CS 1358 Introduction to Programming in Python

Spring Semester 2024

Prof. Pai H. Chou

Self-Check 13

Answer the following questions to check your understanding of your material. Expect the same kind of questions to show up on your tests. This self check is for Python Standard Library part 1.

## 1. Definitions and Short Answers - functions

1. If a function is **built-in**, do you have to **import** it first, or can you just call it without importing?
2. If a function is in the **standard library**, do you have to **import** it first, or can you just call it without importing?
3. Which of the following are valid calls to the eval() function, and what are their return values?
   1. eval(2 + 3)
   2. eval('2 + 3')
   3. eval(len("hello"))
   4. eval('len("hello")')
   5. eval('hello world')
   6. eval('"hello world"')
4. Which of the following are valid calls to the eval() function, and what are their return values?
   1. a = 5

eval('a + 3')

* 1. a = 5  
     eval('a + 3', {'a': 7})
  2. a = 5  
     eval('a + 3', {})

1. Is the following a valid call to exec() function and what is the results?
   1. exec('def hello(s):\n print(f"hello {s}")\n')  
      hello('Mike')
2. When your Python program first starts,  
   >>> x = 3  
   >>> dir()  
   ['\_\_annotations\_\_', '\_\_builtins\_\_', '\_\_doc\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'x']  
   which shows that the name 'x' has been added to the global name space. However,  
     
   >>> max(x, 5)  
   5  
   >>>  
   However, the name max is not in the global name space as shown in dir(). Which of the following is a correct explanation of how Python knows max is the name of a function that you can call?
   1. max is a **keyword** in Python
   2. max is defined in the **global** name space but is just **hidden**
   3. max is defined in the \_\_builtins\_\_ name space, which is searched after the global name space
   4. max is defined in a package in the **standard library** and must be **imported** before it can be called.
3. If you do the following  
   >>> max, min = min, max  
   >>> L = [1, 7, 3, 4, 6]  
   >>> max(L), min(L)  
     
   What do you get?
4. If you do the following  
   >>> L = [1, 7, 3, 4, 6]  
   >>> len = 'a'  
   >>> len(L)  
   1. What do you get by len(L)? or do you get an error?
   2. if the built-in len() function has been redefined to 'a', can you still get the original definition back? If so, how? can the original len() be called?
5. The datetime module contains the following classes  
   datetime  
   date  
   time  
   timedelta  
   1. Do you need to do an import statement before you can use any of the four classes above?
   2. If you see the statement  
      import datetime   
      Does datetime refer to the **module** or **class**?
   3. if you see the statement  
      from datetime import \*  
      What is the meaning of \* ?  
      After the import statement, does datetime refer to the **module** or **class**?
   4. If you want to import the "datetime **class within** the datetime **module**" without importing the other classes (date, time, deltatime), what import statement should you use?
   5. The constructor for the datetime class takes three required parameters: *year*, *day*, and *hour*. What is the Python code for instantiating an object of datetime class with year=2019, month=12, day=10, and assign it to the identifier x? Include also the statement for import.
   6. The documentation says datetime.now is a **class method** that takes no parameter. How do you call it on the datetime class? If x is an instance of datetime from the previous question, are you allowed to call x.now()?
6. **Operator overloading** is supported on datetime and deltatime classes. Indicate the combinations supported by filling in the result type; or indicate if the combination is not supported.

| **class** | **operator** | **class** | **result class** |
| --- | --- | --- | --- |
| timedelta | + | timedelta | timedelta |
| timedelta | - | timedelta | ?? |
| timedelta | \* | int | ?? |
| datetime | + | datetime | ?? |
| datetime | - | datetime | ?? |
| datetime | + | timedelta | ?? |
| datetime | - | timedelta | ?? |
|  |  |  |  |

1. In the calendar module,
   1. what is the difference between the TextCalendar and the HTMLCalendar class?
   2. in the TextCalendar class, what is the difference between the prmonth() and formatmonth() methods?
2. In the namedtuple class in the collections module,  
   >>> from collections import namedtuple  
   >>> Point = namedtuple('Point', ['x', 'y'])
   1. What is the type of Point? Is it an instance of namedtuple class, or is a class?
   2. To continue with the example above, the next line is   
      >>> p = Point(2, 3)  
      What kind of object is p? Is it an instance of namedtuple class? an instance of Point class? an instance of tuple? Is it mutable?
3. In the Counter class in the collections module,  
   >>> from collections import Counter  
   >>> c = Counter(['dog','cat','dog','cow','dog','cat'])  
   >>> c  
   1. What do you expect to see?
   2. How do you find the value with the most occurrences?
   3. How do you find the value with the least occurrences?
4. Consider the overloaded operators defined by Counter class in the collections module, what is the value you expect from the expressions?

