

Higher-Order Functions

Announcements

Office Hours: You Should Go!

You are not alone!

<https://cs61a.org/office-hours/>

Pensieve

Designing Functions

Describing Functions

A function's *domain* is the set of all inputs it might possibly take as arguments.

A function's *range* is the set of output values it might possibly return.

A pure function's *behavior* is the relationship it creates between input and output.

```
def square(x):  
    """Return X * X."""
```

x is a number

square returns a non-negative real number

square returns the square of x

A Guide to Designing Function

Give each function exactly one job, but make it apply to many related situations

```
>>> round(1.23)      >>> round(1.23, 1)      >>> round(1.23, 0)      >>> round(1.23, 5)
1                    1.2                    1                    1.23
```

Don't repeat yourself (DRY): Implement a process just once, but execute it many times

(Demo)

Higher-Order Functions

Summation Example

```
def cube(k):  
    return pow(k, 3)
```

Function of a single argument
(*not* called "term")

```
def summation(n, term):  
    """Sum the first n terms of a sequence.
```

A formal parameter that will
be bound to a function

```
>>> summation(5, cube)
```

```
225
```

```
"""
```

```
    total, k = 0, 1
```

```
    while k <= n:
```

```
        total, k = total + term(k), k + 1
```

```
    return total
```

The cube function is passed
as an argument value

0 + 1 + 8 + 27 + 64 + 125

The function bound to term
gets called here

Modularity

Abstraction

Separation of Concerns

Twenty-One Rules

Two players alternate turns, on which they can add 1, 2, or 3 to the current total

The total starts at 0

The game end whenever the total is 21 or more

The last player to add to the total loses

(Demo)

Functions as Return Values

(Demo)

Locally Defined Functions

Functions defined within other function bodies are bound to names in a local frame

A function that
returns a function

```
def make_adder(n):  
    """Return a function that takes one argument k and returns k + n.  
  
    >>> add_three = make_adder(3)  
    >>> add_three(4)  
    7  
    """  
    def adder(k):  
        return k + n  
    return adder
```

The name add_three is bound
to a function

A def statement within
another def statement

Can refer to names in the
enclosing function

Twenty-One Strategies

(Demo)