This 90-minute exam has 6 questions worth a total of 100 points. Scan the whole test before starting. Budget your time wisely. Use the back of the pages if you need more space. You may tear the pages apart; we have a stapler at the front of the room.

It is a violation of the Academic Integrity Code to look at any exam other than your own, to look at any other reference material, or to otherwise give or receive unauthorized help.

You will be expected to write Python code on this exam. We recommend that you draw vertical lines to make your indentation clear, as follows:

```python
def foo():
    if something:
        do something
        do more things
    do something last
```

You should not use recursion on this exam. Beyond that, you may use any Python feature that you have learned about in class (if-statements, try-except, lists, for-loops and so on), unless directed otherwise.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

The Important First Question:

1. [2 points] Write your last name, first name, netid, and lab section at the top of each page.
Reference Sheet

Throughout this exam you will be asked questions about strings and lists. You are expected to understand how slicing works. In addition, the following functions and methods may be useful.

Math Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqrt(x)</td>
<td>Returns: The square root of x as a float.</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>Returns: The smallest integer value greater than or equal to x as a float.</td>
</tr>
<tr>
<td>floor(x)</td>
<td>Returns: The largest integer value less than or equal to x as a float.</td>
</tr>
</tbody>
</table>

All of these functions are in the module math. This module must be imported to access them.

String Functions and Methods

<table>
<thead>
<tr>
<th>Function or Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(s)</td>
<td>Returns: number of characters in s; it can be 0.</td>
</tr>
<tr>
<td>a in s</td>
<td>Returns: True if the substring a is in s; False otherwise.</td>
</tr>
<tr>
<td>s.find(s1)</td>
<td>Returns: index of the first character of the FIRST occurrence of s1 in s (-1 if s1 does not occur in s).</td>
</tr>
<tr>
<td>s.find(s1,n)</td>
<td>Returns: index of the first character of the first occurrence of s1 in s STARTING at position n. (-1 if s1 does not occur in s from this position).</td>
</tr>
<tr>
<td>s.count(s1)</td>
<td>Returns: number of (non-overlapping) occurrences of substring s1 in s.</td>
</tr>
<tr>
<td>s.isalpha()</td>
<td>Returns: True if s is not empty and its elements are all letters; it returns False otherwise.</td>
</tr>
<tr>
<td>s.isdigit()</td>
<td>Returns: True if s is not empty and its elements are all numbers; it returns False otherwise.</td>
</tr>
<tr>
<td>s.strip()</td>
<td>Returns: A copy of s with all spaces at either the beginning or end removed from s.</td>
</tr>
</tbody>
</table>

List Functions and Methods

<table>
<thead>
<tr>
<th>Function or Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(x)</td>
<td>Returns: number of elements in list x; it can be 0.</td>
</tr>
<tr>
<td>y in x</td>
<td>Returns: True if y is in list x; False otherwise.</td>
</tr>
<tr>
<td>x.index(y)</td>
<td>Returns: index of the FIRST occurrence of y in x (an error occurs if y does not occur in x).</td>
</tr>
<tr>
<td>x.count(y)</td>
<td>Returns: number of times y appears in the list x.</td>
</tr>
<tr>
<td>x.append(y)</td>
<td>Adds y to the end of list x.</td>
</tr>
<tr>
<td>x.insert(i,y)</td>
<td>Inserts y at position i in list x, shifting later elements to the right.</td>
</tr>
<tr>
<td>x.remove(y)</td>
<td>Removes the first item from the list whose value is y. (an error occurs if y does not occur in x).</td>
</tr>
</tbody>
</table>

The last three list methods are all procedures. They return the value None.
2. [20 points total] **Short Answer Questions.**

(a) [6 points] What are the four kinds of variables introduced in class? Define each of them.

A *global variable* is any variable created by an assignment statement outside of the body of a function.

A *local variable* is any variable created by an assignment statement inside of the body of a function.

A *parameter* is any variable within the parentheses of a function header.

An *attribute* is any variable inside of an object folder.

