Lecture #9: More Functions

Another Tree Recursion: Hog Dice

- What are the odds of rolling at least k in hog with n s-sided dice? ($n > 0$ and for us, $s > 0$ is 4 or 6)

$$\frac{\text{\# rolls of } n \ s\text{-sided dice totaling} \geq k}{s^n}$$

- If $k \leq 1$, then clearly the numerator is just s^n.
- For $k > 1$, we consider only rolls that include dice values 2–s, since any 1-die "pigs out." Let's call this quantity rolls2(k, n, s).
- The number of ways to score $\geq k$ is 0 if ______. This is a base case.
- If $n > 0$ then the number of ways to score at least $k \leq 1$ with n dice none of which is 1 is ______. This is also a base case.
- If the first die comes up d ($2 \leq d \leq s$), then there are ____________ ways to throw the remaining $n-1$ dice to get a total of at least k with all n dice.
- This gives us a tree recursion. How would you modify it for the "swine swap" rule?

Another Tree Recursion: Hog Dice

- What are the odds of rolling at least k in hog with n s-sided dice? ($n > 0$ and for us, $s > 0$ is 4 or 6)

$$\frac{\text{\# rolls of } n \ s\text{-sided dice totaling} \geq k}{s^n}$$

- If $k \leq 1$, then clearly the numerator is just s^n.
- For $k > 1$, we consider only rolls that include dice values 2–s, since any 1-die "pigs out." Let's call this quantity rolls2(k, n, s).
- The number of ways to score $\geq k$ is 0 if $\underline{ns < k}$. This is a base case.
- If $n > 0$ then the number of ways to score at least $k \leq 1$ with n dice none of which is 1 is _______. This is also a base case.
- If the first die comes up d ($2 \leq d \leq s$), then there are ________________ ways to throw the remaining $n-1$ dice to get a total of at least k with all n dice.
- This gives us a tree recursion. How would you modify it for the "swine swap" rule?

Another Tree Recursion: Hog Dice

- What are the odds of rolling at least k in hog with n s-sided dice? ($n > 0$ and for us, $s > 0$ is 4 or 6)

$$\frac{\text{\# rolls of } n \ s\text{-sided dice totaling} \geq k}{s^n}$$

- If $k \leq 1$, then clearly the numerator is just s^n.
- For $k > 1$, we consider only rolls that include dice values 2–s, since any 1-die "pigs out." Let's call this quantity rolls2(k, n, s).
- The number of ways to score $\geq k$ is 0 if $ns < k$. This is a base case.
- If $n > 0$ then the number of ways to score at least $k \leq 1$ with n dice none of which is 1 is $(s-1)^n$. This is also a base case.
- If the first die comes up d ($2 \leq d \leq s$), then there are ________________ ways to throw the remaining $n-1$ dice to get a total of at least k with all n dice.
- This gives us a tree recursion. How would you modify it for the "swine swap" rule?

Another Tree Recursion: Hog Dice

- What are the odds of rolling at least k in hog with n s-sided dice? ($n > 0$ and for us, $s > 0$ is 4 or 6)

$$\frac{\text{\# rolls of } n \ s\text{-sided dice totaling} \geq k}{s^n}$$

- If $k \leq 1$, then clearly the numerator is just s^n.
- For $k > 1$, we consider only rolls that include dice values 2–s, since any 1-die "pigs out." Let's call this quantity rolls2(k, n, s).
- The number of ways to score $\geq k$ is 0 if $\underline{ns < k}$. This is a base case.
- If $n > 0$ then the number of ways to score at least $k \leq 1$ with n dice none of which is 1 is $\underline{(s-1)^n}$. This is also a base case.
- If the first die comes up d ($2 \leq d \leq s$), then there are rolls2(k - d, n - 1, s) ways to throw the remaining $n-1$ dice to get a total of at least k with all n dice.
- This gives us a tree recursion. How would you modify it for the "swine swap" rule?

