

## Practice Questions for CSCI-UA.0380 Final

1. Read the code in the right column. Answer questions about it in the left column.

<pre>class Foo(object):     def __init__(self, n):         self.x = n      def bar(self):         print('qux')      def __str__(self):         return self.x * 'foo'  class Corge(Foo):     def __init__(self, n):         super().__init__(n)      def bar(self):         print('grault')  c = Corge(3) c.bar() print(c)</pre>	<p>What's the output of the program on the left? If the output includes an error, show all output before the error and the type of error that occurs.</p>
<pre>def what_is_this(a, b):     if b == 0:         return 1     else:         return a * what_is_this(a, b - 1)  print(what_is_this(5, 0)) print(what_is_this(4, 3))</pre>	<p>What's the output of the program on the left? If the output includes an error, show all output before the error and the type of error that occurs.</p>
<pre>def create_initial_clusters(k):     clusters = []     for i in range(k):         clusters.append([])     return clusters  clusters = create_initial_clusters(3) print(clusters) for result in map(len, clusters):     print(result)</pre>	<p>What's the output of the program on the left? If the output includes an error, show all output before the error and the type of error that occurs.</p>
<p>Change the code above so that the body of <code>create_initial_clusters</code> is a list comprehension instead of a regular for loop. Write your list comprehension below:</p>	
<pre>def mystery(n):     nums = [0, 1]     for i in range(n):         val = nums[0]         del(nums[0])         nums.append(val + nums[0])         yield val  mysterious = mystery(7)</pre>	<p>What's the output of the program on the left? If the output includes an error, show all output before the error and the type of error that occurs.</p>

2. **Part 1** – K-means clustering: write one or two sentence answers to the following questions:

a) How are the initial centroids generated before any data points are put into clusters?

b) How are the subsequent centroids determined (once data points are distributed into clusters)?

c) Recalculating centroids and clustering data points based on new centroids is a process that is repeated applied. When should the repetition of the process above stop?

**Part 2** – Write a Euclidean distance function where that can handle an arbitrary number of data points. For example: `distance((5, 0, 1), (2, 12, 5))` # --> 12. Euclidean distance is found by taking summing the square of the difference of each feature between each data point, and taking the square root of the resulting sum. Here's an example where there are only two feature, x and y:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Your function must work for an arbitrary number of features (for example, x, y and z).

**Part 3** – Audio: write one or two sentence answers to the following questions:

a) Define frequency and sampling rate.

b) Why do the numbers that our pyaudio program create have to be within -1 and 1?

c) Why can sound be described as a signal?

3. **Part 1** – Classes and Objects: True or False

- a) \_\_\_\_\_ `super()` represents the parent class of the class that `super` is being used in
- b) \_\_\_\_\_ `__init__` is automatically called when you create a new object by calling a function that's named the same name as the class of the object you're creating
- c) \_\_\_\_\_ when a method is called on an object, the first argument that is passed in to the method is the object that the method was called on
- d) \_\_\_\_\_ when creating a class, you have to specify a method called `__str__`

**Part 2** – HTTP: write one or two sentence answers to the following questions:

a) If you type in <http://localhost:5000/hello/there> in your browser's URL bar, what is the 1<sup>st</sup> line of the http request that is made to the web server / your web application?

b) Name two HTTP methods.

c) Name three tools or applications that you can use to create an HTTP request.

**Part 3** – Sockets: True or False

- a) \_\_\_\_\_ when communicating using Python's socket module... data received from a client arrives in a bytes object, not a string
- b) \_\_\_\_\_ a socket represents a connection between a client and a server
- c) \_\_\_\_\_ you cannot create your own protocol when communicating via sockets; you must use a defined protocol, such as HTTP, FTP, SMTP, etc.

**Part 4** – Flask: write one or two sentence answers to the following questions:

a) What is placed in the static directory? How are those resources accessed via url?

b) How do you tell your application to associate a function with a specific URL?

c) Why use templating rather than serving html files directly (that is, without `render_template`)?