(b) [3 points] What is a *fruitful function*? What is a *procedure*? Your answer should explain how they differ and how they are the same.

Both fruitful functions and procedures are function. However, a fruitful function returns a value while a procedure does not (it returns nothing).

(c) [3 points] Which of the following two expressions produces an error (none, one, or both)?

\[
\text{False and (5 / 0)} \quad \text{or} \quad (5 / 0) \text{ or True}
\]

Explain why this is the case.

Only the second one produces an error. The first one does not produce an error because Python “short-circuits” the calculation; once it knows that the first value of an and is False, it does not need to evaluate anything else. But it must always evaluate the first expression, which is why the second one produces an error.

(d) [4 points] What is the definition of a *type cast*? What is the difference between a *widening* cast and a *narrowing* cast? Give an example of each.

A *type cast* is the conversion of a value from one type to another. A widening cast converts the type from one with less information to more information (e.g. int to float). An example is float(2) or 3/2.0. A narrowing cast is the reverse – from more information to less. An example is int(2.3).
(e) [4 points] What is the difference between the following two commands?
>>> import math
>>> from math import *

Your answer should describe the affects of each command on global space.

The first command creates a global variable with the name `math`. It also creates a folder with the contents of the module, and stores it in heap space. The variable `math` stores the name of this folder.

The second command takes each function and variable in the module `math` and creates a corresponding variable in global space. The function variables store a name of a folder in heap spaces storing the function definition.

(a) [8 points] Unlike many other languages, lists in Python can contain a mixture of types. For example, the following is legal in Python

```python
s = [1, True, 'Hello', 3.0]
```

This list contains an int, a bool, a string and a float.

For reasons we will see later, we often prefer that the elements of our lists all have the same type. That is why we might have a function like the following:

```python
def extract_int(seq):
    Return: a list containing only the ints in seq.
    The function returns a new list that contains only those elements of seq that are ints, in the same order they occur in seq.
    Example: extract_int([1, True, 'A', 2]) is [1,2]
    Precondition: seq is a list
```

Do not implement this function. Instead, write down a list of at least four test cases that you would use to test out this function. By a test case, we just mean an input and an expected output; you do not need to write an `assert_equals` statement. For each test case explain why it is significantly different from the others.

There are many different possible answers to this question. Below are the different solutions we were thinking of. If you had (at least) four test cases that were close to the ones below, you got full credit. Otherwise, we checked if your test cases were different enough, and awarded you 2 points for each test.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>[]</td>
<td>The list <code>seq</code> is empty</td>
</tr>
<tr>
<td>[1,'Hello',True]</td>
<td>[1]</td>
<td>List with just one int (and a mixture of non-ints)</td>
</tr>
<tr>
<td>[1,'Hello',2,True,3]</td>
<td>[1,2,3]</td>
<td>List with multiple ints (and a mixture of non-ints)</td>
</tr>
<tr>
<td>[1,2,3]</td>
<td>[1,2,3]</td>
<td>List with all ints</td>
</tr>
<tr>
<td>[True,False]</td>
<td>[]</td>
<td>List with no ints</td>
</tr>
</tbody>
</table>
(b) [12 points] In class, we demonstrated the function \texttt{anglicize}, which turns an int in its English equivalent. Listed below is an alternate implementation for the number 100 to 999, called \texttt{ang100to999}. This function has several helpers to aid it. These functions are mostly correct. However there are at least three bugs in this code. These bugs are across all functions and are not limited to a single function.

To help find the bugs, we have added several print statements throughout the code. The result of running the code with these print statements shown on the next page. Using this information as a guide, identify and fix the three bugs on the next page. You should explain your fixes.

\textbf{Hint:} Do not “fix” the lines marked NOT A BUG. It is okay to break up a list this way. In addition, we recommend that you start with the test on the next page, and do not try to read the code without using the tests.

