Higher-Order Functions

Announcements

Example: Prime Factorization

Prime Factorization

Each positive integer n has a set of prime factors: primes whose product is n

```
...
8  = 2 * 2 * 2
9  = 3 * 3
10 = 2 * 5
11 = 11
12 = 2 * 2 * 3
...
```

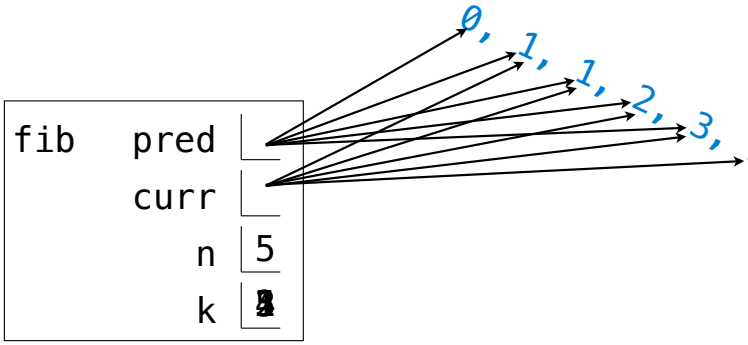
One approach: Find the smallest prime factor of n, then divide by it

```
858  = 2 * 429  = 2 * 3 * 143  = 2 * 3 * 11 * 13
```

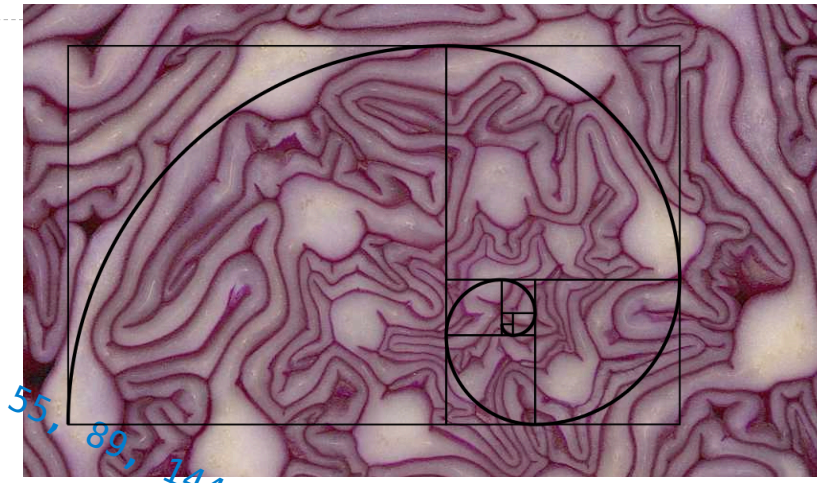
(Demo)

