Final Exam

Started: Aug 3 at 1:34pm

Quiz Instructions

READ THIS BEFORE STARTING:

The final exam is organized like a quiz with multiple-choice-type of questions and several problem for which students have to write Python code. There is only ONE attempt possible.

Once started, the exam must be completed within 120 minutes, but no later than the posted deadline. Note that correct or incorrect answers are not indicated as such at the end of the test.

VERY IMPORTANT: Canvas automatically stops test taking and auto-submits the answers/solutions when the deadline passes, so students should **start** taking the exam at least 120 minutes BEFORE the deadline -- 10 PM, the latest.

It is a good idea to start with the easier problems. Also, always pay attention to details. Keep in mind that incomplete or incorrect programming solutions will receive some partial credit.

The FAU policy on **Academic Integrity** will be enforced. Students must work alone and sharing of information related to the exam questions is prohibited. Using any code from the web constitutes cheating and is easy to identify. Students who are found to have similar solutions are considered equally suspect of cheating.

In case of technical issues (e.g. Canvas offline) do not panic. Take a relevant screenshot showing the problem and send right then, without delay, an email from your @FAU account to the instructor (icardei@fau.edu (mailto:icardei@fau.edu)) with the screenshot, an explanation of the technical problem, and a file (Word or PDF) with the solutions you have at that point. Contact Canvas Help -- they have a chat feature with immediate support. Do **NOT** use the Canvas Inbox/email feature, as Canvas itself may be overloaded or offline. Use your FAU email for that important message.

Protect against system failures:

When taking an important Canvas test it helps to write all solutions to a Word file as they are entered on Canvas, just to protect against possible failures, such as web server crashes, browser or a PC crash. Taking screenshots of your work is quite effective, but does not allow easy re-entry.

This assignment links to all objectives in Modules 5-7.

Question 1 **5 pts**

Consider the following code:

```
try:
    x = 0
    .... a function call that raises NO exceptions ...
except ValueError:
    x += 1
except:
    x += 2
else:
    x += 10
finally:
    x += 100

print(x)
```

What is printed to the terminal?

110

Question 2

5 pts

Consider the following code:

```
try:
    x = 0
    .... a function call that raises ValueError...
except ValueError:
    x += 1
except:
    x += 2
else:
    x += 10
finally:
    x += 100

print(x)
```

What is printed to the terminal?

101

Question 3

5 pts

Consider the following recursive function in Python that computes the minimum element from a list:

```
def lst_min(lst):
   .... part A ....  # missing
    m = lst_min(lst[1:])
    return lst[0] if lst[0] < m else m
```

What lines of code could replace part A that is missing?

○
```
    if len(lst) == 0:
       raise IndexError()
```

[✓]
```
    if len(lst) == 1:
        return lst[0]
```

○
```
    if lst.count() == 1:
        return lst[0]
```

○
```
    if len(lst) == 0:
        return 10000000
```

○
```
    if len(lst) == 1:
        return lst[:-1]
```

Question 4

5 pts

Consider this recursive function that computes the sum of all digits of an integer (in base 10):

```
def digits_sum(n):
    if n < 10:
        return n
    ... part A ...   # missing. You find out what's on this line.
```

The recursive case is missing. Which of the following options is the correct line that substitutes part A ?

- ○ return (n // 10) + digits_sum(n % 10)
- ○ return (n % 10) + n // 10
- ○ return digits_sum(n % 10) + (n % 10)
- ○ return (n % 10) + digits_sum(n // 10)
- [✓ ● return (n % 10) + digits_sum(n / 10)]

Question 5

5 pts

A palindrome is a word (or list) that reads the same backwards. Example palindromes: abcdcba, madam.

Here is a recursive function that returns true if the parameter *s* is a palindrome and false otherwise:

```
def ispali(s):
    if len(s) == 0 or len(s) == 1:
        return True
    ... part A ....    # missing. You need to find it out.
```

The recursive case (part A) is missing. Which of the following options for part A is correct ?

- ○ return (s[0] == s[-1]) and ispali(s[1:])
- [✓ ● return (s[0] == s[-1]) or ispali(s[1:-1])]
- ○ return (s[0] == s[-1]) and ispali(s[1:-1])
- ○ return (s[0] == s[:-1]) and ispali(s[1:])
- ○ return (s[0] == s[-1]) and ispali(s[1:-2])

Question 6 5 pts

Which of the following statements are true?

- ☐ It's OK to have functions with side effects in pure functional languages.
- ☑ A higher-order function takes as arguments other functions.
- ☑ In functional programming languages functions are first-class concepts.
- ☐ Lazy computation is when the CPU is to slow to complete a computation.
- ☑ Python is a multi-paradigm programming language.

Question 7 5 pts

If *gen* is a generator function taking no arguments, then what is the type of expression *gen*() ?