\textbf{Code:}

```python
def ang100to999(n):
    """Returns: English equiv of n.
    Precond: n an int in 100..999"
    pref = ang1to19(n/100) + ' hundred'
    print 'Pref:' + pref  # WATCH
    tens = n % 100
    print 'Tens:' + str(tens)  # WATCH
    if tens > 0 and tens < 20:
        print 'In if-statement'  # TRACE
        suff = ' ' + ang1to19(tens)
        print 'Suff:' + suff  # TRACE
    elif tens > 20:
        print 'In elif-statement'  # TRACE
        suff = ' ' + ang20to99(tens)
        print 'Suff:' + suff  # TRACE
    return pref + suff

def ang20to99(n):
    """Returns: English equiv of n.
    Precond: n an int in 20..99"
    print 'ang20to99'  # TRACE
    pref = tens(n/10)
    print 'Pref:' + pref  # WATCH
    suff = ' ' + ang1to19(n % 10)
    print 'Suff:' + suff  # WATCH
    return pref + suff

def tens(n):
    """Returns: tens-word for n
    Precond: n an int in 2..9"

def ang1to19(n):
    """Returns: English equiv of n.
    Precond: n an int in 1..19"
```

Page 5
First Bug:
The bug for the first test is in \texttt{tens(n)}. The values of \texttt{n} are 2..9, but the list starts at 0. Therefore, we need to change Line 43 to:

\begin{verbatim}
result = words[n-2]
\end{verbatim}

Second Bug:
The bug for the second test is in \texttt{ang100to999(n)}. The variable \texttt{suff} is not defined if \texttt{tens} is 20. Therefore, we need to change Line 14 to:

\begin{verbatim}
elif tens >= 20:
\end{verbatim}

Third Bug:
The bug for the third test is in \texttt{ang20to99(n)}. If \texttt{n} \% 10 is 0, this violates the precondition of \texttt{ang1to19(n)}. Therefore, we need to rewrite the function as follows:

\begin{verbatim}
def anglicize20to99(n):
    Returns: English equiv of n.
    Precond: n an int in 20..99
    print 'ang20to99'
    pref = tens(n/10)
    print 'Pref: ' +pref
    if n \% 10 == 0:
        return prefix
    suff = ' ' +ang1to19(n \% 10)
    print 'Suff: ' +suff
    return pref+suff
\end{verbatim}
4. [16 points] **String Slicing.** When announcing the release date for a new product, companies have to pay attention to the official date format. In the US our date format is mm/dd/yy (e.g. month, day, year). However, Europe often uses the format dd/mm/yy (e.g. day, month, year). For example, a product released on 3/6/16 in the US might be released 9/3/16 in Europe, which is just 3 days later.

As shown in the example above, sometimes the month and day may only be a single digit. However, we typically pad these out to two digits with a leading zero. With this in mind, complete the function below using what you know about strings.

```python
def europeanize(date):
    Returns: European version of this date (type is a string).
    Days and months are padded (if necessary) to become two digits each.
    Examples:
        europeanize('3/6/12') is '06/03/12'
        europeanize('01/29/11') is '29/01/11'
    Precondition: date a string representing a US date.
    # Find dividers
    first = date.find('/')
    secnd = date.find('/', first + 1)

    # Get components
    month = date[:first]
    day = date[first+1:secnd]
    year = date[secnd+1:]

    # Pad if necessary
    if len(month) < 2:
        month = '0'+month

    if len(day) < 2:
        day = '0'+day

    # Put back together
    return day+'/'+'month+'+'year'
```
Consider the following function definitions.

```python
1 def cutout(seq):
2     pos = lookup(seq)
3     a = seq[:pos]
4     b = seq[pos+1:]
5     return a+b
6
def lookup(it):
7     return it[0]
```

Assume that `p = [2,0,-1,4]` is a global variable that stores a reference to a list in heap space, as shown below. On this page and the next, diagram the evolution of the call

```
p = cutout(p)
```

Diagram the state of the entire call stack for the function `cutout` when it starts, for each line executed, and when the frame is erased. If any other functions are called, you should do this for them as well (at the appropriate time). This will require a total of eight diagrams, not including the first one below.

