CSC148H Week 5

February 6, 2015

Announcements

- Tutorial
  - Friday at 17:00 in IB110
  - Term test 1 prep
- Term Test 1
  - Next Friday at 17:00 in IB110

Palindrome

- What are properties of a palindrome?
- How do we access the first and last letter?

```
def isPalindrome(s):
    '''(str) => bool
    Returns True if s is a palindrome and False otherwise

    >>> isPalindrome('')
    True

    >>> isPalindrome('z')
    True

    >>> isPalindrome('radar')
    True

    >>> isPalindrome('level')
    True
    '''
```

Palindrome

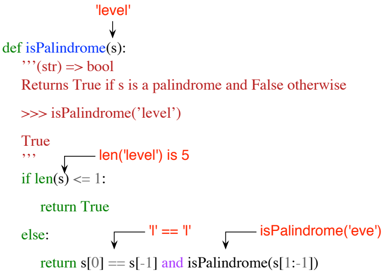
```
def isPalindrome(s):
    ’’’(str) => bool
    Returns True if s is a palindrome and False otherwise

    >>> isPalindrome(’’)
    True

    >>> isPalindrome(’z’)
    True

    >>> isPalindrome(’radar’)
    True

    >>> isPalindrome(’level’)
    True
    ’’’
    if len(s) <= 1:
        return True
    else:
        return s[0] == s[-1] and isPalindrome(s[1:-1])
```

Palindrome Trace

```
def isPalindrome(s):
    '''(str) => bool
    Returns True if s is a palindrome and False otherwise

    >>> isPalindrome('level')

    True
    '''
    if len(s) <= 1:

        return True

    else:

        return s[0] == s[-1] and isPalindrome(s[1:-1])
```

Evaluate when s is level

```
>>> isPalindrome('level')
True
```



'level'

```
def isPalindrome(s):
    '''(str) => bool
    Returns True if s is a palindrome and False otherwise

    >>> isPalindrome('level')

    True
    '''
    if len(s) <= 1:

        return True

    else:

        return s[0] == s[-1] and isPalindrome(s[1:-1])
```

len('level') is 5

'l' == 'l'

isPalindrome('eve')

Palindrome Trace

'eve'

```
def isPalindrome(s):
    '''(str) => bool
    Returns True if s is a palindrome and False otherwise

    >>> isPalindrome('level')

    True
    '''
    if len(s) <= 1:

        return True

    else:

        return s[0] == s[-1] and isPalindrome(s[1:-1])
```

len('eve') is 3

'e' == 'e'

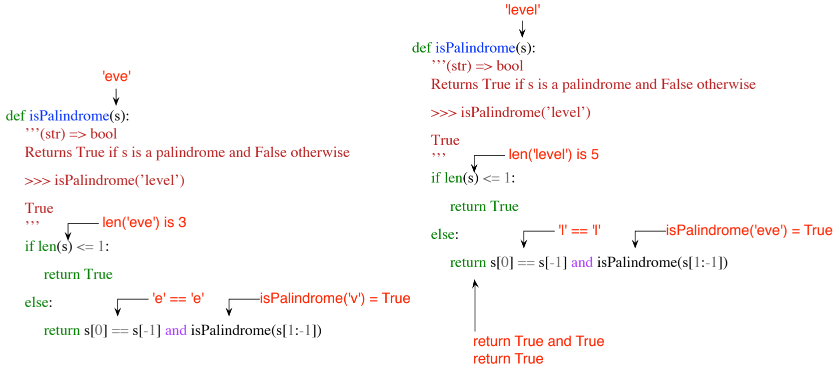
isPalindrome('v')

'v'

```
def isPalindrome(s):
    '''(str) => bool
    Returns True if s is a palindrome and False otherwise

    >>> isPalindrome('level')

    True
    '''
    if len(s) <= 1:

        return True

    else:

        return s[0] == s[-1] and isPalindrome(s[1:-1])
```

len('v') is 1

Palindrome Trace



'eve'

```
def isPalindrome(s):
    '''(str) => bool
    Returns True if s is a palindrome and False otherwise

    >>> isPalindrome('level')

    True
    '''
    if len(s) <= 1:

        return True

    else:

        return s[0] == s[-1] and isPalindrome(s[1:-1])
```

len('eve') is 3

'e' == 'e'

isPalindrome('v') = True

'level'

```
def isPalindrome(s):
    '''(str) => bool
    Returns True if s is a palindrome and False otherwise

    >>> isPalindrome('level')

    True
    '''
    if len(s) <= 1:

        return True

    else:

        return s[0] == s[-1] and isPalindrome(s[1:-1])
```

len('level') is 5

'l' == 'l'

isPalindrome('eve') = True

return True and True
return True

English Ruler

```
---- 0        ---- 0        --- 0
-             -             -
--            --            --
-             -             -
---           ---           --- 1
-             -             -
--            --            --
-             -             -
---- 1        ----          --- 2
-             -             -
--            --            --
-             -             -
---           ---           --- 3
-             -
--            --
-             -
---- 2        ---- 1

(a)           (b)           (c)
```

