Lecture 12, 01 November 2021

Set comprehension

- Defining new sets from old
- $\{x^2 \mid x \in \mathbb{Z}, x \geq 0 \wedge (x \bmod 2) = 0\}$
  - $x \in \mathbb{Z}$, generating set
  - $x \geq 0 \wedge (x \bmod 2) = 0$, filtering condition
  - x^2, output transformation
- More generally $\{f(x) \mid x \in S, p(x)\}$
  - generating set S
  - filtering predicate $p()$
  - transformer function $f()$

Can do this manually for lists

- List of squares of even numbers from 0 to 19

Saved successfully!

- Run through a loop and append elements to output list

```
1 evenlist = []
2 for i in range(20):
3   if i % 2 == 0:
4     evenlist.append(i)
5 print(evenlist)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Operating on each element of a list

- `map(f,l)` applies a function `f` to each element of a list `l`
- `filter(p,l)` extracts elements `x` from `l` for which `p(x)` is `True

```
 1 def even(x):
 2   return(x%2 == 0)
 3
 4 def odd(x):
 5   return(not(even(x)))
 6
 7 def square(x):
 8   return(x*x)
 9
10 N = 20
11 l1 = list(range(N))
12 l2 = list(filter(odd,l1))  # Note that we can pass a function name as an argument
13 l3 = list(map(square,l1))
14
15 # Combine map and filter
16 l4 = list(map(square,filter(even,l1)))
```

```
1 l4
```

```
[0, 4, 16, 36, 64, 100, 144, 196, 256, 324]
```

List comprehension

- `[ f(x) for x in ... if p(x) ]`

```
[ square(x) for x in range(20) if even(x) ]
```

```
[0, 4, 16, 36, 64, 100, 144, 196, 256, 324]
```

```
# A zero vector of length N
[ 0 for i in range(20)]   # The map function can be a constant function
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

- List comprehension can be nested
- A 2 dimensional list : A list of M lists of N zeros

```
M,N = 3,5
onedim = [ 0 for i in range(N)]    #  A list of N zeros
twodim = [ [0 for i in range(N)] for j in range(M) ]
```

Saved successfully!

```
([0, 0, 0, 0, 0], [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]])
```

All Pythagorean triples with value less than n

- (x, y, z) such that $x^2 + y^2 = z^2, x, y, z \leq n$

Pythagorean triples via list comprehension

- Multiple generators
- Start generator for `y` at `x` to avoid enumerating duplicates, like `(3,4,5)` and `(4,3,5)`

```
N = 15
[ (x,y,z) for x in range(1,N+1) for y in range(x,N+1) for z in range(1,N+1) if x*x
```

```
[(3, 4, 5), (5, 12, 13), (6, 8, 10), (9, 12, 15)]
```

Double-click (or enter) to edit

Multiple generators behave like nested loops

```
for x in range(1,N+1):
  for y in range(x,N+1):
    for z in range(1,N+1):
```

List comprehension notation is compact and useful in a number of contexts

- Pull out all dictionary values where the keys satisfy some property: e.g. all marks below 50
  - `[ d[k] for k in d.keys() if p(k) ]`
- Symmetrically, keys whose values satisfy some property: e.g. all roll numbers where marks are below 50
  - `[ k for k in d.keys() if p(d[k]) ]`
- Or, extract (key,value) pairs of interest
  - `[ (k,d[k]) for k in d.keys() if p(d[k]) ]`

Inductive definitions

- $0! = 1$
- $n! = n \times (n-1)!$

- $fib(0) = 0$
- $fib(1) = 1$
- $fib(n) = fib(n-1) + fib(n-2)$

```
1 def factorial(n):
2   if n == 0:
3     return(1)
4   else:
5     return(n*factorial(n-1))  # Recursive call
```

```
1 def fib(n):
2   if n == 0:
3     return(0)
4   elif n == 1:
```

Saved successfully!

```
7     return(fib(n-1)+fib(n-2))
```

Can also do induction on "structures"

- A list consists of the first element and the rest
- Base case is usually the empty list `[]`
- May occasionally also have a base case for a singleton list

```
1 def mylength(l):
2   if l == []:
3     return(0)
4   else:
5     return(1 + mylength(l[1:]))
```

```
1 mylength(list(range(900)))
```

```
900
```

```
1 def mysum(l):
2     if l == []:
3         return(0)
4     else:
5         return(l[0] + mysum(l[1:]))
```

```
1 mysum(list(range(10)))
```

```
45
```

```
1 mysum(['the','long','road'])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-9dbb13abec25> in <module>()
----> 1 mysum(['the','long','road'])

 2 frames 
<ipython-input-13-8e107360d2ff> in mysum(l)
      3         return(0)
```

- Problem is `'the'+'long'+'road'+0`
- Could try to fix this by querying types from within the code, but we won't bother

```
# Can query types
x = [3,3,4]
y = 5
type(x) == type([])  # Compare type() with a "known" type
```

```
True
```

Ascending and descending

- Check if a list is in ascending order, `l[0] < l[1] < ...`
- Similarly, descending, `l[0] > l[1] > ...`

Saved successfully!

- Base case, `len(l)` is 0 or 1, nothing to check
  - Otherwise, check first pair 'l[0] < l[1]`
  - Inductively check that that remaining list `l[1:]` is also ascending

```
def ascending(l):
  if len(l) <= 1:
    return(True)

  else:
    return(l[0] < l[1] and ascending(l[1:]))
    # l[0] < l[1] and l[1] < l[2] and l[2] < l[3] and ...

def descending(l):
  if len(l) <= 1:
    return(True)
  else:
    return(l[0] > l[1] and descending(l[1:]))


```

Zigzag

- Alternate between ascending and descending
- Two possibilities
  - up-down-up-down..., `[1,3,2,7,1,5]`
  - down-up-down-up..., `[8,2,18,-5,7,2,8]`

Up-down

- If `len(l)` is 0 or 1, nothing to do
- Up-down unit repeats after two elements
- 2 element list, check up-down-up
- Recursively check that `l[2:]` is also up-down

Down-up is symmetric

Combine to get zigzag

```
def updown(l):
  if len(l) <= 1:
    return(True)
  elif len(l) == 2:
    return(l[0] < l[1])
  else:
    return(l[0] < l[1] and l[1] > l[2] and updown(l[2:]))

def downup(l):
  if len(l) <= 1:
    return(True)
  elif len(l) == 2:
    return(l[0] > l[1])
  else:
    return(l[0] > l[1] and l[1] < l[2] and downup(l[2:]))

def zigzag(l):
  return(updown(l) or downup(l))
```

Saved successfully!

```
updown(l1), downup(l1), zigzag(l1)
```

```
(True, False, True)
```

```
l2 = [2,1,3,1,4,1]
updown(l2), downup(l2), zigzag(l2)
```

```
(False, True, True)
```

Mutual recursion

- Can define `updown` and `downup` in terms of each other
- **Mutual recursion** -- to be discussed next time

```
def zigzag(l):
    return(updown(l) or downup(l))

def updown(l):
    if len(l) < 2:
        return(True)
    else:
        return(l[0] < l[1] and downup(l[1:]))

def downup(l):
    if len(l) < 2:
        return(True)
    else:
        return(l[0] > l[1] and updown(l[1:]))
```

```
zigzag([0,1,0,1,0])
```

```
True
```

```
zigzag([1,0,1,0,1])
```

```
True
```

For next time, think how to do these inductively

- `insert(l,v)` inserts `v` in the correct position in a sorted list (ascending) `l`
- `delete(l,v)` deletes first `v` in `l` (if any)