of those return an error. Make sure to use proper terminology. In total, 36 points can be earned. No additional resources are allowed. Write your answers on the blank odd numbered pages.

Basics (15 points)

(a) Explain the code below. What does it return? What objects are x and y?

```
x = [1, 1, 2, 3, 5]
x.append([8])
y = x.pop()

print(y)
print(x)
```

(b) Explain the code below. What does it output? Use Python terminology.

```
x = [[x, x**2] for x in range(3)]
[e[1] for i, e in enumerate(x) if i is e[0]]
```

(c) Explain the code below and discuss its output. How, if at all, can the first entry of x be returned?

```
x = {'1': 1, 1: [1], 1.0: not 1}

print(x['1'])
print(x[1])
```

(d) What does the code below return? Explain?

```
try: x[range(4)] = 4
except: print("hip!")
try: x[[4]] = 4
except: print("hep!")
```

(e) What does the code below output? Explain.

```
def f(n):
    # Takes in a number x, returns the n-th power of n
    return lambda x, m = 2: x**n + m
g = [f(i) for i in range(2)]
h = [i(3) for i in g]

help(f)
print(h)
```

Solution:

(a) x is a list with entries [1, 1, 2, 3, 5] [+1]. The appended object in line two has been removed and returned in line three, both list methods modify the original list [+1]. y is the object that has been appended in line two, its a *list* with one entry [8] [+1].

(b) x is a list of lists. The first entry of each of its sublists correspond to the index of each sublists position. **enumerate** *zips* [+1] together the index of the outer list elements with the list elements as a tuple. Within the list comprehension [+1], i and e are assigned via *unpacking* [+1]. The if clause within the comprehension checks if the index i from the tuple corresponds to the first sublist element, which is also the index. Hence, this condition always returns True. Thus, the returned list is [0, 1, 2] [+1].

(c) x is a dictionary [+1]. It has unique keys. Those are instantiated by checking ==, not **is**. Thus, the second key-value pair is immediately overwritten [+1]. Consequently, the key '1' returns value 1, and the key 1 returns value False, because **not** converts 1 into a boolean type [+1]. Dictionaries are unordered, to there is no first entry of x [+1].

(d) The code returns 'hep'! [+1]. Range objects are *immutable* and valid keys, but [4] is a *mutable* list, thus not an invalid key [+1].

(e) The **help** function returns no *docstring* [+1], because such must be a string, not a comment [+1]. g contains a list of the two anonymous functions $x \mapsto x^0 + 2$ and $x \mapsto x + 2$, which are evaluated when assigning h, which consequently is [2, 3] [+1].

Memory (11 points)

(a) Discuss the relative value of the output of code below. If relevant, what is the influence of a?

```
import sys

a = sys.intern('why_do_pangolins_dream_of_quiche')
x = [100**100, 'hello', 'why_do_pangolins_dream_of_quiche']
y = (100**100, 'hello', a)
z = (100, 'hello', 'hello')

print(sys.getsizeof(x))
print(sys.getsizeof(y))
print(sys.getsizeof(z))
```

(b) Explain the code below and discuss the output. What kind of objects are y and z?

```
a = [i for i in y]
x = [100, 'hello', a]
y = x[2]
z = x[2][0]
x[2][0] = -1.0

print(y)
print(z)
```

(c) Explain the code below and discuss its output.

```
x = (x**2 for x in range(3))
y = [2*y for y in x]
z = sum(x)

print(y)
print(z)
```

Solution:

(a) The internalization of a is irrelevant [+1]. x is a list, thus it requires more size than both y and z [+1]. Both y and z contain pointers to three locations, thus their sizes are equal [+1].

(b) Due to *reference semantics* [+1], the list y [+1] has entries [−1.0, 'hello', 'why␣do...'] [+1]. z has always pointed to the integer object with value 100^{100}, and this has not changed [+1].

(c) x is a generator [+1]. It is evaluated when y is instantiated. This produces a list with entries [0, 2, 8] [+1]. This exhausts the generator [+1]. Thus, it is empty when z is instantiated, which consequently points to zero (if student claim this throws an error, this is also to be graded as correct) [+1].

Numpy and Pandas (10 points)

(a) What does the code below return? Explain.

```
import numpy as np

x = [1, 'hi']
y = np.array(x)

[a == b for a, b in zip(x, y)]
```

(b) What does the code below return? Explain.

```
x = np.matrix([[1, 2], [3, 4.0], [1, 2]])

print(x.shape)
print(x.size)
print(x.dtype)
```

(c) What does the code below return? Explain.

```
try: (x - 3) + x
except: print('hep!')
try: x + [[1, 2.0], [3, 4], [3, 4]]
except: print('hip!')
try: x[:,0] @ [1, 2.0]
except: print('hop!')
```

(d) What does the code below return? Explain.

```
import pandas as pd

x = pd.Series([1,2,3], index = [2, 1, 0])

print(x[2])
print(x.iloc[2])
print(x.loc[2])
```

(e) What does the code below return? Explain.

```
y = pd.Series([1,2,3])
try:
    z = y * x
    print(z)
except:
    print('hep!')
```

Solution:

(a) It returns [False, True], because while lists can hold objects of different type, objects of type numpy.array can not [+1] and according to promotion rules, all entries are convertet to strings [+1].

(b) x is a numpy.matrix, so that x.shape returns the tuple (3,2), x.size the number of entries 8 and x.dype the object type it contains, which are floats [+1].

(c) In the first **try**, all operations are performed element-wise. No error is thrown [+1]. The second **try** recasts the list to an numpy.array, no error is thrown [+1]. The last **try** slices through the matrix x and returns a (3,1) matrix. Its shape does not match the two-element list, so an error is thrown: 'hop! is printed [+1].

(d) x is a pandas.series object and thus references by index. Consequently, x[1] and x.loc[2] give the same result at index name 2 [+1], namely 1, whereas x.iloc[1] returns the index irrespective of number, which is thus 3 [+1].

(e) The code does not throw an error. Because pandas.Series objects benefit from *data alignment* [+1], the base index names are aligned with the values given in x. The result is 1∗3+2∗2+3∗1=10 [+1].