Back to Numeric Pairs: Find the Number

- A *numeric pair* is either an empty tuple, an integer, or a tuple consisting of two numeric pairs (slight revision from last time).
- Problem: does the number x occur in a given numeric pair?

```
def occurs(x, pair):
    """X occurs at least once in numeric pair PAIR.
    >>> occurs(3, ((2, 1), ((), (3, ()))))
    True
    >>> occurs(5, ((2, 1), ((), (3, ()))))
    False
    """
    if _________:
        return True
    elif ______________________________:
        return False
    else:
        return ________________________________________
```

- What is the time required by this function proportional to? *A:*

Back to Numeric Pairs: Find the Number

- A *numeric pair* is either an empty tuple, an integer, or a tuple consisting of two numeric pairs (slight revision from last time).
- Problem: does the number x occur in a given numeric pair?

```
def occurs(x, pair):
    """X occurs at least once in numeric pair PAIR.
    >>> occurs(3, ((2, 1), ((), (3, ()))))
    True
    >>> occurs(5, ((2, 1), ((), (3, ()))))
    False
    """
    if x == pair:
        return True
    elif ______________________________:
        return False
    else:
        return _______________________________________
```

- What is the time required by this function proportional to? *A:*

Back to Numeric Pairs: Find the Number

- A *numeric pair* is either an empty tuple, an integer, or a tuple consisting of two numeric pairs (slight revision from last time).
- Problem: does the number x occur in a given numeric pair?

```
def occurs(x, pair):
    """X occurs at least once in numeric pair PAIR.
    >>> occurs(3, ((2, 1), ((), (3, ()))))
    True
    >>> occurs(5, ((2, 1), ((), (3, ()))))
    False
    """
    if x == pair:
        return True
    elif pair == () or type(pair) is int:
        return False
    else:
        return ______________________________
```

- What is the time required by this function proportional to? *A*:

Back to Numeric Pairs: Find the Number

- A *numeric pair* is either an empty tuple, an integer, or a tuple consisting of two numeric pairs (slight revision from last time).
- Problem: does the number x occur in a given numeric pair?

```
def occurs(x, pair):
    """X occurs at least once in numeric pair PAIR.
    >>> occurs(3, ((2, 1), ((), (3, ()))))
    True
    >>> occurs(5, ((2, 1), ((), (3, ()))))
    False
    """
    if x == pair:
        return True
    elif pair == () or type(pair) is int:
        return False
    else:
        return occurs(x, pair[0]) or occurs(x, pair[1])
```