(a) a 2-inch ruler with major tick length 4;

(b) a 1-inch ruler with major tick length 5;

(c) a 3-inch ruler with major tick length 3

- We denote the length of the tick designating a whole inch as the **major tick length**.
- Between the marks for whole inches, the ruler contains a series of **minor ticks**, placed at intervals of 1/2 inch, 1/4 inch, and so on.
- As the size of the interval decreases by half, the tick length decreases by one.
- In general, an interval with a central tick length L 1 is composed of:
  - An interval with a central tick length L - 1
  - A single tick of length L
  - An interval with a central tick length L - 1

English Ruler Strategey

- Ruler depends on the total size and major tick length
- Need a function to draw the intervals
- Need a function to draw the lines
- Need a function to draw the ruler

English Ruler

```
class EnglishRuler:

    def __init__(self, ...):
     pass

    def draw_line(self, ...):
     pass

    def draw_interval(self, ...):
     pass

    def draw_ruler(self, ...):
     pass
```

English Ruler

```python
class EnglishRuler:

    def __init__(self, num_inches, major_length):
        # notice the two underscores
        self.__num_inches = num_inches
        self.__major_length = major_length
```

English Ruler

```
class EnglishRuler:

    def __init__(self, num_inches, major_length):
        # notice the two underscores
        self.__num_inches = num_inches
        self.__major_length = major_length

    def draw_line(self, tick_length, tick_label=''):
        '''Draw one line with given tick length
        (followed by optional label).'''
```

English Ruler

```python
class EnglishRuler:

    def __init__(self, num_inches, major_length):
        # notice the two underscores
        self.__num_inches = num_inches
        self.__major_length = major_length

    def draw_line(self, tick_length, tick_label=''):
        '''Draw one line with given tick length
        (followed by optional label).'''
        line = '-'*tick_length
        if tick_label:
            line += ' '+tick_label
        print(line)
```

English Ruler

```
class EnglishRuler:
    def __init__(self, num_inches, major_length):
        # notice the two underscores
        self.__num_inches = num_inches
        self.__major_length = major_length

    def draw_line(self, tick_length, tick_label=''):
        '''Draw one line with given tick length
        (followed by optional label).'''
        line = '-'*tick_length
        if tick_label:
            line += ' '+tick_label
        print(line)

    def draw_interval(self, center_length):
        '''Draw tick interval based upon a central
        tick length.'''
```

English Ruler

```
class EnglishRuler:
    def __init__(self, num_inches, major_length):
        # notice the two underscores
        self.__num_inches = num_inches
        self.__major_length = major_length

    def draw_line(self, tick_length, tick_label=’’):
        ’’’Draw one line with given tick length
        (followed by optional label).’’’
        line = ’-’*tick_length
        if tick_label:
            line += ’ ’+tick_label
        print(line)

    def draw_interval(self, center_length):
        ’’’Draw tick interval based upon a central
        tick length.’’’
        if center_length > 0:
            self.draw_interval(center_length - 1)
            self.draw_line(center_length)
            self.draw_interval(center_length - 1)
```

English Ruler

```
class EnglishRuler:
    def __init__(self, num_inches, major_length):
        # notice the two underscores
        self.__num_inches = num_inches
        self.__major_length = major_length

    def draw_line(self, tick_length, tick_label=''):
        '''Draw one line with given tick length
        (followed by optional label).'''
        line = '-'*tick_length
        if tick_label:
            line += ' '+tick_label
        print(line)

    def draw_interval(self, center_length):
        '''Draw tick interval based upon a central
        tick length.'''
        if center_length > 0:
            self.draw_interval(center_length - 1)
            self.draw_line(center_length)
            self.draw_interval(center_length - 1)

    def draw_ruler(self):
        '''Draw Enlish ruler with given number of inches
        major tick length.'''
```

