Function Examples

Announcements

Call Expressions

Lab 02 Q2: Higher-Order Functions

```
>>> def cake():
...     print('beets')
...     def pie():
...         print('sweets')
...         return 'cake'
...     return pie
...
>>> chocolate = cake()
beets
>>> chocolate
<function cake.<locals>.pie at ...>
>>> chocolate()
sweets
'cake'
```

```
>>> def snake(x, y):
...     if cake == more_cake:
...         return chocolate
...     else:
...         return x + y
...
>>> snake(10, 20)
<function cake.<locals>.pie at ...>
>>> snake(10, 20)()
sweets
'cake'
>>> cake = 'cake'
>>> snake(10, 20)
30
```

```
>>> more_chocolate, more_cake = chocolate(), cake
sweets
>>> more_chocolate
'cake'
```

https://pythontutor.com/cp/composingprograms.html#code=def%20cake%28%29%3A%0A%20%20print%28'beets'%29%0A%20%20def%20pie%28%29%3A%0A%20%20%20%20print%28'sweets'%29%0A%20%20%20%20return%20'cake'%0A%20%20return%20pie%0A%0Achocolate%20%3D%20cake%28%29%0A%23%20chocolate%0A%23%20chocolate%28%29%0Amore_chocolate,%20more_cake%20%3D%20chocolate%28%29,%20cake%0A%23%20more_chocolate%0Adef%20snake%28x,%20y%29%3A%0A%20%20if%20cake%20%3D%3D%20more_cake%3A%0A%20%20%20%20return%20chocolate%0A%20%20else%3A%0A%20%20%20%20return%20x%20%2B%20y%0A%0A%23%20snake%2810,%2020%29%0Asnake%2810,%2020%29%28%29%0A%23%20cake%20%3D%20'cake'%0A%23%20snake%2810,%2020%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Twenty-One Environment Diagram

Twenty-One Rules

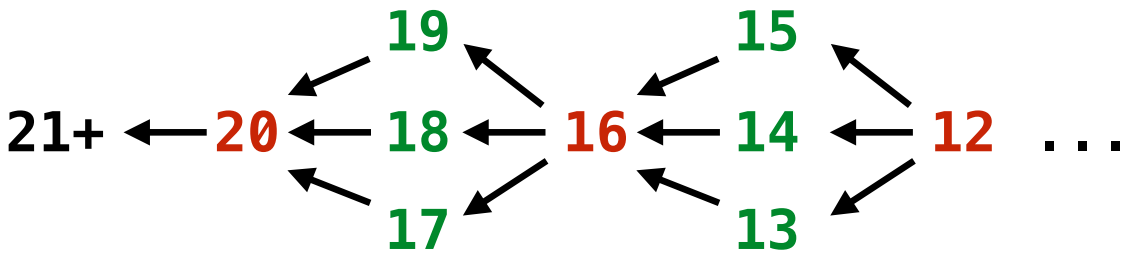
Two players alternate turns, on which they can add 1, 2, or 3 to the current total

The total starts at 0

The game end whenever the total is 21 or more

The last player to add to the total loses

Some states are good; some are bad



(Demo)

https://pythontutor.com/cp/composingprograms.html#code=def%20play%28strategy0,%20strategy1,%20goal%3D21%29%3A%0A%20%20%20n,
%20who%20%3D%200,%200%0A%20%20%20%20while%20n%20%3C%20goal%3A%0A%20%20%20%20%20%20%20if%20who%20%3D%3D%200%3A%0A%20%20%20%20%20%20%20%20%20%20n%20%3D%20n%20%2B%20strategy0%28n%29%0A%20%20%20%20%20%20%20%20%20%20who%20%3D%201%0A%20%20%20%20%20%20%20elif%20who%20%3D%3D%201%3A%0A%20%20%20%20%20%20%20%20%20%20n%20%3D%20n%20%2B%20strategy1%28n%29%0A%20%20%20%20%20%20%20%20%20%20who%20%3D%200%0A%20%20%20%20return%20who%20%0A%0Adef%20two_strat%28n%29%3A%0A%20%20%20%20return%202%0A%0Adef%20noisy_strat%28who,%20s%29%3A%0A%20%20%20%20def%20strat%28n%29%3A%0A%20%20%20%20%20%20%20%20choice%20%3D%20s%28n%29%0A%20%20%20%20%20%20%20%20print%28'Player',%20who,%20'added',%20choice,%20'to',%20n,%20'to%20reach',
%20choice%20%2B%20n%29%0A%20%20%20%20%20%20%20%20return%20choice%0A%20%20%20%20return%20strat%0A%0Aplay%28noisy_strat%280,%20two_strat%29,%20noisy_strat%281,%20two_strat%29%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Function Implementation Practice

