

Lecture #8: More on Functions

Last modified: Mon Feb 22 16:36:28 2016

CS61A: Lecture #8 1

Another Recursion Problem: Counting Partitions

- I'd like to know the number of distinct ways of expressing a non-negative integer as a sum of positive integer "parts."
- To make things more interesting, let's also limit the size of the integer parts to some given value:

```
def num_partitions(n, k):  
    """Number of distinct ways to express N>=0 as a sum of  
    positive integers each of which is <= K, where K > 0."""
```

- Example:

```
6 = 3 + 3  
  = 3 + 2 + 1  
  = 3 + 1 + 1 + 1  
  = 2 + 2 + 2  
  = 2 + 2 + 1 + 1  
  = 2 + 1 + 1 + 1 + 1  
  = 1 + 1 + 1 + 1 + 1 + 1
```

so `num_partitions(6, 3)` is 7.

Last modified: Mon Feb 22 16:36:28 2016

CS61A: Lecture #8 2

Counting Partitions: Solution

```
def num_partitions(n, k):  
    """Number of distinct ways to express N>=0 as a sum of  
    positive integers each of which is <= K, where K > 0."""  
    if _____:  
        return _____  
    else:
```

Last modified: Mon Feb 22 16:36:28 2016

CS61A: Lecture #8 3

Decorators: Pythonic Use of Higher-Order Functions

- The syntax

```
@expr  
def func(expr):  
    body
```

is equivalent to

```
def func(expr):  
    body  
func = (expr)(func)
```

- For example, our `ucb` module defines decorator `trace`. After

```
from ucb import trace  
@trace  
def mysum(x, y):  
    return x + y
```

`mysum` will print its arguments and return value each time it is called.

- Usually, `expr` is a simple name, but it can be any expression that evaluates to a function that takes and returns a function.

Last modified: Mon Feb 22 16:36:28 2016

CS61A: Lecture #8 4

Implement trace

```
def trace(func):  
    """A decorator that accepts the same arguments  
    and returns the same value as FUNC, but also  
    prints the arguments and return value."""  
    def afunc(*args):  
        print(args)  
        v = func(*args)  
        print(v)  
        return v  
  
    return afunc
```

(The actual `trace` function is fancier, but this gives the gist.)

Last modified: Mon Feb 22 16:36:28 2016

CS61A: Lecture #8 5

Design a Decorator

- I'd like a decorator that will check that the output of a function obeys some predicate:

```
@check_result(lambda x: x < 1000)  
def compute(x):  
    ...  
    return whatever # value of whatever must be < 1000.
```

- How would you define `check_result`?

- It must return a function that

- Takes a function, say `func`, as input
- Returns a function that takes the same arguments as `func` and returns the same value as `func` if that value satisfies `PRED`, but complains otherwise.

Last modified: Mon Feb 22 16:36:28 2016

CS61A: Lecture #8 6