Abby Had a Pretty Kitty

(yes I did take professional photos of my cat… sue me)

Consider the code below:

```
class Cat:
    hair_length = "short"
    species_name = "Cat"
    def __init__(self, name, color, age):
        self.name = name
        self.color = color
        self.age = age
    def meow(self):
        print("meowza")
    def __str__(self):
        return f"Kitty named {self.name}, who is {self.color}"
ChaiChai = Cat("Chai Chai", "Calico", 5)
```

What value will the expression `str(Chai Chai)` evaluate to?

A. “Cat”
B. “short”
C. “Kitty named Chai Chai, who is Calico”
D. “Chai Chai”
E. <Cat>

What value will the expression ChaiChai`.species_name` evaluate to?

F. “Cat”
G. “ChaiChai”
H. “short”
I. “Kitty named chai chai, who is calico”
J. <Cat>

What value will the expression `[l for letter in ChaiChai.color if c in "aeiou"]` evaluate to?

A. "Calico"
B. ['a', 'i', 'o']
C. ['C', 'l', 'c']
D. "aio"
E. ['Calico']

Linked up

```
class Link:
    empty = ()

    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

donovan = Link("Donovan")
emily = Link("Emily")
ethan = Link("Ethan")

donovan.rest = Link("Tyler", emily)
emily.rest = ethan
ethan.rest = donovan.rest
```

After this code executes, what is the value of `donovan.rest.rest.first`?

A. `"Donovan"`
B. `"Emily"`
C. `"Ethan"`
D. `"Tyler"`
E. `()`

After this code executes, what is the value of ethan.rest.rest.rest.first?

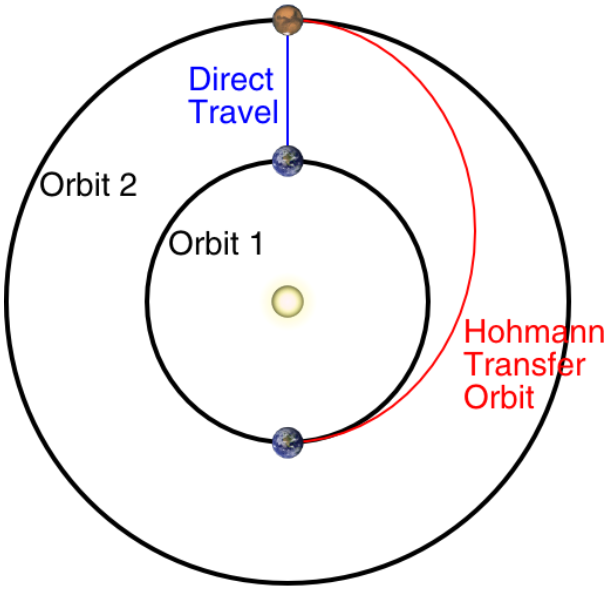
A. `"Emily"`
B. `"Ethan"`
C. `"Donovan"`
D. `"Shadow"`
E. `()`

What does `Link.empty` evaluate to?

A. “empty”
B. 0
C. None
D. ()
E. “Tyler”

(13 points) **Space Travel** 🚀

Have you ever wondered why it takes nine months for a probe to get to Mars after launch? Could it get there faster? In this question we're going to explore interplanetary travel times using two different travel methods (one real, one mostly fictional).



Background – Travel Methods

The Hohmann Transfer

The simplest way to get between two planets is using what is called a *Hohmann Transfer Orbit*. This is how most interplanetary trips occur today. The rocket fires its engines just enough to get into the correct orbit, drifts nearly the entire trip in zero gravity, and then fires its engines at the end to go into orbit around the destination. This uses the least amount of fuel but takes the most time. This is the method of travel we will play with, but the next one below is fun to think about.

Direct Travel

A much faster course is to leave your engines on the entire time, accelerate to the halfway point of your trip, turn around, and then decelerate to your destination. This also has the advantage of simulating gravity for the entire trip instead of being in freefall the entire time. It uses a lot more fuel (and is not currently possible for large craft or high accelerations with today's technology) but is significantly faster.