4. Read the program in the left column. Write the output of the program in the right column.

<pre>def zippy(li_1, li_2, li_3):     s = ''     for a, b, c in zip(li_1, li_2, li_3):         s += '{} {}'.format(a, b)         if c == 'loudly':             s += '!!!!!!'         s += '\n'     return s  names = ['Alice', 'Bob'] actions = ['ambled', 'bellowed'] adverbs = ['anxiously', 'loudly'] print(zippy(names, actions, adverbs))</pre>	<p>What's the output of the program on the left? If the output includes an error, show all output before the error and the type of error that occurs.</p>
<pre>def letter_me():     num = 65     while True:         if num &gt; 90:             num = 65         val = num         num += 1         yield chr(val)  lettered = letter_me()  for i in range(1, 6):     if i % 2 == 1:         print(next(lettered), end='')     else:         next(lettered)</pre>	<p>What's the output of the program on the left? If the output includes an error, show all output before the error and the type of error that occurs.</p>
<pre>def half(s):     return s[:len(s) // 2]  def double(s):     return s * 2  d = {'a': double, 'b': half}  def wordy(letter, s):     funcy = d[letter]     return funcy(s)  print(wordy('b', 'cats')) print(wordy('c', 'cats')) print(wordy('a', 'cats'))</pre>	<p>What's the output of the program on the left? If the output includes an error, show all output before the error and the type of error that occurs.</p>
<pre>import socket # socket setup omitted to save space s.listen(queue)  while True:     client, address = s.accept()     data = client.recv(4096)     if data:         req = data.decode('utf-8')         parts = req.split(' ')         res = parts[0].strip() + 'x'         res + parts[-1].strip()         client.send(bytes(res, 'utf-8'))     client.close()</pre>	<p>What response will the server output to the client if the client is netcat using the following commands.</p> <p>a) nc localhost 5000 what</p> <p>b) nc localhost 5000 hello there how are you</p>

5. Fill in the bodies (the two areas marked `// TODO: FINISH FUNCTION BODY`) of the following two functions that draw squares – one with PIL, the other with turtle. The programs below `draw_square` functions to create a square that's 200 pixels. The function headers and example usage are below. Note that in the PIL version, the color can be specified, and the example has a white outline of a square on a black background.

Drawing a square with PIL	Drawing a square with turtle
<pre>def draw_square(img, left_x, top_y, size, color):      // TODO: FINISH FUNCTION BODY  from PIL import Image img = Image.new('RGB', (400, 400))  <b>draw_square(img, 100, 100, 200, (255, 255, 255))</b>  img.show()</pre>	<pre>def draw_square(t, left_x, top_y, size):      // TODO: FINISH FUNCTION BODY  import turtle my_turtle = turtle.Turtle() wn = turtle.Screen()  <b>draw_square(my_turtle, -100, -100, 200)</b>  wn.mainloop()</pre>

6. Fill in the missing parts (marked `// TODO: FILL IN THIS CODE`) of the flask application below. The flask application allows you to enter comma separated numbers and an operation (`sum` to sum all numbers, `product` to multiply all numbers, and `max` to get the largest number) through a form. When the form is submitted, the page that gets rendered should show the original operation and numbers as well as the result. The form is also displayed underneath the results. If the operation is unrecognized, then the result should be a message that says: “Error, operation not supported”.

#### app.py

```
from flask import Flask
from flask import request
from flask import render_template
app = Flask(__name__)
app.debug = True
```

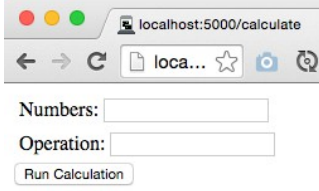
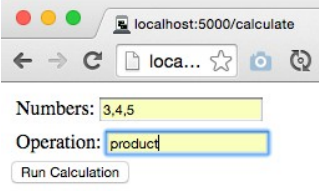
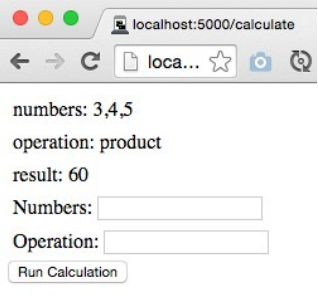
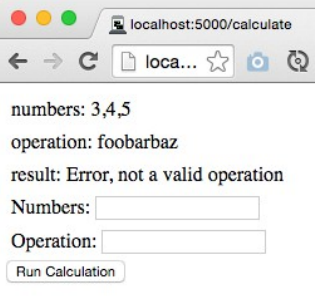
`// TODO: FILL IN THIS CODE`

```
app.run()
```

#### calculate.html

```
{% if res %}
<div>numbers: {{nums}}</div>
<div>operation: {{op}}</div>
<div>result: {{res}}</div>
{% endif %}
```

```
<form method="POST" action="">
  <div>Numbers: <input type="text" name="nums"></div>
  <div>Operation: <input type="text" name="op"></div>
  <input type="submit" value="Run Calculation">
</form>
```

Going to /calculate	Filling out form	After form Submission	If operation doesn't exist
			

7. Create a class called `FixedWidthFile`. This class can be used to access data within a file that has one row of data per line, with each column in the row being a fixed width of characters. The first line of a fixed width file will always contain column names, and each subsequent row will contain actual data. See the example fixed width file in the first column of the table below (“Contents of names.txt”).

Your class will allow you to create a new `FixedWidthFile` object given a file name. Once you have an object, you can call the `get` method on that object to retrieve a piece of data at a given row number (0 for the 1<sup>st</sup>, 1 for the 2<sup>nd</sup>, etc.) and a column name (again, in the example below, `First` would be a column name). Lastly, when `str` is `print` is used with the object, its string representation is just the file's headers separated by commas.

