Higher-Order Functions

Announcements

Office Hours: You Should Go!

You are not alone!

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

Iteration Example: Nice Numbers

Nice Numbers

Rounding off 2,799 to 2,800 makes it nice.

Definition: A nice number doesn't have 98, 99, 01, or 02 among its digits and 00 can only be followed by more 0's.

Not-so-nice numbers:	**99**	**2,799**	**5,016**	**9,902**	**1,200,456**	**98,402,001**
Nicer versions:	**100**	**2,800**	**5,000**	**10,000**	**1,200,000**	**100,000,000**

These numbers are nice enough already and unaffected: **755** **2,859** **45,622,895**

Implement **nice**, which takes a positive integer n. It returns the nearest nice number to n.

- For numbers that end in 98 or 99 or 01 or 02, round to the nearest 100.
- Look for 98 or 99 or 01 or 02 among the digits that aren't at the end.

To solve a problem, describe a process and work through an example:

```
4 7 9 8 4 0 2 0 0 1
4 7 9 8 4 0 2 0 0 0
4 7 9 8 4 0 0 0 0 0
4 8 0 0 0 0 0 0 0 0
```

(Demo)

https://pythontutor.com/cp/
composingprograms.html#code=def%20nice_end%28n%29%3A%0A%20%20%20%20%22Round%20n%20ending%20in%2098,%2099,%2001,%20or%2002%20to%20the%20nearest%20100.%22%0A%20%20%20%20if%20n%20%25%20100%20%3E%3D%2098%3A%0A%20%20%20%20%20%20%20%20return%20n%20//%20100%20*%20100%20%2B%20100%0A%20%20%20%20elif%20n%20%25%20100%20%3C%3D%202%3A%0A%20%20%20%20%20%20%20%20return%20n%20//%20100%20*%20100%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20return%20n%0A%0Adef%20nice%28n%29%3A%0A%20%20%20%20%22Find%20the%20closest%20nice%20number%20to%20n.
%22%0A%20%20%20%20rest%20%3D%20n%0A%20%20%20%20k%20%3D%200%0A%20%20%20%20while%20rest%20%3E%2010%3A%0A%20%20%20%20%20%20%20%20if%20nice_end%28rest%29%20%25%20100%20%3D%3D%200%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20n%20%3D%20nice_end%28rest%29%20*%2010%20**%20k%0A%20%20%20%20%20%20%20%20rest,
%20k%20%3D%20rest%20//%2010,%20k%20%2B%201%0A%20%20%20%20return%20n%0A%0Anice%284798402001%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Designing Functions

Describing Functions

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

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

x is a number

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

square returns a non-negative real number

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

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)
1
```

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

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

```
>>> round(1.23, 5)
1.23
```

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

(Demo)

Higher-Order Functions

Summation Example

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

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

    >>> summation(5, cube)
    225
    """
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total
```

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

A formal parameter that will be bound to a function

The cube function is passed as an argument value

0 + 1 + 8 + 27 + 64 + 125

The function bound to term gets called here

Program Design

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

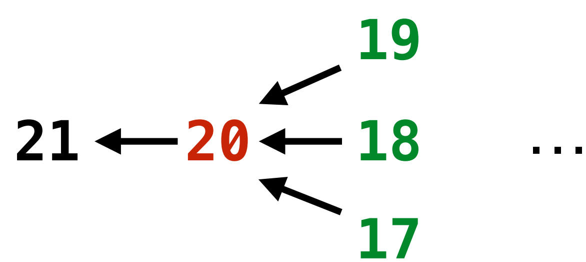
The total starts at 0

The game end whenever the total is 21 or more

The last player to add to the total loses

(Demo)

Some states are good; some are bad



(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