Today

• Quick orders of growth review
• Data Abstraction and making new Abstract Data Types
• Tuples

Review: Orders of Growth

What is the order of growth (using \( \Theta \) notation) for the following Python function?

```python
def foo(x):
    if x < 3:
        return x
    return foo(x % 3) + foo(x - 1)
```

\( \Theta(n) \)

We know that the result of \( n \mod 3 \) is 0, 1, or 2 (the base case), so we know that the first recursive call will always result in a base case and we can treat it as a constant time operation.

The second recursive call will take (about) \( n \) recursive calls before reaching a base case (we subtract one from \( n \) each time). So we have \( \Theta(n) \) recursive calls with constant amount of work done for each call.

Data Abstraction

We want to be able to think about data in terms of its meaning rather than in terms of the way it is represented.

Data abstraction allows us to isolate:

– How the data is represented (as parts)
– How the data is manipulated (as units)

We do this by using functions to help create a division between these two cases.
**PROBLEM: RATIONAL NUMBERS**

*Exact representation of fractions using a pair of integers.*

<table>
<thead>
<tr>
<th>Operation</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplication</td>
<td>$\frac{a}{c} \times \frac{d}{e} = \frac{a \times d}{c \times e}$</td>
</tr>
<tr>
<td>Addition</td>
<td>$\frac{a}{c} + \frac{b}{d} = \frac{a \times d + b \times c}{c \times d}$</td>
</tr>
<tr>
<td>Equality</td>
<td>$\frac{a}{c} = \frac{b}{d} \iff ad = bc$</td>
</tr>
</tbody>
</table>

---

**PROBLEM: RATIONAL NUMBERS**

We’d like to be able to create and decompose rational numbers in our program:

- `make_rat(n, d)` – returns the rational number $\frac{n}{d}$
- `numer(x)` – returns the numerator of x
- `denom(x)` – returns the denominator of x

---

**PRACTICE: USING ABSTRACTIONS**

How would I write a function to invert (flip) a rational number using the constructor and selectors we are using for rational numbers?

```python
def invert_rat(r):
    return make_rat(denom(r), numer(r))
```

---

```
Notice that we don’t have to know how rational numbers work in order to write any code that uses them!
```
TUPLES: OUR FIRST DATA STRUCTURE

Tuples are a built-in datatype in Python for representing a constant sequence of data.

```python
>>> pair = (1, 2)
>>> pair[0]
1
>>> x, y = pair
>>> x
1
>>> y
2
>>> z = pair + (6, 5, 4)
>>> z
(1, 2, 6, 5, 4)
>>> len(z)
5
>>> z[2:5]
(6, 5, 4)
>>> triplet = (1, 2, 3)
>>> for num in triplet:
...     print(num, "potato")
1 potato
2 potato
3 potato
>>> (1)
1
>>> (1,)
(1,)
```

The Python data type **tuple** is an example of what we call a *data structure* in computer science.

A data structure is a type of data that exists primarily to hold other pieces of data in a specific way.

PRACTICE: USING TUPLES AND ABSTRACTIONS

Write the higher order function `map`, which takes a function, `fn`, and a tuple of values, `vals`, and returns the tuple of results of applying `fn` to each value in `vals`.

```python
>>> map(square, (1, 2, 3, 4, 5))
(1, 4, 9, 16, 25)
```

PROBLEM: RATIONAL NUMBERS

```python
def make_rat(n, d):
    return (n, d)
def numer(x):
    return x[0]
def denom(x):
    return x[1]
```
ABSTRACTION DIAGRAMS

rational numbers as numerators and denominators

Using the ADT

make_rat, numer, denom

Implementing the ADT

rational numbers as tuples

tuples as sequences of data

Using the ADT

tuple, getitem

However Python implements tuples

DATA ABSTRACTION: SO WHAT?

It makes code more readable and intuitive.

Which version is clearer?

```
def mul_rats(r1, r2):
    return make_rat(numer(r1) * numer(r2), denom(r1) * denom(r2))
```

```
def mul_rats(r1, r2):
    return (r1[0] * r2[0], r1[1] * r2[1])
```

When we write code that assumes a specific implementation of our ADT, we call this a data abstraction violation (DAV).

DATA ABSTRACTION: SO WHAT?

We don’t have to worry about changing all the code that uses our ADT if we decide to change the implementation!

```
def make_rat(n, d):
    return (d, n)

def numer(x):
    return x[1]

def denom(x):
    return x[0]
```

```
def mul_rats(r1, r2):
    return make_rat(numer(r1) * numer(r2), denom(r1) * denom(r2))
```

# Will break
```
def mul_rats(r1, r2):
    return (r1[0] * r2[0], r1[1] * r2[1])
```

# Will still work
```
def prod_rats(rats):
    total, i = make_rat(1, 1), 0
    while i < len(rats):
        total = make_rat(numer(total) * numer(rats[i]),
                         denom(total) * denom(rats[i]))
        i += 1
    return total
```

PRACTICE: DATA ABSTRACTION

Suppose that Louis Reasoner wrote the following function `prod_rats` that takes a tuple of rational numbers using our ADT and returns their product. Correct his code so that he does not have any data abstraction violations.

```
def prod_rats(rats):
    total, i = make_rat(1, 1), 0
    while i < len(rats):
        total = make_rat(numer(total) * numer(rats[i]),
                         denom(total) * denom(rats[i]))
        i += 1
    return total
```

PRACTICE: DATA ABSTRACTION

Suppose that Louis Reasoner wrote the following function `prod_rats` which takes a tuple of rational numbers using our ADT and returns their product. Correct his code so that he does not have any data abstraction violations.

```
def prod_rats(rats):
    total, i = make_rat(1, 1), 0
    while i < len(rats):
        total = make_rat(numer(total) * numer(rats[i]),
                         denom(total) * denom(rats[i]))
        i += 1
    return total
```

Say I wrote the following functions to define my student ADT.

```
def make_student(name, id):
    return (name, id)

def student_name(s):
    return s[0]

def student_id(s):
    return s[1]
```

If I changed the student ADT to also include the student’s age, what functions would I have to add or change in order to complete the abstraction?
**PRACTICE: DATA ABSTRACTION**

Say I wrote the following functions to define my student ADT.

```python
def make_student(name, id):
    return (name, id)
def student_name(s):
    return s[0]
def student_id(s):
    return s[1]
```

If I changed the student ADT to also include the student's age, what functions would I have to add or change in order to complete the abstraction?

You would have to change `make_student` to take this new parameter. If you just represent a student as the tuple `(name, id, age)`, then you only have to add a selector for the student's age. The other two selectors would not have to be modified in this case.

---

**CONCLUSION**

- Tuples are a nice way to group data in Python.
- Learned how to design new types of data by using data abstraction.
- Preview: Useful data structures.

---

**EXTRAS: USING FUNCTIONS TO CREATE ADTs**

It turns out you don't need to have something like tuples in a language in order to group data together. Say I wanted to make a pair abstraction, which is like a tuple of length 2. I could do this with just functions:

```python
def make_pair(first, second):
    def pair(msg):
        if msg == "first":
            return first
        elif msg == "second":
            return second
        return pair
    return pair

def first(p):
    return p("first")
def second(p):
    return p("second")
```

---

**EXTRAS: MORE ABOUT STRINGS**

Strings and tuples are both sequences, meaning that they are things that you can iterate over with a `for` loop. Interestingly, they can also be indexed into and sliced like tuples.

```python
>>> for letter in "abc":
...     print(letter)
a
b
c
>>> "asdf"[2]
d
>>> "slaughterhouse"[1:9]
"laughter"
```