Implementation

The travel time for using a Hohmann Transfer Orbit is given by the following formula:

$$t_{Hohmann} = \pi \sqrt{\frac{(r1 + r2)^3}{8\mu}}$$

where:

- $t_{Hohmann}$ is the time (in seconds) to make the trip
- r1 and r2 are the radii of the two orbits measured in meters. For example, the radius of the **earth's orbit is 149,600,000,000 (1.496×10^{11})** meters – this number can be represented as `1.496e11` in Python.
- μ (mu) is simply a constant equal to the gravitational constant times the mass of the sun. For this exam we'll use, **$\mu = 1.3274745 \times 10^{20}$**

The Python `math` library has both the variable `pi` and the function `sqrt()` which will be needed. Recall the Python built-in function `pow()` that raises the first parameter to the second parameter. Consider the code:

```
import math

EARTH_RADIUS = 1.496e11
MARS_RADIUS = 2.28e11

def hohmann_time(r1, r2=_____(a)):
  mu = _____(b)
  convert_seconds_to_days = _____(c)
  def cubed_sum(x, y):
    return ____(d)
  return convert_seconds_to_days(__(e) * __(f)(___(g)))

print(f"Hohmann Time to Mars: {hoffman_time(MARS_RADIUS)} days")
```

What line of code could go in blank (a) assuming we want the Earth's radius to be the default?

A. `EARTH_RADIUS`
B. `MARS_RADIUS`
C. `1.3274745e20`
D. `(x+y)**3`
E. `pow(x + y, 3)`
F. `math.pi`
G. `Math.sqrt`

What line of code could go in blank (b) ?

A. `EARTH_RADIUS`
B. `MARS_RADIUS`
C. `1.3274745e20`
D. `(x+y)**3`
E. `pow(x + y, 3)`
F. `math.pi`
G. `math.sqrt`

What line of code could go in blank (c) to assign a lambda function to `convert_seconds_to_days`? — choose all options that could work

A. `lambda secs : secs / 86400`
B. `lambda secs : secs / (60 * 60 * 24)`
C. `lambda secs : secs / 60 / 60 / 12`
D. `lambda x : x / EARTH_RADIUS`

What line of code could go in blank (d) ? — choose all options that could work

A. `EARTH_RADIUS`
B. `MARS_RADIUS`
C. `1.3274745e20`
D. `(x+y)**3`
E. `pow(x + y, 3)`
F. `math.pi`
G. `math.sqrt`

Use the $t_{Hohmann}$ equation and the `math` library for these next questions.

What code could go in blank (e) ?

A. `EARTH_RADIUS`
B. `MARS_RADIUS`
C. `1.3274745e20`
D. `(x+y)**3`
E. `pow(x + y, 3)`
F. `math.pi`
G. `Math.sqrt`

What code could go in blank (f) ? Hint: must evaluate to a function.

A. `EARTH_RADIUS`
B. `MARS_RADIUS`
C. `1.3274745e20`
D. `(x+y)**3`
E. `pow(x + y, 3)`
F. `math.pi`
G. `math.sqrt`

What code could go in blank (g) ?

A. `cubed_sum(r1, r2) / (8 * mu)`
B. `cubed_sum(r1, r2) / mu`
C. `cubed_sum / (8 * mu)`
D. `cubed_sum(r2, r1) / mu`

Nothin' Regular About These Expressions

Given the pattern [qrstuvAD]+34, which of the following would be fully matched? Select the two that are correct.

A. qZ34
B. qrstuAD34
C. QRST34
D. qrs34class
E. qrs34
F. ADqv34
G. ADqv34z

Which of the following patterns would fully match "MATH112" and "MATH 112" but not "MATH213" or "MATH 113"? Select the two that are correct.