English Ruler

```
class EnglishRuler:
    def __init__(self, num_inches, major_length):
        # notice the two underscores
        self.__num_inches = num_inches
        self.__major_length = major_length

    def draw_line(self, tick_length, tick_label=’’):
        ’’’Draw one line with given tick length
        (followed by optional label).’’’
        line = ’-’*tick_length
        if tick_label:
            line += ’ ’+tick_label
        print(line)

    def draw_interval(self, center_length):
        ’’’Draw tick interval based upon a central
        tick length.’’’
        if center_length > 0:
            self.draw_interval(center_length - 1)
            self.draw_line(center_length)
            self.draw_interval(center_length - 1)

    def draw_ruler(self):
        ’’’Draw Enlish ruler with given number of inches
        major tick length.’’’
        self.draw_line(self.__major_length, ’0’)
        for j in range(1, 1+self.__num_inches):
            self.draw_interval(self.__major_length-1)
            self.draw_line(self.__major_length, str(j))

if __name__ == ’__main__’:
    ruler = EnglishRuler(5, 5)
    ruler.draw_ruler()
```

Pascal's Triangle

- Notice the top value (hint: base case)
- Take note of the side values of the triangle
- How are the internal values calculated?

```
                  1
                1   1
              1   2   1
            1   3   3   1
          1   4   6   4   1
        1   5  10  10   5   1
      1   6  15  20  15   6   1
    1   7  21  35  35  21   7   1
```

Pascal's Triangle

```
def pascal(n):
    if n == 1:
        return [1]
    return
```

Pascal's Triangle

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)

        for i in range(len(previous_line)-1):
            line.append(previous_line[i] + previous_line[i+1])

        line += [1]
    return line
```

Pascal’s Triangle Trace

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)

        for i in range(len(previous_line)-1):

            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

Evaluate when n is 5

```
>>> pascal(5)
[1, 4, 6, 4, 1]
```

5

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)

        for i in range(len(previous_line)-1):

            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

previous_line = pascal(4)

Pascal’s Triangle Trace

4

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)

        for i in range(len(previous_line)-1):

            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

previous_line = pascal(3)

3

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)

        for i in range(len(previous_line)-1):

            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

previous_line = pascal(2)

Pascal's Triangle Trace

2

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)

        for i in range(len(previous_line)-1):

            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

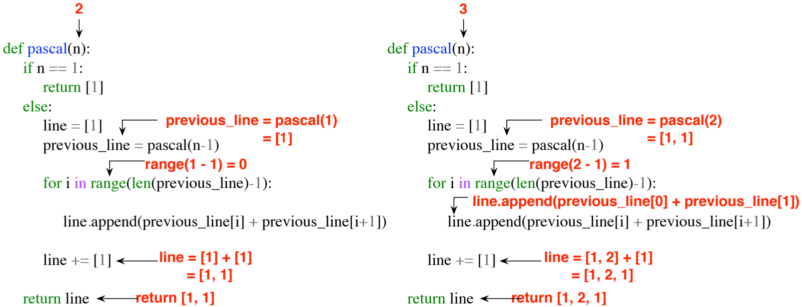
previous_line = pascal(1)

1

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)

        for i in range(len(previous_line)-1):

            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

return [1]

Pascal's Triangle Trace



2

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)
        for i in range(len(previous_line)-1):

            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

previous_line = pascal(1)
= [1]

range(1 - 1) = 0

line = [1] + [1]
= [1, 1]

return [1, 1]

3

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)
        for i in range(len(previous_line)-1):
            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

previous_line = pascal(2)
= [1, 1]

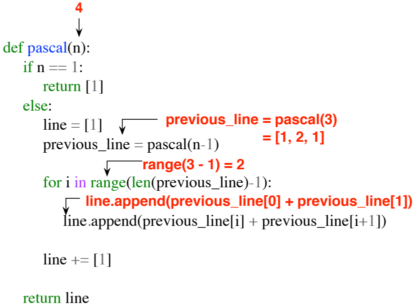
range(2 - 1) = 1

line.append(previous_line[0] + previous_line[1])

line = [1, 2] + [1]
= [1, 2, 1]

return [1, 2, 1]

Pascal's Triangle Trace



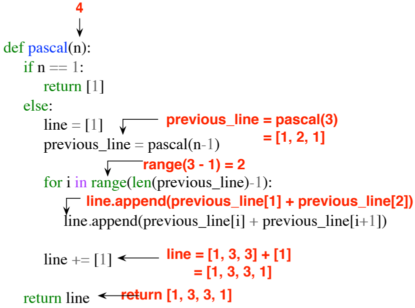
4

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)
        for i in range(len(previous_line)-1):
            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

previous_line = pascal(3)
= [1, 2, 1]

range(3 - 1) = 2

line.append(previous_line[0] + previous_line[1])



4

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)
        for i in range(len(previous_line)-1):
            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

previous_line = pascal(3)
= [1, 2, 1]

range(3 - 1) = 2

line.append(previous_line[1] + previous_line[2])

line = [1, 3, 3] + [1]
= [1, 3, 3, 1]

return [1, 3, 3, 1]

Pascal’s Triangle Trace

5

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)
        for i in range(len(previous_line)-1):
            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

previous_line = pascal(4)
= [1, 3, 3, 1]

range(4 - 1) = 3

line.append(previous_line[0] + previous_line[1])

5

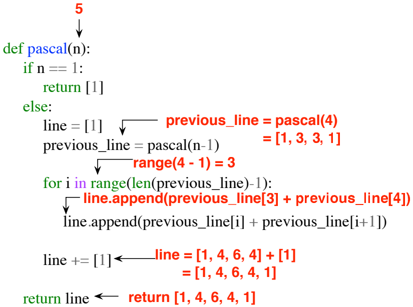
```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)
        for i in range(len(previous_line)-1):
            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