| a | op | b | value |
| --- | --- | --- | --- |
| Counter('abacus') | + | Counter('aba') | Counter({'a': 4, 'b': 2, 'c': 1, 'u': 1, 's': 1}) |
| Counter('abacus') | - | Counter('ada') | ?? |
| Counter('ada') | - | Counter('abacus') | ?? |
| Counter('dust') | & | Counter('rust') | ?? |
| Counter('dust') | | | Counter('rust') | ?? |

1. In the collections.abc module for **abstract base classes**, some such base classes include Container, Hashable, Iterable, Iterator, …, Sequence, MutableSequence, etc.
   1. Can you **instantiate** an object from one of these abc's? for example,  
      from collections import abc  
      x = abc.Iterable()  
      ?
   2. Can you test if an object is **an instance of** one of these abc's? for example,  
      x = 'hello'  
      if isinstance(x, abc.Iterable):  
       print('str is iterable')  
      else:  
       print('str is not iterable')
2. In the types module, a number of classes are also defined, including FunctionType, LambdaType, GeneratorType, MethodType, BuiltinFunctionType, ModuleType, etc.
   1. Are these classes in types module also abstract base classes like those defined in collections.abc module?
   2. Are you expected to use these classes for instantiation by calling their constructors? If not, why not? For example, is the following the expected usage  
      >>> import types  
      >>> f = types.FunctionType()
3. What is the purpose of an **enumerated type** as in the Enum class in the enum module? Consider the code  
    1 from enum import Enum  
    2 Animal = Enum('Animal', ['ANT', 'BEE', 'CAT', 'DOG'])
   1. What is the value of Animal(1)? Animal(3)?
   2. What is the value of Animal.BEE?
   3. What is the value of str(Animal['CAT'])
   4. What is the value of Animal.DOG > Animal.ANT?
4. What is the difference between the built-in float type and Decimal class in the decimal module?
   1. What is the value of   
      1.1 + 2.2 == 3.3  
      ?
   2. Assuming you have from decimal import Decimal, what is the value of  
      Decimal('1.1') + Decimal('2.2') == Decimal('3.3')  
      ?
5. Assume you have from fractions import Fraction, what is the value of  
   Fraction(16, -10)  
   ?
6. In the random module,
   1. What are the possible values of   
      random.randrange(10)  
      ?
   2. What are possible values of   
      random.choice(['win', 'lose', 'draw'])  
      ?
   3. What is the purpose of a random **seed**? If you have  
      import random  
      r = random.Random()

s = random.Random()  
r.seed(100)  
x = r.randint(1, 100)  
s.seed(100)  
y = s.randint(1, 100)  
is x == y?

1. In the itertools module,
   1. there is a class named count, and you can use it like  
      >>> c = itertools.count(10)  
      >>> next(c)  
      10  
      >>> next(c)  
      11  
      >>> next(c)  
      12  
      Why is this class useful, and why can't it be done with the built-in range()?
   2. There is another class named cycle, and you can use it like  
      >>> cy = itertools.cycle(['a', 'b', 'c'])  
      >>> next(cy)  
      'a'  
      >>> next(cy)  
      'b'  
      >>> next(cy)  
      'c'  
      >>> next(cy)  
      'a'  
      >>> next(cy)  
      'b'  
      Why is this class useful, and why can't it be done with the built-in range()?
   3. There is another class named zip\_longest. Example use is  
      >>> list(itertools.zip\_longest('ABCD', 'WXY', '12', fillvalue='-'))

[('A', 'W', '1'), ('B', 'X', '2'), ('C', 'Y', '-'), ('D', '-', '-')]

Explain how this can be useful for adding two polynomial functions.

## 2. Programming

1. (Difficulty: ★★☆☆☆) Write a "rock, paper, scissors" game using the random module.  
     
   $ python3 rps.py  
   rock, paper, scissors? rock  
   I am also rock - tied!  
   rock, paper, scissors? paper  
   I am rock - I lose!  
   rock, paper, scissors? scissors  
   I am rock - you lose!  
   rock, paper, scissors? rabbit  
   rabbit is invalid - try again? quit  
   bye  
   $
2. (Difficulty: ★★★☆☆) Define a Matrix class to represent numbers as a two-dimensional array.  
     