- What is the time required by this function proportional to? *A:*

Back to Numeric Pairs: Find the Number

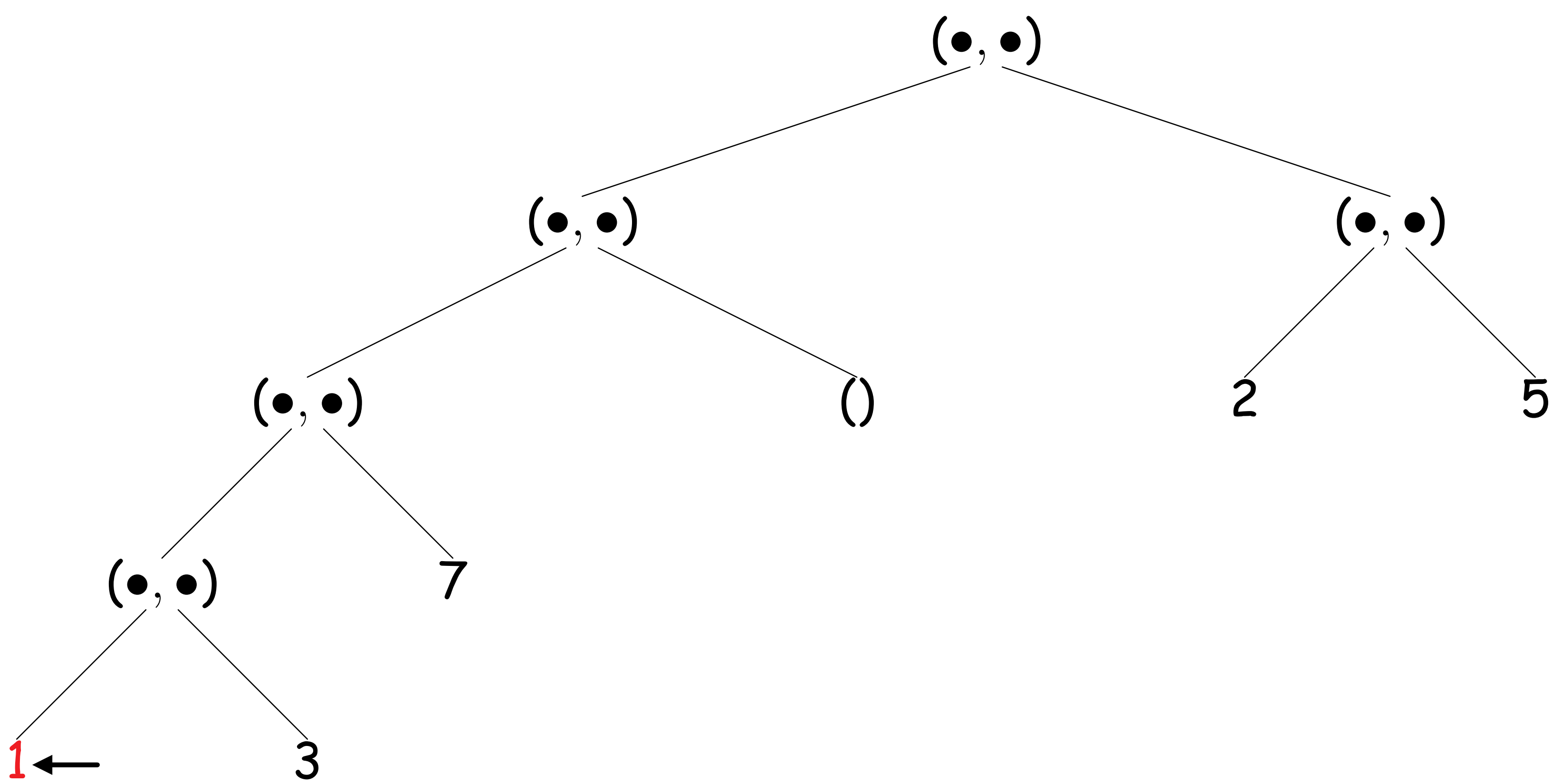
- A *numeric pair* is either an empty tuple, an integer, or a tuple consisting of two numeric pairs (slight revision from last time).
- Problem: does the number x occur in a given numeric pair?

```
def occurs(x, pair):
    """X occurs at least once in numeric pair PAIR.
    >>> occurs(3, ((2, 1), ((), (3, ()))))
    True
    >>> occurs(5, ((2, 1), ((), (3, ()))))
    False
    """
    if x == pair:
        return True
    elif pair == () or type(pair) is int:
        return False
    else:
        return occurs(x, pair[0]) or occurs(x, pair[1])
```

- What is the time required by this function proportional to? *A*: The total number of tuples and integers in pair.

Numeric Pairs: First Leaf

- A *leaf* in a numeric pair is the empty tuple or an integer.
- Define the *first leaf* as the leftmost leaf in the Python expression that denotes a tree.
- Example: the first leaf of ((((1, 3), 7), ()), (2, 5)) is 1:



First Leaf Code

```
def first_leaf(pair):
    """The first leaf in PAIR, reading left to right.
    >>> first_leaf(())
    ()
    >>> first_leaf(5)
    5
    >>> first_leaf((((3, ()), (2, 1)), ()))
    3
    >>> first_leaf(((((), 3), (2, 1)), ()))
    ()
    """
    if ________________________________:
        return pair
    else:
        return ____________________
```

What kind of a recursive process is this? A: ______________________________

First Leaf Code

```
def first_leaf(pair):
    """The first leaf in PAIR, reading left to right.
    >>> first_leaf(())
    ()
    >>> first_leaf(5)
    5
    >>> first_leaf((((3, ()), (2, 1)), ()))
    3
    >>> first_leaf(((((), 3), (2, 1)), ()))
    ()
    """
    if type(pair) is int or pair == ():
        return pair
    else:
        return ___________________
```

What kind of a recursive process is this? A: ______________________________

First Leaf Code

```
def first_leaf(pair):
    """The first leaf in PAIR, reading left to right.
    >>> first_leaf(())
    ()
    >>> first_leaf(5)
    5
    >>> first_leaf((((3, ()), (2, 1)), ()))
    3
    >>> first_leaf(((((), 3), (2, 1)), ()))
    ()
    """
    if type(pair) is int or pair == ():
        return pair
    else:
        return first_leaf(pair[0])
```

