Announcements

- HW4 due Wednesday at 11:59pm

- Hog contest deadline next week
  - Completely optional, opportunity for extra credit
  - See website for details
Fibonacci Sequence

The Fibonacci sequence is defined as

\[
\text{fib}(n) = \begin{cases} 
0, & n = 0 \\
1, & n = 1 \\
\text{fib}(n - 1) + \text{fib}(n - 2), & n > 1 
\end{cases}
\]

Example: [http://goo.gl/DZbRG](http://goo.gl/DZbRG)
The Fibonacci sequence is defined as

\[
\text{fib}(n) = \begin{cases} 
0, & n = 0 \\
1, & n = 1 \\
\text{fib}(n - 1) + \text{fib}(n - 2), & n > 1 
\end{cases}
\]

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    return fib(n - 1) + fib(n - 2)
```

Example: [http://goo.gl/DZbRG](http://goo.gl/DZbRG)
Fibonacci Sequence

The Fibonacci sequence is defined as

\[
\text{fib}(n) = \begin{cases} 
0, & n = 0 \\
1, & n = 1 \\
\text{fib}(n - 1) + \text{fib}(n - 2), & n > 1 
\end{cases}
\]

def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    return fib(n - 1) + fib(n - 2)

Example: [http:// goo.gl/DZbRG](http://goo.gl/DZbRG)
The Fibonacci sequence is defined as:

\[
\text{fib}(n) = \begin{cases} 
0, & n = 0 \\
1, & n = 1 \\
\text{fib}(n - 1) + \text{fib}(n - 2), & n > 1 
\end{cases}
\]

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    return fib(n - 1) + fib(n - 2)
```

Example: [http://goo.gl/DZbRG](http://goo.gl/DZbRG)
Tree recursion
Tree recursion

Executing the body of a function may entail more than one recursive call to that function
Tree recursion

Executing the body of a function may entail more than one recursive call to that function
This is called *tree recursion*
Tree recursion

Executing the body of a function may entail more than one recursive call to that function
This is called *tree recursion*

```
fib(5)
```
Tree recursion

Executing the body of a function may entail more than one recursive call to that function
This is called \textit{tree recursion}
Tree recursion

Executing the body of a function may entail more than one recursive call to that function

This is called tree recursion
Tree recursion

Executing the body of a function may entail more than one recursive call to that function.
This is called *tree recursion*.
Tree recursion

Executing the body of a function may entail more than one recursive call to that function.

This is called *tree recursion*. 
Tree recursion

Executing the body of a function may entail more than one recursive call to that function

This is called *tree recursion*
Tree recursion

Executing the body of a function may entail more than one recursive call to that function

This is called *tree recursion*
Tree recursion

Executing the body of a function may entail more than one recursive call to that function.

This is called tree recursion.
Executing the body of a function may entail more than one recursive call to that function.
This is called *tree recursion*.
Tree recursion

Executing the body of a function may entail more than one recursive call to that function

This is called *tree recursion*
Executing the body of a function may entail more than one recursive call to that function.
This is called tree recursion.
Tree recursion

Executing the body of a function may entail more than one recursive call to that function

This is called *tree recursion*
Tree recursion

Executing the body of a function may entail more than one recursive call to that function.

This is called **tree recursion**.
Tree recursion

Executing the body of a function may entail more than one recursive call to that function

This is called *tree recursion*
Tree recursion

Executing the body of a function may entail more than one recursive call to that function

This is called tree recursion
Tree recursion

Executing the body of a function may entail more than one recursive call to that function.

This is called *tree recursion*.
Tree recursion

Executing the body of a function may entail more than one recursive call to that function
This is called tree recursion
Tree recursion

Executing the body of a function may entail more than one recursive call to that function

This is called *tree recursion*
Tree recursion

Executing the body of a function may entail more than one recursive call to that function

This is called tree recursion
Tree recursion

Executing the body of a function may entail more than one recursive call to that function
This is called tree recursion
Tree recursion

Executing the body of a function may entail more than one recursive call to that function.

