CS 301 - Spring 2018

Instructor: Laura Hobbes LeGault

Midterm Exam 2 — 16.67%

(Last) Surname: ____________________ (First) Given name: ____________________

NetID (email): ____ @wisc.edu

IMPORTANT: Answers for Dual and Multiple Choice questions *must* be marked on a scantron. Answers for Fill-in-the-blank questions must be marked **on this exam.**

Fill in these fields (left to right) on the scantron form (use #2 pencil):

1. LAST NAME (surname) and FIRST NAME (given name), fill in bubbles
2. IDENTIFICATION NUMBER is your Campus ID number, fill in bubbles
3. Under *ABC* of SPECIAL CODES, write your **lecture number** and fill in:
   - **001** - MWF 9:55a (Hobbes morning)
   - **002** - MWF 1:20p (Hobbes afternoon)
   - **003** - TR 1:00p (Paul Barford)
4. Under F of SPECIAL CODES, write **A** (exam version), fill in bubble **0**

I certify that I will keep my answers covered and do my best to not allow my exam paper to be viewed by another student during the exam or prior to completion of their exam. I also certify that I have not viewed or in any way used another's work in completing my answers. I understand that being caught allowing another to view my work or being caught viewing another's work are both violations of this agreement and either will result in automatic failure of the course and an academic misconduct letter to the Deans Office for myself and any other individuals involved.

Signature: ____

The following exam has 23 questions and is worth a total of 42 points. You will have 50 minutes to complete the exam. **Be sure to read through every question completely.**

1. **Dual Choice** — 12 questions worth 1 point each. Choose the *best* answer.
2. **Multiple Choice** — 9 questions worth 2 points each. Choose the *best* answer.
3. **Fill-in-the-blank** — 4 blanks worth 3 points each. Be complete.

You may not use notes or books, your neighbors, or calculators or any other electronic devices on this exam. **Turn off and put away** any portable electronics now.

Disclaimer: the following are provided for your reference only, and the inclusion of information here does not guarantee it will be used on the exam.

Operator Precedence Table:

level	operator	description
higher	`( <expression> )`	grouping with parentheses
	`x[index]`	indexing
	`* / %`	multiplicative
	`+ -`	additive
↕	`< <= > >=`	relational
	`== !=`	equality
	`not`	logical not
	`and`	logical and
lower	`or`	logical or
	`= += *=`	(compound) assignment

Built-in functions:

`raw_input(p)`	Prompt the user for input using `p` and return the input as a string.
`len(s)`	Return the length (the number of items) of an object.
`range(n)`	Return a list of `n` consecutive integers beginning at `0`.
`range(a,b)`	Return a list of integers beginning at `a` and ending before `b`.
`sorted(x)`	Return a new sorted list from the items in `x`.
`type(x)`	Return the *data type* of the value stored in `x`.

String methods:

`w.isalpha()`	Return true if all characters in string `w` are letters, `w` not empty.
`w.isdigit()`	Return true if all characters in string `w` are numbers, `w` not empty.
`w.upper()`	Return the string `w` transformed to upper case.
`w.lower()`	Return the string `w` transformed to lower case.

Random functions:

`random.randint(a,b)`	Generate a random integer between `a` and `b`, inclusive.

List and dictionary methods:

`list.append(x)`	Add the value `x` to the end of `list`, in place.
`list.insert(i,x)`	Insert the value `x` at the `i`th index of `list`, in place.
`list.remove(x)`	Remove the first instance of the value `x` from `list`, in place.
`list.pop(i)`	Remove and return the value at index `i` from `list`, in place.
`dict.keys()`	Return a copy of `dict`'s list of keys.
`dict.values()`	Return a copy of `dict`'s list of values.

Dual Choice: Terminology

1. Passing a list to a function as an **argument** gives the function a ________ copy of that list. (1)

   A. shallow

   B. deep

2. Incrementing only the first element of a **list** is possible because lists are ________ . (1)

   A. immutable

   B. mutable

3. Given that the following code **does not** cause an error, `x` **must** be a ________ . (1)

   ```
   print (x[11.0])
   ```

   A. list

   B. dictionary

4. **List comprehension** is used for ________ . (1)

   A. replacing any `for` loop

   B. building lists

5. **Appending** the value `x` to the list `m` is the equivalent of **inserting** it as follows: (1)

   A. `m.insert(len(m)-1, x)`

   B. `m.insert(len(m), x)`

6. **Iteration** requires ________ a set of statements. (1)

   A. repeating

   B. calling

7. **Slicing** a list creates a ________ copy of that list. (1)

