Functional Abstraction

Announcements

Zero-Argument Functions

(Demo)

Dice Functions

The six_sided function returns an integer 1-6 that is the outcome of rolling once. (Demo)

Implement repeat, which returns the # of times in n rolls that an outcome repeats.

5 3 3 4 2 1 6 5 3 4 2 2 2 4 4 3 4 3 5 5   repeats(20) -> 5

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

f1: repeats [parent=Global]

n	20
count	0
previous	05
outcome	53

Return value

Lambda Expressions Practice

Lambda and Def

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

```
                 twice                      square
>>> (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
```

CS 61A Fall 2022 Midterm 1 Question 4(a)

(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`.

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

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

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

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

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

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

```
F:  def funsquare(f):
        def g(x):
            return f(f(x))
        return g
```

CS 61A Fall 2022 Midterm 1 Question 4(a)

(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`.

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

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

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

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

```
[✓ D:]  def funsquare(f):
            return lambda x: f(f(x))
```

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

```
[✓ F:]  def funsquare(f):
            def g(x):
                return f(f(x))
            return g
```

CS 61A Spring 2020 Midterm 1 Question 1

```
>>> snap = lambda chat: lambda: snap(chat)
>>> snap, chat = print, snap(2020)
What is displayed here?
>>> chat()
What is displayed here?
```

https://pythontutor.com/cp/composingprograms.html#code=snap%20%3D%20lambda%20chat%3A%20lambda%3A%20snap%28chat%29%0Asnap,%20chat%20%3D%20print,%20snap%282020%29%0Achat%28%29%0A%0A&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Higher-Order Loops

(Demo)

https://pythontutor.com/cp/composingprograms.html#code=def%20reprint%28n%29%3A%0A%20%20%20%20def%20a%28word%29%3A%0A%20%20%20%20%20%20%20%20k%20%3D%20n%0A%20%20%20%20%20%20%20%20while%20k%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20print%28word%29%0A%20%20%20%20%20%20%20%20%20%20%20%20k%20-
%3D%201%0A%20%20%20%20return%20a%0A%20%20%20%20%0Areprint%282%29%28'hey'%29%0A&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Conditional Expressions (and/or)

True and False Values

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

```
>>> bool(0)
False
>>> bool(-1)
True
>>> bool(0.0)
False
>>> bool(' ')
True
>>> bool('')
False
>>> bool(False)
False
>>> bool(print('fool'))
fool
False
```

Call Expressions

Assigning Names to Values

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

- Assignment statements (e.g., 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))
h(abs)(-3)
```

```
f = abs
x = -3
f(f(x))
```

```
h = lambda f: f(f(x))
x = -3
h(abs)
```