PDSP 2024, Lecture 05, 22 August 2024

Lists

- Sequences of values, indexed by position
- For a list with `n` values, valid positions are `0` to `n-1`
  - `len(l)` gives the length of a list
- Accessing a position beyond `len(l)-1` results in `IndexError`

In [1]:
```
l = list(range(20,40))
```

In [2]:
```
len(l), l[3], l[19]
```

Out[2]:
```
(20, 23, 39)
```

In [3]:
```
l[20]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[3], line 1
----> 1 l[20]

IndexError: list index out of range
```

- What about indices below `0`?
- Index `-j` is interpreted as `len(l)-j`
  - Useful for accessing values from the end of the list
  - Valid indices in reverse are `-1`, `-2`, ..., `-len(l)`

In [4]:
```
l[-1], l[-20]
```

Out[4]:
```
(39, 20)
```

Slices

- Recall that `nprimes(n)` computed the first `n` primes

In [5]:
```
def isprime(n):
    for j in range(2,n):
        if n % j == 0:
            return(False)
    return(True)

def nprimes(n):
    plist = []
    j = 2
    while (len(plist) < n):
        if isprime(j):
            plist.append(j)
        j = j+1
    return(plist)
```

In [6]:
```
first20primes = nprimes(20)
```

In [7]:
```
first20primes
```

Out[7]: `[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71]`

- What are the primes from 11 to 15?
- Need a sublist of the original list
- `l[i:j]` is the list `[l[i], l[[i+1], ..., l[j-1]]`
- Similar to

```
newl = []
for k in range(i,j):
   newl.append(k)
```

In [8]:
```
first20primes[11:16]
```

Out[8]: `[37, 41, 43, 47, 53]`

- like `range()` if the indices don't make sense, you get an empty list

In [9]:
```
first20primes[11:10]
```

Out[9]: `[]`

- Unlike accessing `l[i]`, can give upper bound beyond the list
- `l[i:len(l)+10]` is interpreted as `l[i:len(l)]`

In [10]:
```
first20primes[11:40]
```

Out[10]: `[37, 41, 43, 47, 53, 59, 61, 67, 71]`

- Can omit the upper bound, defaults to `len(l)`

In [11]:
```
first20primes[15:]
```

Out[11]: `[53, 59, 61, 67, 71]`

- Likewise, omit the lower bound, defaults to `0`

In [12]:
```
first20primes[:10]
```

Out[12]: `[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]`

- Omit both lower and upper bound to get a `full slice`
  - Full slice returns a new list that is a copy of the list
  - Significance will become clearer later

In [13]:
```
first20primes[:]
```

Out[13]: `[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71]`

More about `range()`

- `range(i,j)` generates the sequence `i,i+1,...,j-1`
- `range(n)` generates the sequence `0,1,...,n-1` -- implicitly starts with `0`
- What if we want to skip over some numbers
  - All even numbers from `4` to `40`

- Optional third argument is the step size
  - `range(i,j,k)` is `i,i+k,...,i+mk` for the largest `m` such that `i+mk < j` and `i+(m+1)k >= j`

In [14]:
```
list(range(4,41,2)) # Even numbers from 4 to 40
```

Out[14]:
```
[4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40]
```

- Can also count down -- give a negative step!