previous_line = pascal(4)
= [1, 3, 3, 1]

range(4 - 1) = 3

line.append(previous_line[1] + previous_line[2])

Pascal’s Triangle Trace



5

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)
        for i in range(len(previous_line)-1):
            line.append(previous_line[i] + previous_line[i+1])

        line += [1]

    return line
```

previous_line = pascal(4)
= [1, 3, 3, 1]

range(4 - 1) = 3

line.append(previous_line[3] + previous_line[4])

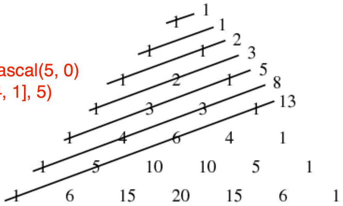
line = [1, 4, 6, 4] + [1]
= [1, 4, 6, 4, 1]

return [1, 4, 6, 4, 1]

Pascal's Triangle and Fibonacci Sequence

- Let's print out the Fibonacci sequence using Pascal's triangle

```
>>> fib(5)
5

>>> fib_pascal(5, 0)
([1, 4, 6, 4, 1], 5)
```



```
def fib(n):
    """int -> int
    Return the nth fibonacci number.
    Where the fibonacci numbers are defined as:
    1, 1, 2, 3, 5, 8, 13
    (each number is the sum of the two previous numbers)

    >>> fib(4)
    3
    """

    if n <= 2:
        return 1
    else:
        return fib(n-1)+fib(n-2)
```

Pascal's Triangle and Fibonacci Sequence

```python
def fib_pascal(n, fib_pos):
    if n == 1:
        line = [1]
        if fib_pos == 0:
            fib_sum = 1
        else:
            fib_sum = 0
    else:
        line = [1]
        (previous_line, fib_sum) = fib_pascal(n-1, fib_pos+1)

        for i in range(len(previous_line)-1):
            line.append(previous_line[i] + previous_line[i+1])

        line += [1]
        if fib_pos < len(line):
            fib_sum += line[fib_pos]

    return (line, fib_sum)

def fib(n):
    return fib_pascal(n,0)[1]
```

Sieve of Eratosthenes

- Algorithm for finding all prime numbers up to any given limit.
- Works by iteratively marking as composite (i.e., not prime) the multiples of each prime, starting with the multiples of 2.

2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	15
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	15
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~

2	**3**		**5**		**7**				**11**		**13**		

Sieve of Eratosthenes - Steps

1. Create a list of integers from two to n: 2, 3, 4, ..., n (why aren't we starting from 1?)
2. Start with a counter **i** set to 2 (first prime number)
3. Starting from **i + i**, count up by **i** and remove those numbers from the list, i.e. 2*i, 3*i, 4*i, and so on...
4. Find the first number of the list following **i**. This is the next prime number.
5. Set **i** to the number found in the previous step
6. Repeat steps 3 and 4 until **i** is greater than **n** or the square root of **n**. (Think of the definition of a composite number).
7. All the numbers, which are still in the list, are prime numbers

Sieve of Eratosthenes

```
def primes(n):
    '''(int) -> list of int
    Return all primes between 2 and n

    >>> primes(5)
    [2, 3, 5]
    '''
```