A Slight Variant of Fall 2022 Midterm 1 3(b)

Implement nearest_prime, which takes an integer n above 5. It returns the nearest prime number to n. If two prime numbers are equally close to n, return the larger one. Assume is_prime(n) is implemented already.

```
def nearest_prime(n):
    """Return the nearest prime number to n.
    In a tie, return the larger one.

    >>> nearest_prime(8)
    7
    >>> nearest_prime(11)
    11
    >>> nearest_prime(21)
    23
    """
    ____
    while True:
        if ___:
            ____
        if ___:
            ____:
            k = -k
        else:
            ____
```

From discussion:

Describe a process (in English) that computes the output from the input using simple steps.

Figure out what additional names you'll need to carry out this process.

Implement the process in code using those additional names.

From the videos:

Read the description

Verify the examples & pick a simple one

Read the template

Annotate names with values from your chosen example

Write code to compute the result

Did you really return the right thing?

Check your solution with the other examples

Currying

Function Currying

```
def make_adder(n):
    return lambda k: n + k
```

```
>>> make_adder(2)(3)
5
>>> add(2, 3)
5
```

There's a general relationship between these functions

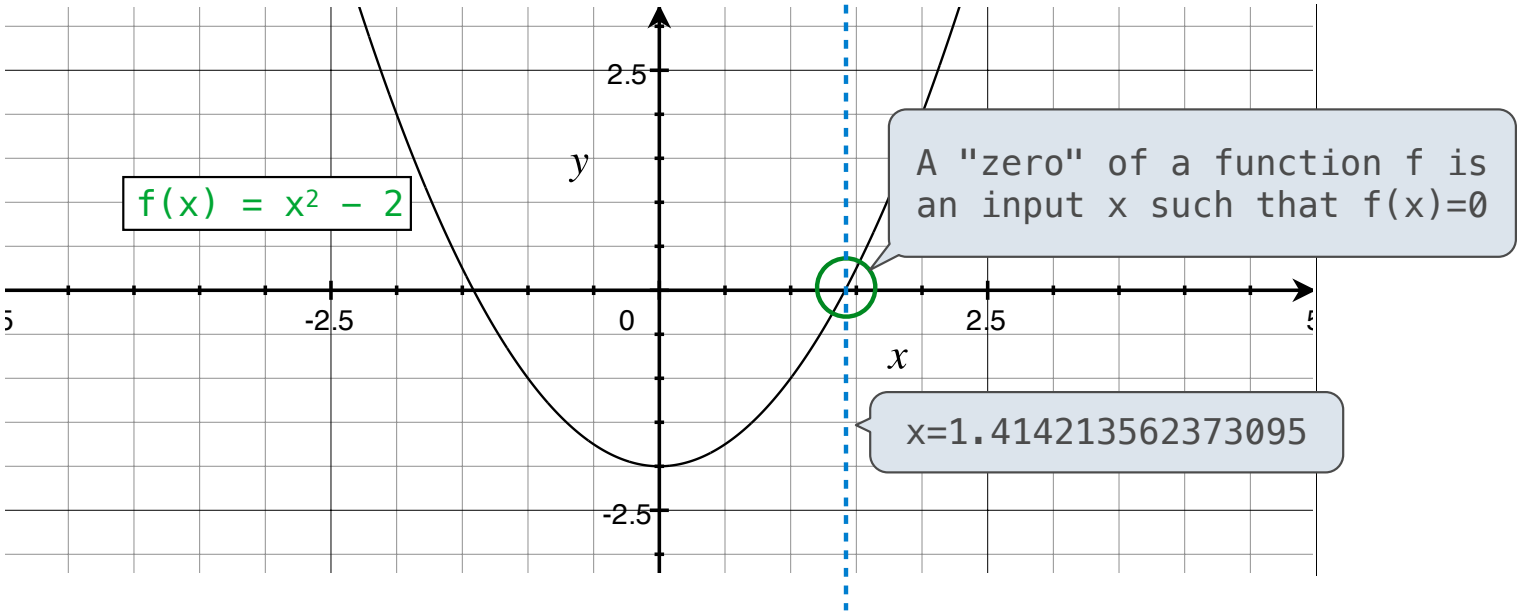
(Demo)

Curry: Transform a multi-argument function into a single-argument, higher-order function

Example: Newton's Method (OPTIONAL)

Newton's Method Background

Quickly finds accurate approximations to zeroes of differentiable (smooth) functions



Application: a method for computing square roots, cube roots, etc.