   A. deep

   B. shallow

8. If `d` is a dictionary `d = {"a":1, "b":2}`, then `len(d) =` ________ . (1)

   A. 2

   B. 4

True or False: Evaluating boolean expressions

9. `2 in { 1:"apple", 2:"banana", 3:"carrot" }` (1)

   A. True

   B. False

10. `"-12.34".isdigit()` (1)

    A. True

    B. False

11. `sorted([3, 4, 2, 1]) == [1, 2, 3, 4]` (1)

    A. True

    B. False

12. `True and (False or True)` (1)

    A. True

    B. False

Multiple Choice: Reading code

13. **Challenge!** What is the *value* in `x` after the following code is executed? (2)

```
def recursive_fcn(string):
    if len(string) == 1:
        return string
    return string[-1] + recursive_fcn(string[:-1])

x = recursive_fcn("parrot")
```

   A. `"torrap"`

   B. `"tptptp"`

   C. `"t"`

   D. This is infinite recursion and `x` will never get a value.

14. What is the *value* in `x` after the following line of code is executed? (2)

```
x = list(range(1,5)).remove(3)
```

A. `None`

B. `4`

C. `3`

D. `2`

15. What is the length of `x` after this code completes? **Be careful!** (2)

```
x = [2, 5, 4, 2, 0, 3]
for elem in x:
    if elem % 2 == 0:
        x.remove(elem)
```

A. 6

B. 4

C. 3

D. 2

16. What is the *data type* of `x` after the following line of code is executed? (2)

```
x = ["14", "0", "37"].pop(0).upper()
```

A. `bool` (boolean)

B. `str` (string)

C. This code produces an AttributeError, cannot use `.upper()` with **NoneType**.

D. This code produces an AttributeError, cannot use `.upper()` with an **int**.

17. If the following expression does not produce an error, what *data type* must `x` be? (2)

```
x[0] == "P"
```

A. `str` (string)

B. `list`

C. `dict` (dictionary)

D. All of the above are legal data types for `x` with this syntax.

18. Which of the following is *not* a legal dictionary **key**? (2)

   A. `{"a":1, "b":2}`
   B. `"apple"`
   C. `5`
   D. All of the above are legal dictionary **keys**.

19. What is the *value* in `x` after this code executes? (2)

```
x = "question"
for i in range(len(x)):
    x += "?"
```

   A. `"question????????"`
   B. `"q?u?e?s?t?i?o?n?"`
   C. `"????????"`
   D. This code produces a TypeError; strings do not support item assignment.

20. What is the *value* in `x` after this code executes? (2)

```
x = [i*2 for i in range(5)]
```

   A. `[0, 0, 1, 1, 2, 2, 3, 3, 4, 4]`
   B. `[[0,1,2,3,4], [0,1,2,3,4]]`
   C. `[0, 2, 4, 6, 8]`
   D. `[0, 1, 4, 9, 16]`

21. Given a dictionary `dict` initialized as follows: (2)

```
dict = { 1:"apple", 2:"banana", 3:"carrot" }
```

which of the following lines of code will cause `len(dict)` to increase?

   A. `dict.append(4, "durian")`
   B. `dict.insert(5, "eggplant")`
   C. `dict[0] = "zucchini"`
   D. `dict[3] = "cucumber"`

Fill-in-the-blank: Writing code

Fill in the blanks to complete the functions as their docstrings indicate. Each blank is worth **3 points**, and there are a total of 4 lines.

22.
```
import random
def plant_power (heroes):
    """ This function returns the total strength of our plant heroes
        as the integer sum (NO factorials).  Provided is a dictionary
        of plant power levels; you simply need to use it properly.
        Assume heroes is a VALID string (e.g.  "pqpqr").
    """
    d = { "p":  random.randint(1,10),
          "q":  random.randint(1,10),
          "r":  random.randint(1,10) }
    result = 0

    for ____________________________ :  # loop through heroes string        (3)

        ________________________ # add correct value from d to result        (3)
    return result
```

23.
```
def add_meeting (new_mtg, sorted):
    """ This function takes in a list of meetings, which look like:
               ["name", start, end]
        If new_mtg fits in the schedule, add it in the correct place.
    """
    i = 0
    # check that new_mtg starts AFTER the meeting at i ENDS

    while i < len(sorted) and ________________________________ :        (3)
        i += 1
    if is_available(new_mtg, sorted):  # True if and only if it fits

        ___________________________   # add new_mtg to the right spot        (3)
```

This page intentionally left blank.