   The constructor for the matrix is a list of lists of numbers. A 3x3 matrix

| 1 | 2 | 3 |
| --- | --- | --- |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

would be constructed as  
M = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
  
Define the following methods  
class Matrix:  
 def \_\_init\_\_(self, data):  
 # data is the list of lists of value  
  
 def row(self, r):  
 # return the r-th row in the form of a list  
 # r is from 0.. number of rows  
 # in the exmaple above, M.row(1) would return  
 # [4, 5, 6]  
  
 def column(self, c):  
 # retun a the c-th column in the form of a list.  
 # in the example above, M.column(2) would return   
 # [3, 6, 9]  
  
 @property  
 def nrows(self):  
 # return the number of rows  
  
 @property  
 def ncolumns(self):  
 # return the number of columns  
  
 def \_\_getitem\_\_(self, ij):  
 # return the matrix element at ij, where ij is a tuple  
 # for the (row, column). For example above,  
 #   
  
 def \_\_setitem\_\_(self, ij, val):  
 # assign the value to the matrix as ij, where ij is   
 # a tuple for (row, column)  
  
  
 def transpose(self):  
 # return a new Matrix whose content is same as this   
 # Matrix except the row and column positions are   
 # switched. In the example above,   
 # M.transpose() would return  
 # Matrix([[1, 4, 7], [2, 5, 8], [3, 6, 9]])  
 # Note: use zip() to do the transpose  
  
 def randomize(self):  
 # return another matrix whose content is the same as  
 # this matrix except their positions are randomized.  
  
 def \_\_matmul\_\_(self, other):  
 # return the matrix product for the two matrices A B  
 # p[i,j] = sum(A[i,k] \* B[k,j]) for 0 <= k <=

　 #　A.ncolumns where A.ncolumns must be == B.nrows  
  
Note: use itertools.longest\_zip in at least one of the methods.

1. (Difficulty: ★★★★☆) Write a function that can take a variable number of parameters to make a postfix calculator for dates. This is similar to the postcalc example from [HW8](https://docs.google.com/document/d/17qlevY6uUAdHg-a-nJQu42i2Ouq4bwzv8vzRr5wRzT0/edit?usp=sharing), except you work on dates instead of numbers.  
     
   The parameter list consists of either the operands or the operators. An operand is either a date string or a date-delta string and is pushed on the stack. An operator is a string that indicates the action to take. A binary arithmetic operator pops the top two elements from the stack and pushes back the result.

| **argument** | **action** |
| --- | --- |
| date(year, month, day) | push date onto stack |
| days(d) | push days delta onto the stack |
| weeks(w) | push weeks delta onto the stack |
| months(m) | push months delta onto the stack |
| 'today' 'tomorrow' 'yesterday' | push today's date, tomorrow's date, or yesterday's date onto the stack |
| 'add' | A = pop(); B = pop(); push(A+B) |
| 'sub' | A= pop(); B = pop(), push(A-B) |
| 'swap' | A=pop(); B=pop(); push(A); push(B) |

Write a stack-style date-time calculator function called datecalc. Here is an example  
  
$ python3 datecalc.py  
>>> datecalc('today', 'tomorrow', 'yesterday', days(4))  
[date(2019, 12, 3), date(2019, 12, 4), date(2019, 12, 2), days(4)]  
>>> datecalc('today', 'tomorrow', 'yesterday', days(4), 'add')  
[date(2019, 12, 3), date(2019, 12, 4), date(2019, 12, 06)]  
>>> datecalc('today', months(2), 'add')  
[date(2020, 02, 03)]  
>>> datecalc('today', date(2019, 12, 10), date(2019, 12, 20), 'sub', 'add')  
[date(2019, 12, 13)]  
>>> datecalc('today', weeks(2), 'add', months(2), 'swap', 'sub')  
[date(2019, 10, 17)]  
  
Hint:   
  
Obviously, you should take advantage of the datetime module to do as much of the work as possible. You will need to define your own classes  
  
days  
weeks  
months  
  
The datetime.timedelta class can be the base class for your own days class and weeks class. The days class would simply define a constructor that passes the days parameter to the base class; the weeks class is similar except it is simply in units of 7 days. The overloaded operators can be inherited directly from the timedelta class. You also need to define your own \_\_repr\_\_ special method for these two classes so their values can be displayed accordingly.  
  
Your months class would be your own class. The reason is that the number of days depends on the actual date. So, you can't simply say 2 months is 60 days. Instead, when you do date + months or date - months, you operate on the month (and maybe year) field and return a newly constructed date object.

Your months class needs to define the special methods   
\_\_add\_\_(self, RHS) # if RHS is date, return new date;   
 # if RHS is months, return sum months  
\_\_sub\_\_(self, RHS) # RHS can only be months;   
 # => return date constructor call with  
 # updated month value  
 # all other types => type error  
\_\_radd\_\_(self, LHS) # simply return self + LHS  
 # and LHS can only be date.  
\_\_rsub\_\_(self, LHS) # LHS can only be months;   
 # => return date constructor call with  
 # updated month value  
  
Once these classes are all working, then put your code into the loop structure as in the postcalc. You also need to update the string comparison so that the strings such as yesterday, today, and tomorrow get mapped to the respective date object.