In [15]:
```
list(range(10,0,-1))
```

Out[15]:
```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

- `range(i,j,k)` generate `i, i+k,...` so that the sequence does not *cross* `j`
- Depending on whether it is increasing or decreasing, the last value will be less than `j` or greater than `j`

Stepped slices

- Can similarly give a third argument in a slice

In [16]:
```
first20primes[2:20:3]
```

Out[16]:
```
[5, 13, 23, 37, 47, 61]
```

In [17]:
```
first20primes[19:0:-1]
```

Out[17]:
```
[71, 67, 61, 59, 53, 47, 43, 41, 37, 31, 29, 23, 19, 17, 13, 11, 7, 5, 3]
```

- Explain the following output. (Hint, what is `l[-1]`?)

In [18]:
```
first20primes[19:-1:-1]
```

Out[18]:
```
[]
```

- Can omit upper and lower bounds but give a step
- `l[::-1]` is the entire list in reverse
  - Note that the default lower and upper bound are determined by the step

In [19]:
```
first20primes[::-1]
```

Out[19]:
```
[71, 67, 61, 59, 53, 47, 43, 41, 37, 31, 29, 23, 19, 17, 13, 11, 7, 5, 3, 2]
```

Assigning slices

- Can assign a list to a slice

In [20]:
```
l = list(range(20,30))
```

In [21]:
```
l[3:5] = [53,54,55]
```

In [22]:
```
l
```

Out[22]:
```
[20, 21, 22, 53, 54, 55, 25, 26, 27, 28, 29]
```

- Can contract or expand the slice when reassigning
- Indices of values to the right will change

In [23]:
```
l = list(range(20,30))
l[3:5] = [53,54,55,63,64,65]
```

In [24]:
```
l
```

Out[24]:
```
[20, 21, 22, 53, 54, 55, 63, 64, 65, 25, 26, 27, 28, 29]
```

In [25]:
```
l = list(range(20,30))
l[3:5] = []
```

In [26]:
```
l
```

Out[26]:
```
[20, 21, 22, 25, 26, 27, 28, 29]
```

Operations on lists

- Recall that `+` concatenates two lists

In [27]:
```
l1 = [1,2,3]
l2 = [4,5,6]
l3 = l1 + l2
```

In [28]:
```
l3
```

Out[28]:
```
[1, 2, 3, 4, 5, 6]
```

- A useful invariant about slices
- For any list `l`, and any integer `j`, `l == l[:j] + l[j:]`

In [29]:
```
l3[:-1]+l3[-1:]
```

Out[29]:
```
[1, 2, 3, 4, 5, 6]
```

In [30]:
```
l3[:2]+l3[2:]
```

Out[30]:
```
[1, 2, 3, 4, 5, 6]
```

In [31]:
```
l3[:9]+l3[9:]
```

Out[31]:
```
[1, 2, 3, 4, 5, 6]
```

Applying functions to lists

- `l.append(v)` is the same as `l = l+[v]`
  - Ask the list `l` to append `v` to itself
  - `l.append(v)` updates `l` in place
  - `l = l+[v]` creates a new list and reassigns the list pointed to by `l`
  - Again, we will see the significance of this later

In [32]:
```
l3.append(7)
```

In [33]:
```
l3
```

Out[33]:
```
[1, 2, 3, 4, 5, 6, 7]
```

Other functions

- `l.insert(pos,val)` inserts `val` at position `p`
  - Similar to `l = l[:pos] + [val] + l[pos:]`
- `l.extend(newl)` extends `l` with a list of values `newl`
  - Similar to `l = l + newl`

In [34]:
```
l3.insert(0,0)
```

In [35]:
```
l3
```

Out[35]:
```
[0, 1, 2, 3, 4, 5, 6, 7]
```

In [36]:
```
l3.extend([8,9,10])
```

In [37]:
```
l3
```

Out[37]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Sorting

- `l.sort()` sorts a list in place
  - Python allows lists of mixed types
  - To sort a list, the values must be of a uniform comparable type

In [38]:
```
l = [1,15,3,7,9,2]
```

In [39]:
```
l.sort()
```

In [40]:
```
l
```

Out[40]:
```
[1, 2, 3, 7, 9, 15]
```

In [41]:
```
badl = [1,"CSK",True,7.5]
```

In [42]:
```
badl = [1,'CSK',True,7.5]
```

In [43]:
```
badl.sort()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[43], line 1
----> 1 badl.sort()

TypeError: '<' not supported between instances of 'str' and 'int'
```

In [44]:
```
blist = [True,False]
```

In [45]:
```
blist.sort()
```

In [46]:
```
blist
```

Out[46]:
```
[False, True]
```

- If you want a sorted copy of `l` without disturbing `l`, use `sorted(l)`

```
In [47]: l = [15, 1000, 9, 7, 3, 2, 1]
```

```
In [48]: l
```

```
Out[48]: [15, 1000, 9, 7, 3, 2, 1]
```

```
In [49]: sorted(l)
```

```
Out[49]: [1, 2, 3, 7, 9, 15, 1000]
```

```
In [50]: l
```

```
Out[50]: [15, 1000, 9, 7, 3, 2, 1]
```

- Can we copy `l` and sort the copy?

```
In [51]: newl = l
```

```
In [52]: newl.sort()
```

```
In [53]: newl
```

```
Out[53]: [1, 2, 3, 7, 9, 15, 1000]
```

```
In [54]: l
```

```
Out[54]: [1, 2, 3, 7, 9, 15, 1000]
```

- Sorting `newl` also sorts `l`
- This does not happen with types like `int`
- We will investigate this later

```
In [55]: y = 7
```

```
In [56]: x = y
```

```
In [57]: x, y
```

```
Out[57]: (7, 7)
```

```
In [58]: x = 17
```

```
In [59]: x, y
```

```
Out[59]: (17, 7)
```

- Many other built-in functions on lists
  - `l.reverse()` reverses a list
- Look up Python documentation