In addition, you should draw the state of global space and heap space at each step. You can ignore the folders for the function definitions. Only draw folders for lists or objects. You are also allowed to write “unchanged” if no changes were made to either global or heap space.

**Hint**: Pay close attention to the line numbers. They are different than those in the assignment.
6. [20 points total] Objects and Functions.

Two dimensional vectors have a lot of applications in computer graphics. A vector \( \mathbf{v} \) is defined by its components \((v_x, v_y)\), as shown in the picture below.

There are several important functions on vectors.

**Dot Product and Norm.** The dot-product takes two vectors and combines them into single value, as given by the following formula.

\[
\mathbf{v} \cdot \mathbf{u} = v_x u_x + v_y u_y
\]

The length (or norm) of vector is the square root of the dot-product with itself, as per the following formula.

\[
||\mathbf{v}|| = \sqrt{\mathbf{v} \cdot \mathbf{v}} = \sqrt{v_x^2 + v_y^2}
\]

The norm is nonzero so long as either \(v_x\) and \(v_y\) are non-zero.

**Unit Vector.** For any given vector \( \mathbf{v} \), the unit vector \( \hat{\mathbf{v}} \) is the vector with the same direction as \( \mathbf{v} \), but with a length of 1. It is defined by the following formula.

\[
\hat{\mathbf{v}} = \left( \frac{v_x}{||\mathbf{v}||}, \frac{v_y}{||\mathbf{v}||} \right)
\]

Note that this only defined if \( \mathbf{v} \) is not the zero vector.

**Vector Projection.** For any two vector \( \mathbf{v} \) and \( \mathbf{u} \), the projection of \( \mathbf{u} \) onto \( \mathbf{v} \) is written \( \text{proj}_\mathbf{v} \mathbf{u} \). This is a vector with same direction of \( \mathbf{v} \), but whose length forms a right angle with the vector \( \mathbf{u} \), as shown in picture (c) above. We compute the projection with the following formula:

\[
\text{proj}_\mathbf{v} \mathbf{u} = \left( \frac{v_x(u \cdot v)}{v \cdot v}, \frac{v_y(u \cdot v)}{v \cdot v} \right)
\]

Again, this assumes that \( \mathbf{v} \) is not zero.

In this problem you will be working with the class \texttt{Vec2} which provides two dimension vectors in Python. This is a class with exactly two attributes – \( x \) and \( y \) – specifying the vector components. The only invariant for these attributes is that they must be floats. You create a new \texttt{Vec2} with the constructor function \texttt{Vec2(x, y)} which assigns the attributes in that order.

One the next page, you will implement functions computing the unit vector and vector projection. You can assume that the module providing \texttt{Vec2} is already imported, and you do not need to use the module as a prefix when calling the constructor (for example, you can just call \texttt{Vec2(1, 2)}. Implement these functions exactly as specified.
(a) [10 points] **Unit Vector**

```python
def normalize(v):
    """Changes the vector v into a unit vector in the same direction."
    """This function is a procedure. It does not return a new vector. It modifies the x and y attributes of the parameter v. If v is a zero vector, the function leaves it unchanged.
    Preconditions: v is a Vec2 object.
    # Compute the norm
    nmsq = v.x*v.x+v.y*v.y
    norm = math.sqrt(nmsq)
    if norm == 0:
        return # Do nothing
    # Adjust the vector
    v.x = v.x/norm
    v.y = v.y/norm
```

(b) [10 points] **Vector Projection**

```python
def project(u,v):
    """Returns: the projection of u onto v."
    """This function returns a new vector and does not modify either u or v. If v is the zero vector, then it returns a new zero vector.
    Preconditions: u and v are Vec2 objects.
    # Dot product of u and v
    dot = u.x*v.x+u.y*v.y
    # Square of the norm
    nmsq = v.x*v.x+v.y*v.y
    # Handle the zero vector
    if nmsq == 0:
        return Vec2(0.0,0.0)
    # Compute the result
    result = Vec2(v.x*dot/nmsq,v.y*dot/nmsq)
    return result
```