This is called tree recursion.
Tracing the Order of Calls
Tracing the Order of Calls

We can use a higher-order function to see the order in which calls are made and complete
We can use a higher-order function to see the order in which calls are made and complete

```python
def trace1(fn):
    """Return a function equivalent to fn that also prints trace output."""
    def traced(x):
        print('Calling', fn, '(', x, ')')
        res = fn(x)
        print('Got', res, 'from', fn, '(', x, ')')
        return res
    return traced
```
Tracing the Order of Calls

We can use a higher-order function to see the order in which calls are made and complete

```python
def trace1(fn):
    """Return a function equivalent to fn that also prints trace output.""
    def traced(x):
        print('Calling', fn, '(', x, ')
        res = fn(x)
        print('Got', res, 'from', fn, '(', x, ')')
        return res
    return traced

# Rebind the name fib to a traced version of fib
fib = trace1(fib)
```
Function Decorators
@trace1
def triple(x):
    return 3 * x
Function Decorators

@trace1
def triple(x):
    return 3 * x
Function Decorators

@trace1
def triple(x):
    return 3 * x
Function Decorators

Function decorator

@trace1
def triple(x):
    return 3 * x

is identical to

Decorated function
Function Decorators

@trace1
def triple(x):
    return 3 * x

Decorated function

is identical to

def triple(x):
    return 3 * x
triple = trace1(triple)
Function Decorators

Function decorator

@trace1
def triple(x):
    return 3 * x

Decorated function

is identical to

Why not just use this?
def triple(x):
    return 3 * x
triple = trace1(triple)
The Recursive Leap of Faith
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)

Is factorial implemented correctly?
Is factorial implemented correctly?

1. Verify the base case.
The Recursive Leap of Faith

```python
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
```

Is factorial implemented correctly?

1. Verify the base case.

2. Treat `factorial(n-1)` as a functional abstraction.
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)

Is factorial implemented correctly?

1. Verify the base case.
2. Treat \texttt{factorial(n-1)} as a functional abstraction.
3. Assume that \texttt{factorial(n-1)} is correct.
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)

Is factorial implemented correctly?

1. Verify the base case.
2. Treat \texttt{factorial(n-1)} as a functional abstraction.
3. Assume that \texttt{factorial(n-1)} is correct.
Is factorial implemented correctly?

1. Verify the base case.
2. Treat \texttt{factorial(n-1)} as a functional abstraction.
3. Assume that \texttt{factorial(n-1)} is correct.
4. Verify that \texttt{factorial(n)} is correct, assuming that \texttt{factorial(n-1)} is correct.

\begin{verbatim}
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
\end{verbatim}
Is factorial implemented correctly?

1. Verify the base case.
2. Treat $\text{factorial}(n-1)$ as a functional abstraction.
3. Assume that $\text{factorial}(n-1)$ is correct.
4. Verify that $\text{factorial}(n)$ is correct, assuming that $\text{factorial}(n-1)$ is correct.

```python
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
```
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)

Is factorial implemented correctly?

1. Verify the base case.
2. Treat $\text{factorial}(n-1)$ as a functional abstraction.
3. Assume that $\text{factorial}(n-1)$ is correct.
4. Verify that $\text{factorial}(n)$ is correct, assuming that $\text{factorial}(n-1)$ is correct.
Simplifying a Problem
Simplifying a Problem

Pig Latinization:
Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word
Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart → arts may
Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart ➔ artsmay

```python
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```
Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart  \rightarrow  artsmay

def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))

smart
Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart  $\rightarrow$  artsmaay

```python
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))

smart  $\rightarrow$  marts
Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

\[
\text{smart} \rightarrow \text{artsmay}
\]

```python
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

\[
\text{smart} \rightarrow \text{marts} \rightarrow \text{artsm}
\]
Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart  →  artsmay

def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))

smart  →  marts  →  artsm  →  artsmay
Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

