Final Examples

Announcements

Joins Practice

Discussion Question

What's the maximum difference between leg count for two animals with the same weight?

Approach #1: Consider all pairs of animals.

```
SELECT MAX(a.legs - b.legs) AS difference
  FROM animals AS a, animals AS b
  WHERE a.weight = b.weight;
```

Approach #2: Group by weight.

```
SELECT MAX(legs) - MIN(legs) AS difference
  FROM animals
  GROUP BY weight
  ORDER BY difference DESC
  LIMIT 1;
```

animals:

kind	legs	weight
dog	4	20
cat	4	10
ferret	4	10
parrot	2	6
penguin	2	10
t-rex	2	12000

difference
2

Discussion Question

animals:

kind	legs	weight
dog	4	20
cat	4	10
ferret	4	10
parrot	2	6
penguin	2	10
t-rex	2	12000

What are all the kinds of animals that have the maximal number of legs?

```
sqlite> SELECT * FROM animals WHERE legs = MAX(legs);
Parse error: misuse of aggregate function MAX()
```

Approach #1: Give the maximum number of legs a name.

```
CREATE TABLE m AS SELECT MAX(legs) AS max_legs FROM animals;
SELECT kind FROM animals, m WHERE legs = max_legs;
```

Approach #2: For each kind of animal, compare its legs to the maximum legs by grouping.

```
SELECT a.kind FROM animals AS a, animals AS b GROUP BY a.kind HAVING a.legs = MAX(b.legs);
```

Implementing Functions

Implementing a Function

```
def remove(n, digit):
    """Retu... n all ... digits of non-negative N
    ... are ... IT, for some
    ... ega ... IT less than 10.

    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    """
    kept, digits = 0, 0

    while ____n > 0____:

        n, last = n // 10, n % 10

        if ____last != digit____:

            kept = ____10*kept + last*10**digits____

            digits = digits + 1

    return ____kept____
```

Annotations: n = 231; digit = ~~3~~ 4; "10*" in the kept line is crossed out, and callouts mark the other two parts of that line as 1 and 2; return value ~~21~~ 231

```
   1          1
+ 20       + 30
           + 200
-----      -----
  21        231
```

```
kept= 0, last=1
kept= 1, last=3
kept=31, last=2
kept=51
```

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

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):     Example: n is 21
    """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
    """
    k = 0
    while True:
        if _is_prime(23)_:
            return __23__
        if _______:            keep
            k = -k             looking
        else:                  for a
            k = ________       prime
```

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.

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

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):      Example: n is 21
    """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
    """
    k = 0
    while True:
        if is_prime(n + k): is_prime(23)
            return n + k          23
        if k > 0:
            k = -k
        else:
            k = -k + 1
```

keep looking for a prime

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.

Process:
Check whether a number is prime in this order:
- ***original n***
- ***n + 1***
- ***n − 1***
- ***n + 2***
- ***n − 2***
- ***n + 3***
- ***n − 3***
- ***n + 4***
- ***...***

All of these look like n + k for various k

(Demo)

Designing Functions

How to Design Programs

From Problem Analysis to Data Definitions
Identify the information that must be represented and how it is represented in the chosen programming language. Formulate data definitions and illustrate them with examples.

Signature, Purpose Statement, Header
State what kind of data the desired function consumes and produces. Formulate a concise answer to the question *what* the function computes. Define a stub that lives up to the signature.

Functional Examples
Work through examples that illustrate the function's purpose.

Function Template
Translate the data definitions into an outline of the function.

Function Definition
Fill in the gaps in the function template. Exploit the purpose statement and the examples.

Testing
Articulate the examples as tests and ensure that the function passes all. Doing so discovers mistakes. Tests also supplement examples in that they help others read and understand the definition when the need arises—and it will arise for any serious program.

https://htdp.org/2018-01-06/Book/

Tree Processing

Tree-Structured Data

```
class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        self.branches = list(branches)

    def is_leaf(self):
        return not self.branches
```

A tree can contains other trees:

```
[5, [6, 7], 8, [[9], 10]]
```

```
(+ 5 (- 6 7) 8 (* (- 9) 10))
```

```
(S
  (NP (JJ Short) (NNS cuts))
  (VP (VBP make)
      (NP (JJ long) (NNS delays)))
  (. .))
```

```
<ul>
  <li>Midterm <b>1</b></li>
  <li>Midterm <b>2</b></li>
</ul>
```

Tree processing often involves recursive calls on subtrees

Designing a Function

Implement **smalls**, which takes a Tree instance t containing integer labels. It returns the non-leaf nodes in t whose labels are smaller than any labels of their descendant nodes.