A. MATH\s?112
B. MATH ?1{3}
C. MATH ?\d+
D. MATH ?112
E. \w+112
F. MATH\s112
G. MATH\s?1+

Which of the following patterns would fully match "abby", "jason", "jed", "luke", "grace", "ari", and "cool" but not completely match "tom"?
Select the two that are correct.

A. (abby|jason|jed|luke|grace|ari|cool)
B. \w{3,5}
C. [ajlgc].*
D. t\w+
E. [abcgjl].{2,}
F. ([ajgl].{2,}|cool)
G. \w+

Every Other One

```
1  def every_other(s):
2      """Returns every other letter from s,
3      starting with the first letter.
4      >>> every_other('racecar')
5      'rcea'
6      """
7      if len(s) <= 1:
8          return s
9      else:
10         return s[0] + every_other(s[2:])
```

Which lines contain the recursive base case?

A. Line 1

B. Lines 2–6

C. Lines 7–8

D. Lines 9–10

Which lines contain the recursive call?

A. Line 1

B. Lines 2–6

C. Lines 7–8

D. Lines 9–10

Which lines contain a docstring?

A. Line 1

B. Lines 2–6

C. Lines 7–8

D. Lines 9–10

What value will this function return if called with the parameter "pineapple"?

A. "pineapple"

B. "pnape"

C. "peple"

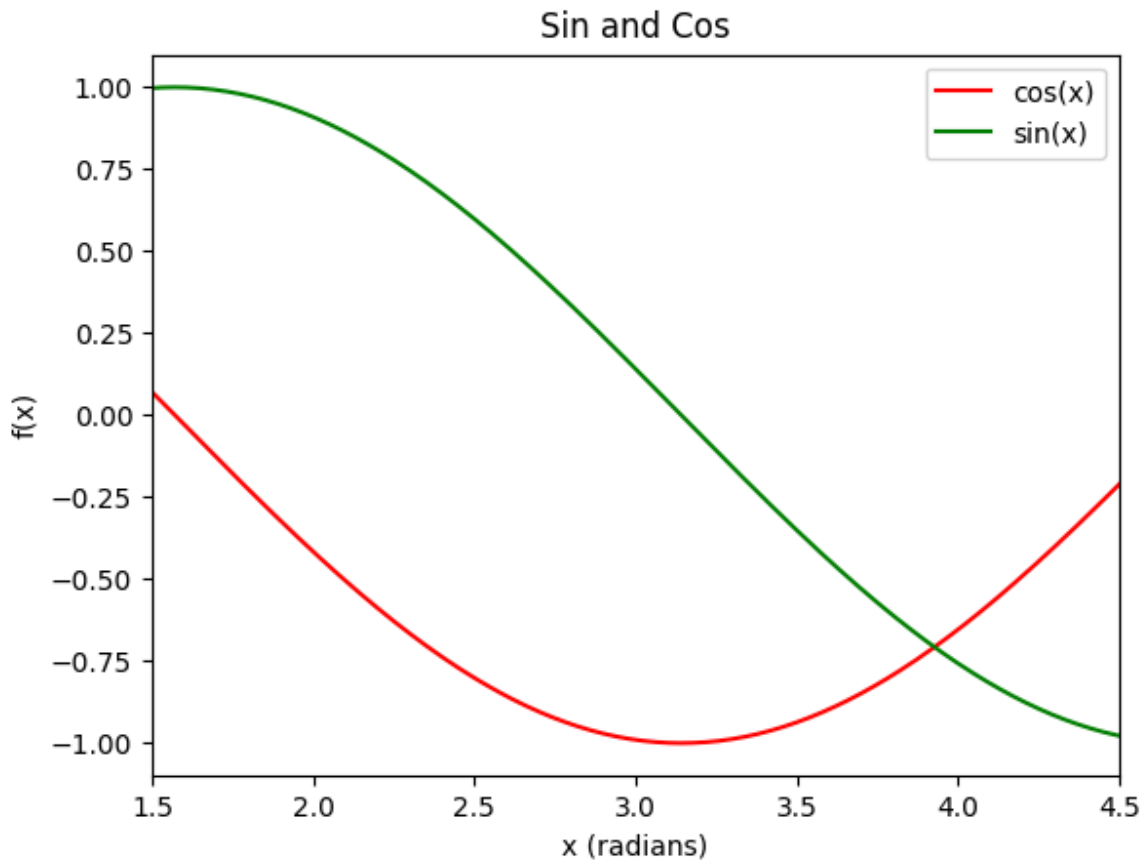
D. "pnapl"