```
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

smart  →  artsmay

smart  →  marts  →  artsm  →  artsmay

2 consonants to be moved
Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart  →  artsmay

```python
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

smart  →  marts  →  artsm  →  artsmay

2 consonants to be moved
1 consonant to be moved
Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

\[ \text{smart} \rightarrow \text{artsmay} \]

```python
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

\[ \text{smart} \rightarrow \text{marts} \rightarrow \text{artsm} \rightarrow \text{artsmay} \]

2 consonants to be moved
1 consonant to be moved
Base case
Counting Change
Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05
Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1$ half dollar, 1 quarter, 2 dimes, 1 nickel
Counting Change

$1 = 0.50 + 0.25 + 0.10 + 0.10 + 0.05$

$1 = 1 \text{ half dollar, 1 quarter, 2 dimes, 1 nickel}$

$1 = 2 \text{ quarters, 2 dimes, 30 pennies}$
Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1$ half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2$ quarters, 2 dimes, 30 pennies

$1 = 100$ pennies
Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1 \text{ half dollar, 1 quarter, 2 dimes, 1 nickel}

$1 = 2 \text{ quarters, 2 dimes, 30 pennies}

$1 = 100 \text{ pennies}

How many ways are there to change a dollar?
Counting Change

$1 = 0.50 + 0.25 + 0.10 + 0.10 + 0.05$

$1 = 1$ half dollar, $1$ quarter, $2$ dimes, $1$ nickel

$1 = 2$ quarters, $2$ dimes, $30$ pennies

$1 = 100$ pennies

How many ways are there to change a dollar?

How many ways to change $0.11$?
Counting Change

$1 = \$0.50 + \$0.25 + \$0.10 + \$0.10 + \$0.05$

$1 = 1$ half dollar, $1$ quarter, $2$ dimes, $1$ nickel

$1 = 2$ quarters, $2$ dimes, $30$ pennies

$1 = 100$ pennies

How many ways are there to change a dollar?

How many ways to change $\$0.11$?

Use a
dime
Counting Change

$1 = \$0.50 + \$0.25 + \$0.10 + \$0.10 + \$0.05

$1 = 1 \text{ half dollar, 1 quarter, 2 dimes, 1 nickel}

$1 = 2 \text{ quarters, 2 dimes, 30 pennies}

$1 = 100 \text{ pennies}

How many ways are there to change a dollar?

How many ways to change $0.11?

Use a dime
Counting Change

$1 = \$0.50 + \$0.25 + \$0.10 + \$0.10 + \$0.05$

$1 = 1$ half dollar, $1$ quarter, $2$ dimes, $1$ nickel

$1 = 2$ quarters, $2$ dimes, $30$ pennies

$1 = 100$ pennies

How many ways are there to change a dollar?

How many ways to change $\$0.11$?

Use a dime

10 1
Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1 \text{ half dollar}, 1 \text{ quarter}, 2 \text{ dimes}, 1 \text{ nickel}

$1 = 2 \text{ quarters}, 2 \text{ dimes}, 30 \text{ pennies}

$1 = 100 \text{ pennies}

How many ways are there to change a dollar?

How many ways to change $0.11?

<table>
<thead>
<tr>
<th>Use a dime</th>
<th>No dimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1$ half dollar, 1$ quarter, 2$dimes, 1$nickel

$1 = 2$quarters, 2$dimes, 30$pennies

$1 = 100$pennies

How many ways are there to change a dollar?

How many ways to change $0.11?

<table>
<thead>
<tr>
<th>Use a dime</th>
<th>No dimes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Use a nickel</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
### Counting Change

$1 = \$0.50 + \$0.25 + \$0.10 + \$0.10 + \$0.05$

$1 = 1$ half dollar, $1$ quarter, $2$ dimes, $1$ nickel

$1 = 2$ quarters, $2$ dimes, $30$ pennies

$1 = 100$ pennies

How many ways are there to change a dollar?

How many ways to change $\$0.11$?