Contents of names.txt	Code Using FixedWidthFile Class	Output
FIRST    LAST    MIDDLE Alice    AndersonA BenjaminBerger    B	<pre>f = FixedWidthFile('names.txt', 8) print(f) print(f.get(0, 'LAST')) print(f.get(1, 'FIRST'))</pre>	FIRST, LAST, MIDDLE Anderson Benjamin

8. Write a decorator called `constrain`. It will:

- (a) take all incoming arguments of a function that it decorates and constrain the values of those arguments to values between 1 and 10, inclusive
- (b) if an argument is less than or greater than this range, then its value will be changed to 1 or 10 respectively
- (c) it will then call the old function with the constrained arguments. The constructor must be able to deal with functions that take arbitrary number of arguments
- (d) see the example and hints below:

Example	Hints
<pre># in this example, the function, # product is decorated with # @constrain  @constrain def product(*args):     p = 1     for n in args:         p *= n     return p  print(product(0, 2))      # --&gt; 2 print(product(2, 99, 3)) # --&gt; 60  # note that the product in the 1<sup>st</sup> # print is 2 and not 0 because the # decorator changes the 1<sup>st</sup> argument # argument to 1 (a similar change # occurs in the 2<sup>nd</sup> print as well)</pre>	<pre># remember that all arguments passed # in to a function as *args shows up # as elements in a tuple  def show_args(*args):     print(args)  show_args("foo", "bar", "baz")  # prints out the tuple: # ("foo", "bar", "baz")  # you can unpack a list or a tuple # into separate arguments to a # function by prefixing with a *  nums = [0, 10, 3] print(list(range(*nums)))</pre>



Net ID: \_\_\_\_\_

Name: \_\_\_\_\_

## Reference Material / Scrap Paper

### Reference for Flask, socket, etc. to be added

<u>Built-in</u>	<u>String Methods</u>	<u>Turtle and Screen</u>	<u>Math Module</u>	<u>PIL</u>	<u>List Methods</u>	<u>File Object Methods</u>
abs bool chr dict enumerat e filter float format input int len list map max min open ord pow print range round set sorted str sum	capitalize count endswith find format index isalnum isalpha isdecimal isdigit islower isnumeric isprintable isspace istitle isupper join lower replace split startswith strip title upper	<u>Turtle Object</u>  back(distance) begin_fill() circle(radius) clear() color(color) color(colorstring) down() end_fill() forward(distance) goto(x, y) hideturtle() left(angle) pensize(size) right(angle) setheading(angle) up()  <u>Screen Object</u>  bgcolor(colorstring) listen() onkeypress(func, key) ontimer(func, time_ms) setup(width, height) tracer(0) update()	acos acosh asin asinh atan atan2 atanh ceil cos cosh degrees floor log log10 log2 pi pow radians sin sinh sqrt tan tanh	<u>Image Module</u>  Image.new(...) Image.open(...)  <u>Image Object</u>  img.size img.load()  <u>PixelAccess Object</u>  (like a dict) get using []'s set using []'s	append count extend index insert pop remove reverse sort	read readline readlines write
					<u>Dictionary Methods</u>	<u>Random Module Functions</u>
					get items keys pop popitem update values	choice randint sample shuffle

#### ASCII Chart

Char	Dec	Char	Dec	Char	Dec	Char	Dec
(nul)	0	(sp)	32	@	64	`	96
(soh)	1	!	33	A	65	a	97
(stx)	2	"	34	B	66	b	98
(etx)	3	#	35	C	67	c	99
(eot)	4	\$	36	D	68	d	100
(eng)	5	%	37	E	69	e	101
(ack)	6	&	38	F	70	f	102
(bel)	7	'	39	G	71	g	103
(bs)	8	(	40	H	72	h	104
(ht)	9	)	41	I	73	i	105
(nl)	10	*	42	J	74	j	106
(vt)	11	+	43	K	75	k	107
(np)	12	,	44	L	76	l	108
(cr)	13	-	45	M	77	m	109
(so)	14	.	46	N	78	n	110
(si)	15	/	47	O	79	o	111
(dle)	16	0	48	P	80	p	112
(dcl)	17	1	49	Q	81	q	113
(dc2)	18	2	50	R	82	r	114
(dc3)	19	3	51	S	83	s	115
(dc4)	20	4	52	T	84	t	116
(nak)	21	5	53	U	85	u	117
(syn)	22	6	54	V	86	v	118
(etb)	23	7	55	W	87	w	119
(can)	24	8	56	X	88	x	120
(em)	25	9	57	Y	89	y	121
(sub)	26	:	58	Z	90	z	122
(esc)	27	;	59	[	91	{	123
(fs)	28	<	60	\	92	}	124
(gs)	29	=	61	]	93	}	125
(rs)	30	>	62	^	94	~	126
(us)	31	?	63	_	95	(del)	127