E. "pnal"

Yes, We are Plotting Against You

Consider the following code that constructs the plot below it

```
1   import matplotlib.pyplot as plt
2   from math import cos, sin
3
4   x = [i/100 for i in range(0, 1300)]
5   y1 = [cos(val) for val in x]
6   y2 = [sin(val) for val in x]
7
8   plt.plot(x, y1, _____(a)_____, _____(c)_____="cos(x)")
9   plt.plot(x, y2, _____(b)_____, _____(c)_____="sin(x)")
10  plt.xlim(1.5, 4.5)
11  plt._____(d)_____("Sin and Cos")
12  plt.xlabel("x (radians)")
13  plt.ylabel("f(x)")
14  plt.legend()
15
16  plt.show()
```



What goes in positions (a) & (b) on lines 8 & 9 to produce the red and green lines?

A. ‘red’, ‘green’
B. ‘r’, ‘g’
C. ‘green’, ‘red’
D. ‘g’, ‘r’

What keyword goes in position (c) on lines 8 & 9 to give a title to the sets of points?

E. title
B. label
C. name
D. id
E. text

What keyword goes in position (d) on line 11 to put the text at the top of the plot?

J. title
B. label
C. name
D. id
E. text

HTML

```
<html>
  <head>
    <title>A CS 111 Practice Final Doc</title>
  </head>
  <body>
  <h1 id="intro">Introduction</h1>
  <p id="intro">This page is from your <a href="/staff/#tas">TA's</a>
on the final exam!</p>
  <img src="/assets/images/staff/david_bauch.jpeg" alt="The best TA">
  <!-- Notice the '/' in src -->
  <p class="list-start">Let's remember some things:</p>
  <ul>
    <li class="odd">How to request a web page</li>
    <li class="even">How to find a tag</li>
    <li class="odd">How to construct URLs</li>
    <li class="even">How to print content on a page</li>
    <li class="odd">How to use regular expressions in <a

href="https://www.crummy.com/software/BeautifulSoup/bs4/doc/">
        BeautifulSoup</a></li>
  </ul>
  <h1 id="farewell">Goodbye now</h1>
  <p id="farewell">Hopefully this preps you for the final.</p>
  <img src="assets/images/staff/connor_nesbit.jpeg" alt="The 2nd best
TA">
  <!-- Notice the NO '/' in src -->
  </body>
</html>
```

Assume that this page is located at http://cs111.byu.edu/practice/final.html, and that you have created a BeautifulSoup object called soup from this HTML file.

Which of the following would return the paragraph tag with the 'id' attribute "farewell"?

A. soup.find_all("p")
B. soup.find_all("p", id=True)
C. soup.find_all(True, "farewell")
D. soup.find_all(id="farewell")
E. soup.find_all("p", {"id": "farewell"})
F. soup.find_all(True, {"id": "farewell"})

What is the full URL to the image after the first paragraph and the image after p#farewell?

A. http://cs111.byu.edu/assets/images/staff/david_bauch.jpeg
B. http://byu.edu/assets/images/staff/david_bauch.jpeg
C. http://cs111.byu.edu/practice/david_bauch.jpeg
D. http://cs111.byu.edu/practice/assets/images/staff/connor_nesbit.jpeg
E. http://byu.edu/practice/assets/images/staff/david_bauch.jp eg
F. http://byu.edu/practice/assets/images/staff/david_bauch.jp eg

What is the full URL to the staff page linked in the first paragraph?