What kind of a recursive process is this? A: ______________________

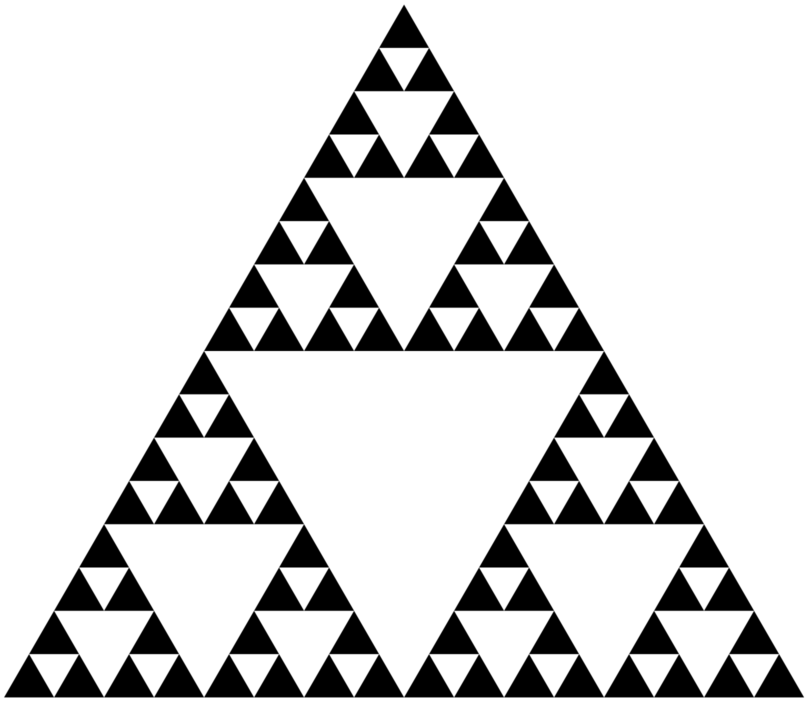
First Leaf Code

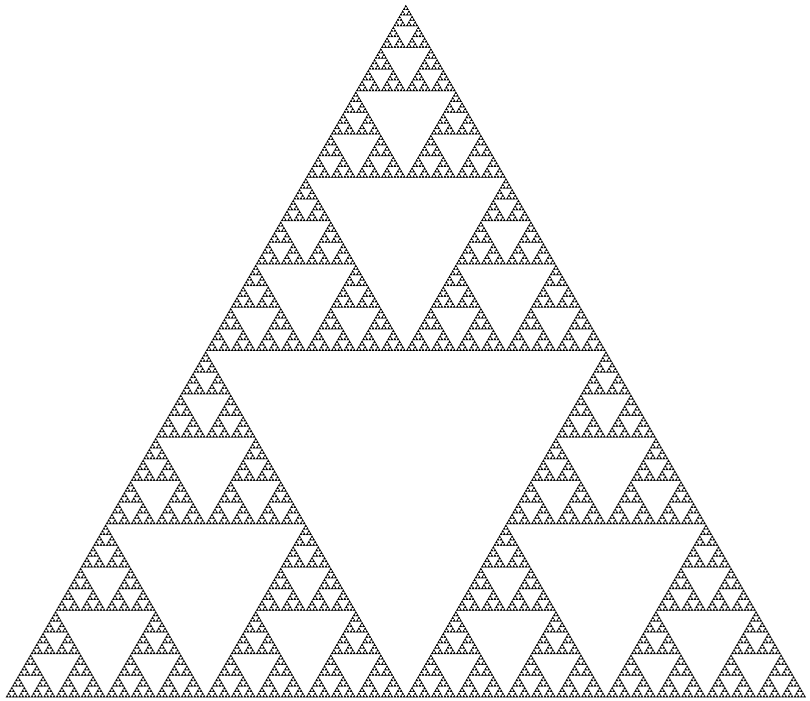
```
def first_leaf(pair):
    """The first leaf in PAIR, reading left to right.
    >>> first_leaf(())
    ()
    >>> first_leaf(5)
    5
    >>> first_leaf((((3, ()), (2, 1)), ()))
    3
    >>> first_leaf(((((), 3), (2, 1)), ()))
    ()
    """
    if type(pair) is int or pair == ():
        return pair
    else:
        return first_leaf(pair[0])
```