Example: Iteration

The Fibonacci Sequence



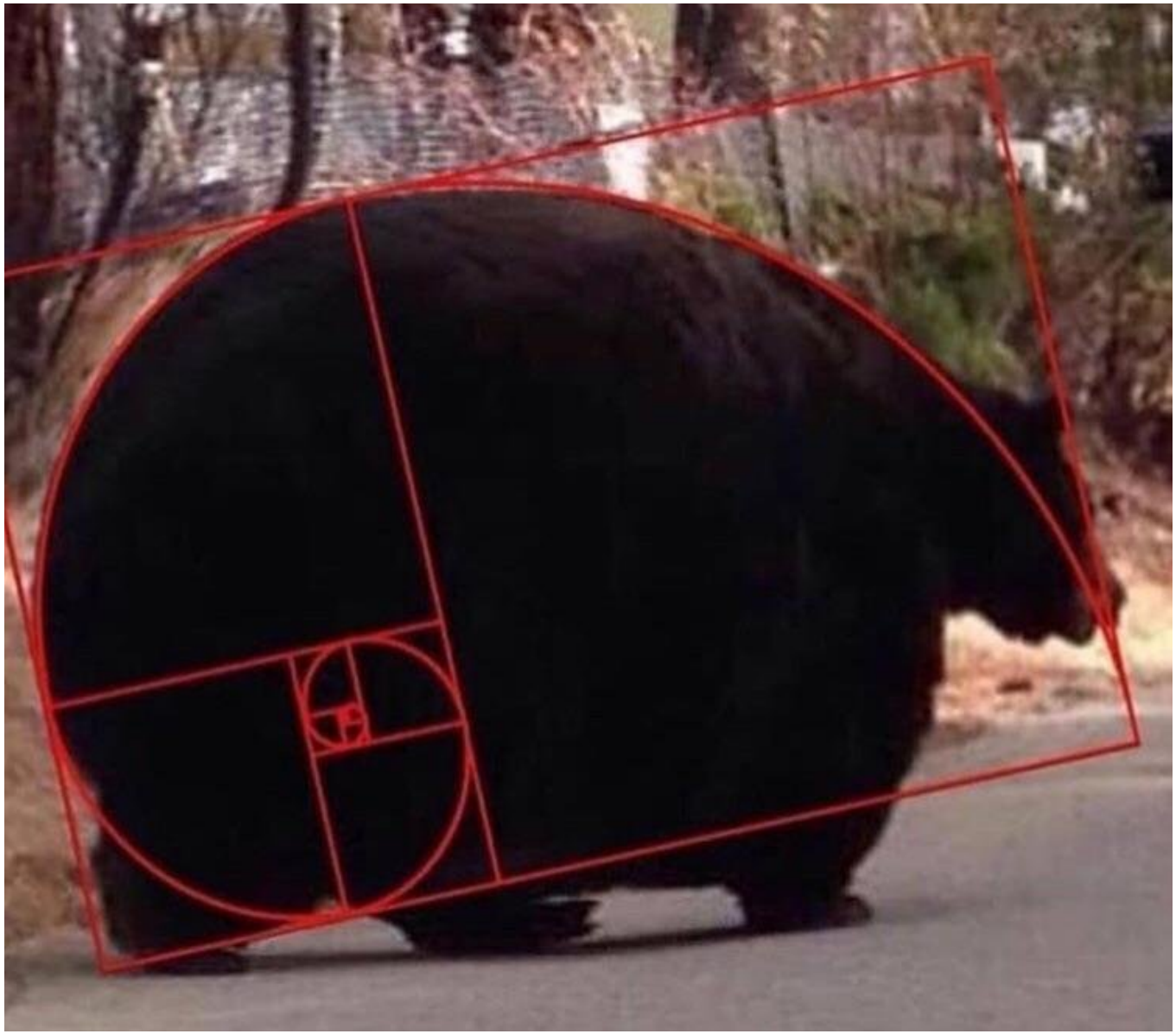
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987



```
def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""
    pred, curr = 0, 1   # 0th and 1st Fibonacci numbers
    k = 1               # curr is the kth Fibonacci number
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

The next Fibonacci number is the sum of the current one and its predecessor

Go Bears!



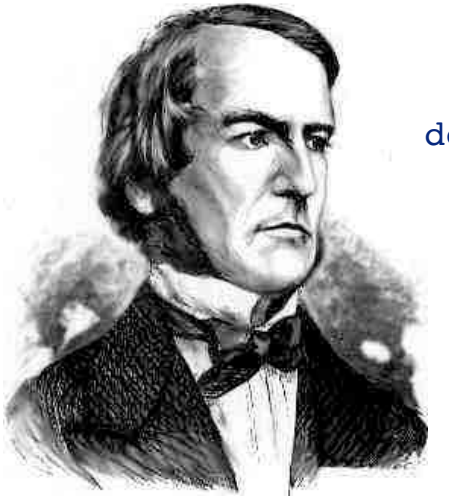
Control

Boolean Contexts

George Boole

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Boolean Contexts



George Boole

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Two boolean contexts

False values in Python: False, 0, '', None *(more to come)*

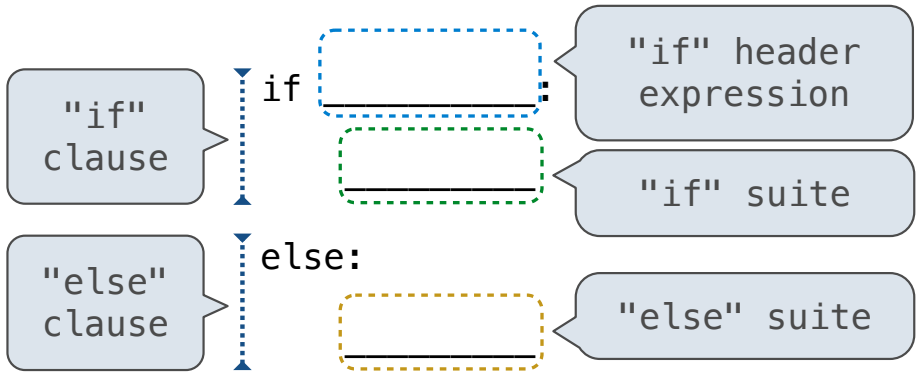
True values in Python: Anything else (True)

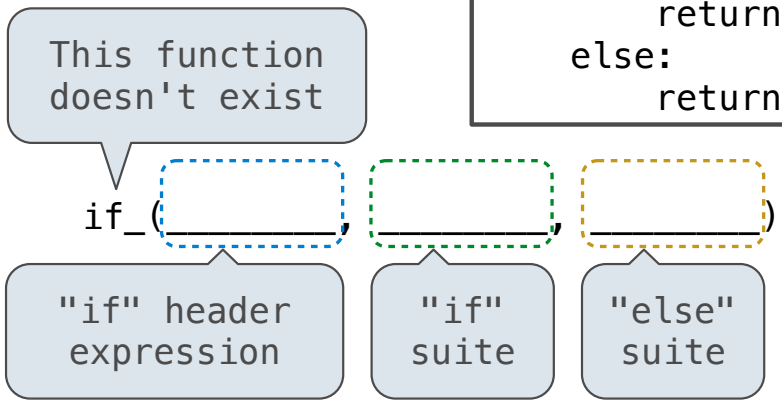
(Demo)

If Statements and Call Expressions

Let's try to write a function that does the same thing as an if statement.

```
def if_(c, t, f):
    if c:
        return t
    else:
        return f
```





Execution Rule for Conditional Statements:

Each clause is considered in order.

1. Evaluate the header's expression (if present).
2. If it is a true value (or an else header), execute the suite & skip the remaining clauses.

(Demo)

Evaluation Rule for Call Expressions:

1. Evaluate the operator and then the operand subexpressions
2. Apply the function that is the value of the operator to the arguments that are the values of the operands

Short-Circuiting Expressions

Logical Operators

To evaluate the expression **<left>** **and** **<right>**:

1. Evaluate the subexpression **<left>**.
2. If the result is a false value **v**, then the expression evaluates to **v**.
3. Otherwise, the expression evaluates to the value of the subexpression **<right>**.

To evaluate the expression **<left>** **or** **<right>**:

1. Evaluate the subexpression **<left>**.
2. If the result is a true value **v**, then the expression evaluates to **v**.
3. Otherwise, the expression evaluates to the value of the subexpression **<right>**.

(Demo)

Higher-Order Functions

Generalizing Over Computational Processes

The common structure among functions may be a computational process, rather than a number.

$$\sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 = 15$$

$$\sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225$$

$$\sum_{k=1}^{5} \frac{8}{(4k-3)\cdot(4k-1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04$$

(Demo)

Summation Example

```
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