A. http://cs111.byu.edu/final/staff
B. http://byu.edu/staff
C. http://cs.byu.edu/staff/
D. http://cs111.byu.edu/staff/

What code would you write to get the text of the link to staff page (not the link, but the text that is being linked)

A. soup.find_all('a')
B. soup.find_all('href')
C. soup.find_all('href')[0].string
D. soup.find_all('a')[0]
E. soup.find_all('a')[0].string
F. soup.find_all('a')[0]['href']

What does the following code print?

```
tags = soup.find_all('li')
for tag in tags:
    print(tag.string)
```

A. How to request a web page
   How to find a tag
   How to construct URLs
   How to print content on a page
   How to use regular expressions in
B. How to request a web page
   How to construct URLs
   How to use regular expressions in
C. How to find a tag
   How to print content on a page
D. Hopefully you remember the following:
E. It prints nothing

Let's plant a garden! 🌱🍈🌻

It is common for gardeners to start many vegetables, herbs, and flowers from seed during the end of winter and early spring. Most seed packages give the number of weeks before the last average frost date the gardener should plant the seed. Then, after the last average frost date, the plant will be big enough to be planted outside.
Consider the `Seed` class and fill in the missing code:

```
class Seed:
  """A Seed has two instance attributes: name and weeks.
  name must be a string, and weeks is a number representing the number of
weeks
  before the last average frost date the seed should be planted.
  """
  def __init__(self, name, weeks):
    self.name = name
    self.weeks = weeks

  def __str__(self):
    return f'Plant {___(a)} seeds {___(b)} weeks before the last average
frost date'
```

What line of code could go in blank (a)?

A. `type(rest)`
B. `self.weeks`
C. `Seed`
D. `self.name`
E. `rest = rest.rest`
F. `n += 1`
G. `first = first.rest`

What line of code could go in blank (b)?

A. `atype(rest)`
B. `self.weeks`
C. `Seed`
D. `self.name`
E. `rest = rest.rest`
F. `n += 1`
G. `first = first.rest`

Consider the following linked-list like class `SeedCatalog` that stores a linked list of `Seed`s:

```
class SeedCatalog:
  """A SeedCatalog has two instance attributes: first and rest. first must
be a Seed
  rest must be a SeedCatalog or None
  """
  def __init__(self, first, rest):
    assert issubclass(type(first), ___(c)___) or type(first) == ___(c)___
    self.first = first
    self.rest = rest
```

What line of code could go in both blanks (c) ?

H. `type(rest)`
I. `self.weeks`
J. `Seed`
K. `self.name`
L. `rest = rest.rest`
M. `n += 1`
N. `first = first.rest`

The following function is for the `SeedCatalog` class to compute the length of the list.

```
  def __len__(self):
    n = 1
    rest = self.rest
    while isinstance(rest, SeedCatalog):
        ___(d1)___
        ___(d2)___
    if rest is not None:
        raise TypeError('length attempted on improper list')
    return n
```

What two lines of code could go in blanks (d1) and (d2) ? — choose two that work together

O. `type(rest)`
P. `self.weeks`
Q. `Seed`
R. `self.name`
S. `rest = rest.rest`
T. `n += 1`
U. `first = first.rest`

Consider the following commentary function which, when given a `SeedCatalog` and the number of weeks before the last average frost date, will tell the gardener when each type of seed needs to be planted. Each subsequent call to the returned commentary function will decrease the weeks before frost by one.

```
def when_to_plant(seed_catalog=None, weeks_before_frost=0):
  """
  >>> catalog = SeedCatalog(Seed('cucumber', 3), SeedCatalog(Seed('basil',
5), SeedCatalog(Seed('tomatillo', 0), None)))
  >>> spring_planting = when_to_plant(catalog, 4)
  >>> spring_planting = spring_planting()
  Need to plant cucumber seeds in 1 week(s).
  Too late to plant basil.
  Need to plant tomatillo seeds in 4 week(s).
  >>> spring_planting = spring_planting()
  Need to plant cucumber seeds now!
  Too late to plant basil.
  Need to plant tomatillo seeds in 3 week(s).
  """
  def say():
    catalog = seed_catalog
    while catalog != None:
      if catalog.first.weeks > weeks_before_frost:
        print(f"Too late to plant {catalog.first.name}.")
      elif catalog.first.weeks (e) weeks_before_frost:
        print(f"Need to plant {catalog.first.name} seeds now!")
      else:
        print(f"Need to plant {catalog.first.name} seeds in {   (f)  }
week(s).")
      catalog = catalog.rest
    return when_to_plant(seed_catalog,    (g)  )
  return say
```

What *boolean* operator could go in blank (e) ?