<table>
<thead>
<tr>
<th>Use a dime</th>
<th>No dimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use a nickel</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
### Counting Change

\[ \$1 = \$0.50 + \$0.25 + \$0.10 + \$0.10 + \$0.05 \]

\[ \$1 = \text{1 half dollar, 1 quarter, 2 dimes, 1 nickel} \]

\[ \$1 = \text{2 quarters, 2 dimes, 30 pennies} \]

\[ \$1 = \text{100 pennies} \]

**How many ways are there to change a dollar?**

**How many ways to change $0.11?**

<table>
<thead>
<tr>
<th>Use a dime</th>
<th>No dimes</th>
<th>Use a nickel</th>
</tr>
</thead>
<tbody>
<tr>
<td>10d</td>
<td>10n</td>
<td>10c</td>
</tr>
<tr>
<td>10d</td>
<td>10n</td>
<td>10c</td>
</tr>
<tr>
<td>5d</td>
<td>10n</td>
<td>10c</td>
</tr>
<tr>
<td>5d</td>
<td>10n</td>
<td>10c</td>
</tr>
<tr>
<td>5d</td>
<td>10n</td>
<td>10c</td>
</tr>
<tr>
<td>5d</td>
<td>10n</td>
<td>10c</td>
</tr>
<tr>
<td>5d</td>
<td>10n</td>
<td>10c</td>
</tr>
<tr>
<td>5d</td>
<td>10n</td>
<td>10c</td>
</tr>
<tr>
<td>5d</td>
<td>10n</td>
<td>10c</td>
</tr>
<tr>
<td>5d</td>
<td>10n</td>
<td>10c</td>
</tr>
</tbody>
</table>
## Counting Change

$1 = \$0.50 + \$0.25 + \$0.10 + \$0.10 + \$0.05$

$1 = 1$ half dollar, $1$ quarter, $2$ dimes, $1$ nickel

$1 = 2$ quarters, $2$ dimes, $30$ pennies

$1 = 100$ pennies

### How many ways are there to change a dollar?

### How many ways to change $\$0.11$?

<table>
<thead>
<tr>
<th>Use a dime</th>
<th>No dimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use a nickel</td>
<td>No nickles</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1$ half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2$ quarters, 2 dimes, 30 pennies

$1 = 100$ pennies

How many ways are there to change a dollar?

How many ways to change $0.11$?

<table>
<thead>
<tr>
<th>Use a dime</th>
<th>Use a nickel</th>
<th>No dimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1111</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11111</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>111111</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1111111</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11111111</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>111111111</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1111111111</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11111111111</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>111111111111</td>
<td></td>
</tr>
</tbody>
</table>
Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1$ half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2$ quarters, 2 dimes, 30 pennies

$1 = 100$ pennies

How many ways are there to change a dollar?

How many ways to change $0.11$?

<table>
<thead>
<tr>
<th>Use a dime</th>
<th>Use a nickel</th>
<th>No dimes</th>
<th>No nickles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ways to make 1 cent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5 5 1</td>
<td>1 1 1 1 1 1 1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>
### Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05$

$1 = 1$ half dollar, $1$ quarter, $2$ dimes, $1$ nickel

$1 = 2$ quarters, $2$ dimes, $30$ pennies

$1 = 100$ pennies

---

**How many ways are there to change a dollar?**

**How many ways to change $0.11$?**

<table>
<thead>
<tr>
<th>Use a dime</th>
<th>Use a nickel</th>
<th>No dimes</th>
<th>No nickles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ways to make 6 cents using no dimes</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Ways to make $1$ cent:

- No dimes: 1 way
- No nickles: 1 way

Ways to make $6$ cents:

- No dimes: 1 way
- No nickles: 1 way

---

**Cal**
How many ways are there to change a dollar?

<table>
<thead>
<tr>
<th>Use a dime</th>
<th>Use a nickel</th>
<th>No dimes</th>
<th>No nickles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use a nickel</td>
<td>Ways to make 1 cent</td>
<td>Ways to make 6 cents using no dimes</td>
<td></td>
</tr>
</tbody>
</table>
# Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount $a$ using $n$ kinds of coins is:

<table>
<thead>
<tr>
<th>Use a</th>
<th>Use a nickel</th>
<th>No dimes</th>
<th>No nickles</th>
</tr>
</thead>
<tbody>
<tr>
<td>dime</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Ways to make 6 cents using no dimes
Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount $a$ using $n$ kinds of coins is:
1. The number of ways to change $a-d$ using all kinds, where $d$ is the amount of the first kind of coin.
How many ways are there to change a dollar?

