Functional Abstraction
Announcements
Zero-Argument Functions

(Demo)
Dice Functions

In the Hog project, there are multiple zero-argument functions that represent dice. A dice function returns an integer that is the outcome of rolling once. (Demo)

Implement repeat, which returns the # of times in n rolls that repeat the last roll.

```
5 3 3 4 2 1 6 5 3 4 2 2 4 4 3 4 3 5 5               repeat(20, six_sided) -> 5
```

def repeats(n, dice):
    count = 0
    previous = 0
    while n:
        outcome = dice()
        if previous == outcome:
            count += 1
            previous = outcome
        n -= 1
    return count

f1: repeats [parent=Global]

<table>
<thead>
<tr>
<th>outcome</th>
<th>n</th>
<th>count</th>
<th>previous</th>
<th>dice</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20</td>
<td>0</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return value
Higher-Order Loops

(Demo)
Conditional Expressions Practice
(3 and 4) – 5
True and False Values

The built-in bool(x) returns True for true x and False for false x.

```python
>>> bool(0)
False
>>> bool(-1)
True
>>> bool(0.0)
False
>>> bool('')
True
>>> bool('')
False
>>> bool(False)
False
>>> bool(print('fool'))
'fool'
False
```
Lambda Expressions Practice
Lambda and Def

Any program containing lambda expressions can be rewritten using def statements.

```python
>>> (lambda f: lambda x: f(f(x)))(lambda y: y * y)(3)
81

>>> def twice(f):
...     def g(x):
...         return f(f(x))
...     return g
...

>>> def square(y):
...     return y * y
...

>>> twice(square)(3)
81
```
(2.0 pt) Choose all correct implementations of `funsquare`, a function that takes a one-argument function `f`. It returns a one-argument function `f2` such that `f2(x)` has the same behavior as `f(f(x))` for all `x`.

```python
>>> triple = lambda x: 3 * x
>>> funsquare(triple)(5)  # Equivalent to triple(triple(5))
45
```

A: ```python
def funsquare(f):
    return f(f)
```

B: ```python
def funsquare(f):
    return lambda x: f(f(x))
```

C: ```python
def funsquare(f, x):
def g(x):
    return f(f(x))
return g
```

D: ```python
def funsquare(f):
    return lambda x: f(f(x))
```

E: ```python
def funsquare(f, x):
    return f(f(x))
```

F: ```python
def funsquare(f):
def g(x):
    return f(f(x))
return g
```
```python
>>> snap = lambda chat: lambda: snap(chat)
>>> snap, chat = print, snap(2020)
What is displayed here?
>>> chat()
What is displayed here?
```
Call Expressions
Assigning Names to Values

There are three ways of assigning a name to a value:

• Assignment statements (e.g., \texttt{y = x}) assign names in the current frame

• Def statements assign names in the current frame

• Call expressions assign names in a new local frame

\[
h = \lambda f: \lambda x: f(f(x)) \quad f = \texttt{abs} \quad h = \lambda f: f(f(x)) \\
h(\texttt{abs})(-3) \quad x = -3 \quad x = -3 \\
f(f(x)) \quad h(\texttt{abs})
\]