- ● the same type as the value output by yield
- ○ iterable class
- ○ generator class
- ○ object class
- ○ iterator class

Question 8 5 pts

Recall the *gen_filter*() and gen_infinite_fibonacci() generators explained in class. What is returned by the call:

```
gen_filter(lambda x: x%3==0,
             gen_filter(lambda x: x%5==0, gen_infinite_fibonacci()))?
```

- the sequence of Fibonacci numbers not divisible to 15
- [✓ the sequence of Fibonacci numbers divisible to 15]
- the empty sequence
- the sequence of Fibonacci numbers divisible to 3

Question 9

30 pts

This is a programming problem. Create a new Python file named p1.py and write the solution in it. When done, upload the file by clicking on the button below. Make sure your code works, follows strictly all requirements, and complies with the Python coding style taught in class.

Do not change the signature (name, parameter list) of any function, if given. Do not worry if Canvas changes the file name after you upload it.

To get credit for this problem the code must **NOT** use: *for* loops, *while* loops, functions from other modules, the *map* function, or any functions except *type*(), *len*(), and list slices.

A **flat list** has elements that are not lists themselves, while a nested list has some elements that are lists. E.g. [1,2,3] is a flat list, while [1, [[2], 3], 4] is a nested list. For this problem a list *lst* is defined flat if for each element *x* in *lst*, type(*x*) != **list**. [ ] is considered to be flat. The flat version of [1, [[2], 3], 4] is [1, 2, 3, 4].

a) Write a **recursive function** *mapflatlst*(*lst, fun*) that takes as parameters a list *lst* (assumed to be flat) and a function *fun* and returns a list with elements from *lst* mapped through the function *fun*. This means that if *lst* has elements [x,y,z], the function returns [fun(x), fun(y), fun(z)]. *mapflatlst*(**[ ]**, *fun*) returns **[ ]**, the empty list.

Example: suppose we have function times2 defined like this:

```
def times2(x): return 2 * x
```

Then, *mapflatlst*([1,2,3,4,5], *times2)* returns [2, 4, 6, 8, 10].

b) Write a **recursive function** *maplst*(*lst, fun*) that takes as parameters a list *lst* (that can be **nested**) and a function *fun* and returns a **flat** list with elements from the flattened version of *lst* mapped through the function *fun*.

maplst(**[ ]**, *fun*) returns **[ ]**, the empty list.

Example: with function *times2* defined like above,

maplst([[[1]], [2, [3, 4], 5], 6]*, times2)* returns [2, 4, 6, 8, 10, 12].

Hint: read the final-prep.py file posted on the Unit 6 module, under "final exam like problems". Do not use the functions from that file to flatten the list to simplify your task. Use the ideas only.

Upload p1.py

Your file has been successfully uploaded.

Question 10

30 pts

This is a programming problem. Create a new Python file named p2.py and write the solution in it. When done, upload the file by clicking on the button below. Make sure your code works, follows strictly all requirements, and complies with the Python coding style taught in class.

Do not change the signature (name, parameter list) of any function, if given.

Pay attention to details.

a) Write a **generator** *running_max*(*seq*) that takes a sequence *seq* and generates a sequence with the current running maximum for successive elements from *seq*. In other words, for each element x from input sequence *seq* generator *running_max*(*seq*) yields the maximum value between *x* and all other elements in *seq* that came before it. Use operator > for comparison.
Example: the next program:

```
for x in running_max([1,3,2,4,0,2]):
    print(x, end=", ")
```

prints 1, 3, 3, 4, 4, 4.

Remember that running_max is a GENERATOR and not a normal function. It must use **yield** and not return.

b) For many applications that need a sequence of running maxima we need to "transform" each element of a sequence and use that new value for consideration when computing the next maximum element in the output sequence.

For example, consider a sequence of (*year*, *wins*) tuples describing the number of games won by a team in each year.

```
teamdata = [(2010, 15), (2011, 12), (2012, 20), (2013, 23), (2014, 19), (2015, 21)]
```

We want to obtain the sequence of running maxima with elements from sequence *teamdata* with the comparison being done on the *wins* part of the tuples: (2010, 15), (2010, 15), (2012, 20), (2013, 23), (2013, 23), (2013, 23).

The generator from part a) does not work since '>' applied to tuples uses the first element (the year). We need a new generator that also takes as parameter a function to transform each sequence element to the value used for comparison.

Write a **generator** *running_mapped_max*(*seq, fun*) that takes a sequence *seq* and generates a sequence with the current running maximum for successive elements from *seq* where the value used for comparison is computed with a function *fun* that takes as parameter one value from *seq*.

Test your generator using the *fun* parameter *lambda yw*: *yw*[1]:

```
for year_wins in running_mapped_max(teamdata, lambda yw: yw[1]):
    print(year_wins, end=", ")
```

The output should be sequence (2010, 15), (2010, 15), (2012, 20), (2013, 23), (2013, 23), (2013, 23).

c) Write a **generator expression** that uses generator *running_mapped_max*, variable *teamdata*, to generate the sequence of (year, wins) tuples from the running max examples above that have a year value <= 2012. The result should be sequence (2010, 15), (2010, 15), (2012, 20).

Upload Choose a File

Quiz saved at 3:06pm

Submit Quiz