The number of ways to change an amount $a$ using $n$ kinds of coins is:

1. The number of ways to change $a-d$ using all kinds, where $d$ is the amount of the first kind of coin
2. The number of ways to change $a$ using all but the first kind of coin

### Ways to make 6 cents using no dimes

<table>
<thead>
<tr>
<th>Use a dime</th>
<th>No dimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use a nickel</td>
<td>No nickles</td>
</tr>
<tr>
<td>10 1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>5 5 1</td>
<td></td>
</tr>
<tr>
<td>5 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

Ways to make 1 cent: 1

Ways to make 5 cents: 1

Ways to make 10 cents: 1

Ways to make 15 cents: 1

Ways to make 20 cents: 1

Ways to make 25 cents: 1

Ways to make 30 cents: 1

Ways to make 35 cents: 1

Ways to make 40 cents: 1

Ways to make 45 cents: 1

Ways to make 50 cents: 1

Ways to make 55 cents: 1

Ways to make 60 cents: 1

Ways to make 65 cents: 1

Ways to make 70 cents: 1

Ways to make 75 cents: 1

Ways to make 80 cents: 1

Ways to make 85 cents: 1

Ways to make 90 cents: 1

Ways to make 95 cents: 1

Ways to make 100 cents: 1
How many ways are there to change a dollar?

The number of ways to change an amount $a$ using $n$ kinds of coins is:

1. The number of ways to change $a-d$ using all kinds, where $d$ is the amount of the first kind of coin
2. The number of ways to change $a$ using all but the first kind
Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount \( a \) using \( n \) kinds of coins is:

1. The number of ways to change \( a-d \) using all kinds, where \( d \) is the amount of the first kind of coin
2. The number of ways to change \( a \) using all but the first kind

```python
def count_change(a, d):
    if a == 0:
        return 1
    if a < 0 or d == 0:
        return 0
    return (count_change(a-d, d) +
            count_change(a, next_coin(d)))
```
Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount \( a \) using \( n \) kinds of coins is:

1. The number of ways to change \( a-d \) using all kinds, where \( d \) is the amount of the first kind of coin
2. The number of ways to change \( a \) using all but the first kind

\[
def \text{count_change}(a, d):
    \text{if } a == 0:
        \text{return 1}
    \text{if } a < 0 \text{ or } d == 0:
        \text{return 0}
    \text{return (count_change}(a-d, d) +
            \text{count_change}(a, \text{next_coin}(d)))
\]
Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount $a$ using $n$ kinds of coins is:
1. The number of ways to change $a-d$ using all kinds, where $d$ is the amount of the first kind of coin
2. The number of ways to change $a$ using all but the first kind

```python
def count_change(a, d):
    if a == 0:
        return 1
    if a < 0 or d == 0:
        return 0
    return (count_change(a-d, d) + count_change(a, next_coin(d)))
```

One way to make no amount

Can’t make negative amount, or any amount with no coins
How many ways are there to change a dollar?

The number of ways to change an amount $a$ using $n$ kinds of coins is:
1. The number of ways to change $a-d$ using all kinds, where $d$ is the amount of the first kind of coin
2. The number of ways to change $a$ using all but the first kind

```python
def count_change(a, d):
    if a == 0:
        return 1
    if a < 0 or d == 0:
        return 0
    return (count_change(a-d, d) +
            count_change(a, next_coin(d)))
```

One way to make no amount
Can’t make negative amount, or any amount with no coins
Functional abstraction to get next coin