What kind of a recursive process is this? *A*: Iterative process (tail recursion)

Sierpinski Triangle

- No discussion of recursion is complete without a mention of *fractal patterns,* which exhibit self-similarity when scaled.
- We'll define a "Sierpinski Triangle of depth k and side s" to be
  - A filled equilateral triangle with sides of length s, if $k = 0$, else
  - Three Sierpinski Triangles of depth $k - 1$ and side $s/2$ arranged in the three corners of an equilateral triangle with side s.
- Here are triangles of degree 4 and 8:





Drawing Sierpinski Triangles

- Assume the existence of the function triangle:

```
def triangle(x, y, side):
    """Draw a filled equilateral triangle with its lower-left corner
    at (X, Y) and with given SIDE.  The base is aligned with the x-axis."""
```

- We can now read off the definition of the triangle:

```
def sierpinski(x, y, side, depth):
    """Draw a Sierpinski triangle of given DEPTH with given SIDE and
    lower-left corner at (X, Y)."""

    if depth == 0:
        ____________________
    else:
        height = 0.25 * sqrt(3) * side

        _________________________________
        ___________________________________________________
        __________________________________________
```

Drawing Sierpinski Triangles

- Assume the existence of the function triangle:

```
def triangle(x, y, side):
    """Draw a filled equilateral triangle with its lower-left corner
    at (X, Y) and with given SIDE.  The base is aligned with the x-axis."""
```

- We can now read off the definition of the triangle:

```
def sierpinski(x, y, side, depth):
    """Draw a Sierpinski triangle of given DEPTH with given SIDE and
    lower-left corner at (X, Y)."""

    if depth == 0:
        triangle(x, y, side)
    else:
        height = 0.25 * sqrt(3) * side

        _________________________________
        ___________________________________________________
        __________________________________________
```

Drawing Sierpinski Triangles

- Assume the existence of the function triangle:

```
def triangle(x, y, side):
    """Draw a filled equilateral triangle with its lower-left corner
    at (X, Y) and with given SIDE.  The base is aligned with the x-axis."""
```

- We can now read off the definition of the triangle:

```
def sierpinski(x, y, side, depth):
    """Draw a Sierpinski triangle of given DEPTH with given SIDE and
    lower-left corner at (X, Y)."""

    if depth == 0:
        triangle(x, y, side)
    else:
        height = 0.25 * sqrt(3) * side

        sierpinski(x, y, side/2, depth-1)
        ______________________________________________________
        ____________________________________________
```

Drawing Sierpinski Triangles

- Assume the existence of the function triangle:

```
def triangle(x, y, side):
    """Draw a filled equilateral triangle with its lower-left corner
    at (X, Y) and with given SIDE.  The base is aligned with the x-axis."""
```

- We can now read off the definition of the triangle:

```
def sierpinski(x, y, side, depth):
    """Draw a Sierpinski triangle of given DEPTH with given SIDE and
    lower-left corner at (X, Y)."""

    if depth == 0:
        triangle(x, y, side)
    else:
        height = 0.25 * sqrt(3) * side

        sierpinski(x, y, side/2, depth-1)
        sierpinski(x + side/4, y + height, side/2, depth-1)
        ____________________________________________
```

Drawing Sierpinski Triangles

- Assume the existence of the function triangle:

```
def triangle(x, y, side):
    """Draw a filled equilateral triangle with its lower-left corner
    at (X, Y) and with given SIDE.  The base is aligned with the x-axis."""
```

- We can now read off the definition of the triangle:

```
def sierpinski(x, y, side, depth):
    """Draw a Sierpinski triangle of given DEPTH with given SIDE and
    lower-left corner at (X, Y)."""

    if depth == 0:
        triangle(x, y, side)
    else:
        height = 0.25 * sqrt(3) * side

        sierpinski(x, y, side/2, depth-1)
        sierpinski(x + side/4, y + height, side/2, depth-1)
        sierpinski(x + side/2, y, side/2, depth-1)
```