V. !=
B. <=
C. &=
D. ==
E. =

What line of code could go in blank (f) ?

AA. weeks_before_frost
B. catalog.first.weeks
C. weeks_before_frost - catalog.first.weeks
D. weeks_before_frost - 1
EE. weeks_before_frost + 1

What line of code could go in blank (g) ?

FF. weeks_before_frost
B. catalog.first.weeks
C. weeks_before_frost - catalog.first.weeks
D. weeks_before_frost - 1
E. weeks_before_frost + 1

Suppose you want to use inheritance to create classes of specific types of seeds. Which of the following would be valid Python classes that inherit from the `Seed` class? — choose all options that could work

A.
```
class Basil(Seed):
    name = 'basil'
    weeks = 5
    def __init__(self):
        super().__init__(Basil.name, Basil.weeks)
```

B.
```
class Tomatillo(Seed):
    def __init__(self):
        super().__init__('tomatillo', 0)
```

C.
```
class Cucumber(Seed):
    def __init__(self):
        self.name = 'cucumber'
        self.weeks = 3
```

Calculate this

We are throwing it back to the calculator project (project three). Read the code below and answer the following questions.

```
class nil:
    """The empty list"""
    def __len__(self):
        return 0
nil = nil()
class Pair:
    """A pair has two instance attributes: first and rest. rest
       must be a Pair or nil
    """
    def __init__(self, first, rest=nil):
        self.first = first
        self.rest = rest
```

This function prints a calculator expression from the syntax tree:

```
def expression_string(tree):
  """ Given an expression tree of Pair objects, return the
original
  calculator expression
  >>> exp = Pair('+', Pair(Pair('*', Pair(3, Pair(2, nil))),
Pair(1, nil)))
  >>> expression_string(exp)
  '(+ (* 3 2) 1)'
  """
  s = f"({tree.first}"
  node = tree.rest
  while node != nil:
      if isinstance(node.first, Pair):
```

```
            s += f" {expression_string(node.first)}"
        else:
            s += f" {node.first}"
        node = node.rest
    return s + ")"
```

What will be the output of the following expression:

```
exp = Pair('-', Pair(4, Pair(2, nil)))
expression_string(exp)
```

A. (- 4 2)

B. (4 - 2)

C. (- (4 2))

D. (- Pair(4, Pair(2, nil)))

E. ('- 4 2')

How many Pair instances are created by this line:

```
exp = Pair('+', Pair(Pair('*', Pair(2, Pair(3, nil))), Pair(5, nil)))
```

A. 3

B. 4

C. 5

D. 6

E. 7

Which of the following expressions accesses the number 3 in the tree below:

```
exp = Pair('+', Pair(Pair('*', Pair(2, Pair(3, nil))), Pair(5, nil)))
```

A. exp.first.rest.rest.first

B. exp.rest.first.rest.first

C. exp.rest.rest.rest.first

D. exp.rest.first.rest.rest.first

E. exp.rest.first.rest.first

What will be the output of this code:

```
exp = Pair('*', Pair(Pair('+', Pair(1, Pair(2, nil))), nil))
expression_string(exp)
```

A. (* (+ 1 2))

B. (* 1 2)

C. (* Pair(1, 2))

D. (1 + 2)

E. (* (+ Pair(1, Pair(2, nil))))

What type must tree.rest always be in a valid expression?

A. int

B. str

C. Pair or nil

D. list

E. callable