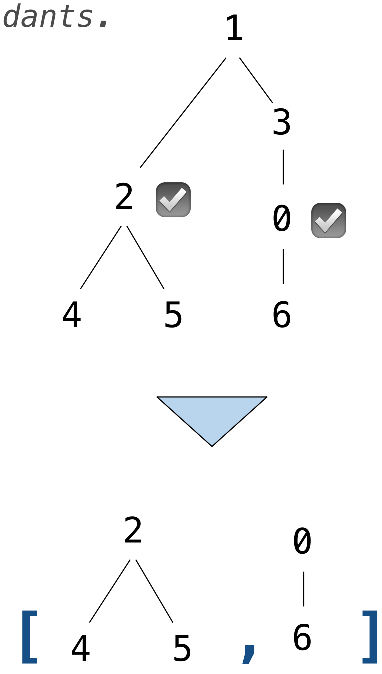
```
def smalls(t):                 Signature: Tree -> List of Trees
    """Return a list of the non-leaf nodes in t that are smaller than all their descendants.

    >>> a = Tree(1, [Tree(2, [Tree(4), Tree(5)]), Tree(3, [Tree(0, [Tree(6)])])])
    >>> sorted([t.label for t in smalls(a)])
    [0, 2]

    """
    result = []                Signature: Tree -> number
    def process(t):            "Find smallest label in t & maybe add t to result"
        if t.is_leaf():
            return t.label
        else:




            return min(...)
    process(t)
    return result
```



Designing a Function

Implement **smalls**, which takes a Tree instance t containing integer labels. It returns the non-leaf nodes in t whose labels are smaller than any labels of their descendant nodes.

Signature: Tree -> List of Trees

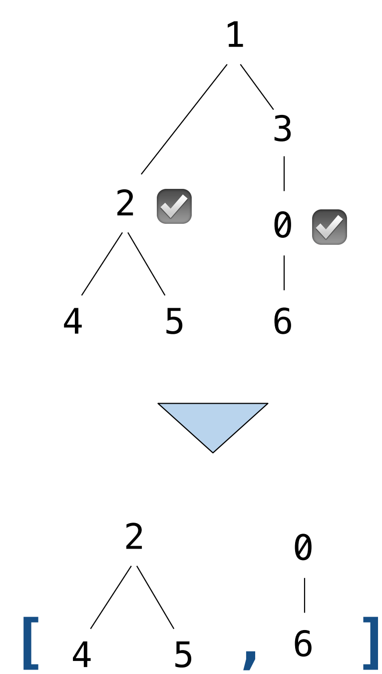
```
def smalls(t):
    """Return a list of the non-leaf nodes in t that are smaller than all descendants.

    >>> a = Tree(1, [Tree(2, [Tree(4), Tree(5)]), Tree(3, [Tree(0, [Tree(6)])])])
    >>> sorted([t.label for t in smalls(a)])
    [0, 2]
    """
    result = []
    def process(t):
        if t.is_leaf():
            return ____t.label____
        else:
            smallest = ____min([process(b) for b in t.branches])____
            if ____t.label < smallest____:
                ____result.append( t )____
            return min(smallest, t.label)
    process(t)
    return result
```

Signature: Tree -> number

"Find smallest label in t & maybe add t to result"

smallest label in a branch of t (pointing to `smallest`)



Fall 2022 Midterm 2 Question 4(b)

A *hydra* is a Tree with a special structure. Each node has 0 or 2 children. All leaves are heads labeled 1. Each non-leaf body node is labeled with the number of leaves among its descendants.

Implement **chop_head(hydra, n),** which takes a hydra and a positive integer n. It mutates the hydra by replacing the nth head from the left with two new adjacent heads & updating all ancestor labels.

```
def chop_head(hydra,  n): Signature: (hydra, int) -> None
    assert n > 0 and n <= hydra.label
    if hydra.is_leaf():
        hydra.label = 2
        hydra.branches = [Tree(1),  Tree(1)]
    else:
        hydra.label += 1
        left, right = hydra.branches
        if n > left.label :
            chop_head(right, n - left.label)
        else:
            chop_head(left, n)
```

Mutate the hydra

Update ancestor

nth head