

# 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 an outcome repeats.

5 3 3 4 2 1 6 5 3 4 2 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]

n	20
dice	
count	0
previous	0
outcome	5

Return value

func ...

## Higher-Order Loops

( Demo )

<https://pythontutor.com/cp/composingprograms.html#code=def%20reprint%28n%2A9%A0%20%20%20%20%20word%29%3A%A0%20%20%20%20%20%20k%20%20%20%20%20%20%20%20%20while%20k%3A%A0%20%20%20%20%20%20%20%20%20print%20word%29%A0%20%20%20%20%20%20%20%20return%20%A0%20%20%20%A0reprint%282%29%20'hey'%29%A0%20cumulative=true&curInstr=8&displayOrigin=composingprograms.js&p=3&rawInputListJSO=%5B%5D>

## Conditional Expressions Practice

## Fall 2022 Midterm 1 Question 1

---

(3 and 4) – 5

## 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
```



## 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
```

## 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)`

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

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

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

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

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

## 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?*

## 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)
```

## Environment Diagram Practice

## Fall 2022 CS 61A Midterm 1, Question 2

- The Diagram
- Annotations

```

1: def f(x):
2:     """f(x)(t) returns max(x*x, 3*x)
3:     if t(x) > 0, and 0 otherwise.
4:     """
5:     y = max(x * x, 3 * x)
6:     def zero(t):
7:         if t(x) > 0:
8:             return y
9:         return 0
10:    return zero
11:
12: # Find the largest positive y below 10
13: # for which f(y)(lambda z: z - y + 10)
14: # is not 0.
15: y = 1
16: while y < 10:
17:     if f(y)(lambda z: z - y + 10):
18:         max = y
19:     y = y + 1

```

The diagram illustrates the environment structure during the execution of the function `func f(x) [p=G]`. It consists of several frames and function objects:

- Global frame:** Contains bindings for `f` (pointing to `func f(x) [p=G]`), `y` (pointing to `1`), and `max` (pointing to `1`). The `func f(x) [p=G]` object has three red triangles below it, indicating it is a function object.
- f1 frame:** Created by the global frame. It contains bindings for `x` (pointing to `1`), `y` (pointing to `3`), and `zero` (pointing to `func zero(t) [p=f1]`). The `Return Value` field is empty.
- f2 frame:** Created by the `f1` frame. It contains bindings for `t` (pointing to `3`) and `Return Value` (pointing to `3`).
- f3 frame:** Created by the `f2` frame. It contains bindings for `z` (pointing to `1`) and `Return Value` (pointing to `10`).
- f4 frame:** Created by the `f3` frame. It contains bindings for `x` (pointing to `2`) and `Return Value` (empty).

Arrows indicate the parent environment for each frame: `f1` points to the global frame, `f2` points to `f1`, `f3` points to `f2`, and `f4` points to `f3`.