The positive zero of $f(x) = x^2 - a$ is $\sqrt{a}$. (We're solving the equation $x^2 = a$.)

Newton's Method

Given a function f and initial guess x,

Repeatedly improve x:

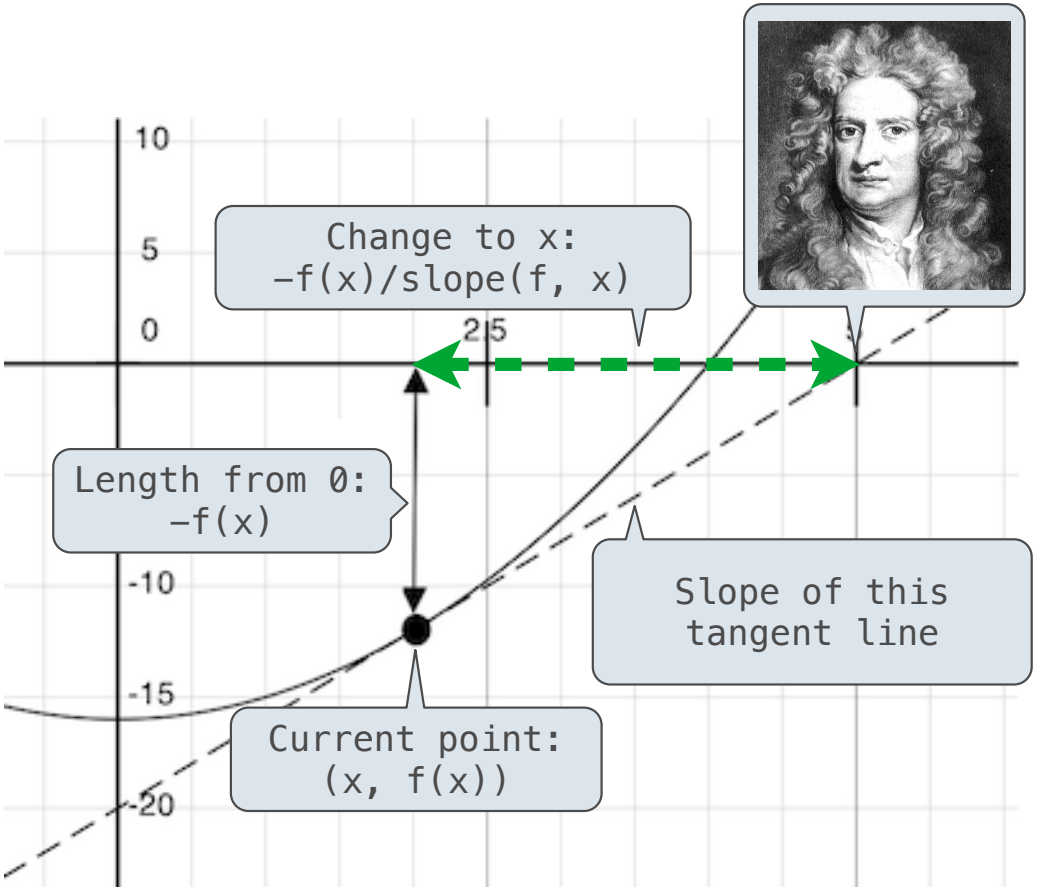
Compute the value of f at the guess: f(x)

Compute the slope of f at the guess: slope(f, x)

Update guess x to be:

$$x - \frac{f(x)}{\text{slope}(f, x)}$$

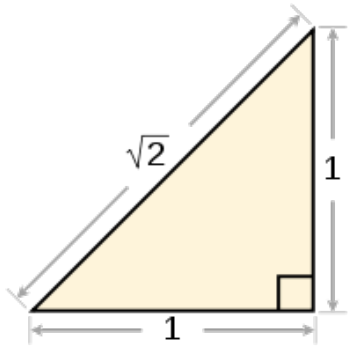
Finish when f(x) = 0 (or close enough)



http://en.wikipedia.org/wiki/File:NewtonIteration_Ani.gif

Using Newton's Method

How to find the square root of 2?




```
>>> f  = lambda x: x*x - 2
>>> find_zero(f)
1.4142135623730951
```

Applies Newton's method