PDSP 2026, Lecture 05, 25 August 2026

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`
- 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)`

Slices

- `l[i:j]` returns the list `[l[i], l[i+1], ..., l[j-1]]`
- `l[i:]` -- missing upper bound defaults to `len(l)`
- `l[:i]` -- missing lower bound defaults to `0`
- `l[:]` -- *full* slice, returns a copy of the entire list

More about `range()`

- 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`
  - Step can be negative

Stepped slices

- Can similarly give a third argument in a slice -- `l[i:j:k]`
- 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

Assigning slices

- Can assign a list to a slice

In [1]:
```
l = list(range(20,30))
```

In [2]:
```
l
```

Out[2]:
```
[20, 21, 22, 23, 24, 25, 26, 27, 28, 29]
```

In [3]:
```
l[3:6] = [53,54,55]
```

In [4]:
```
l
```

Out[4]:
```
[20, 21, 22, 53, 54, 55, 26, 27, 28, 29]
```

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

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

In [6]:
```
l
```

Out[6]:
```
[20, 21, 22, 53, 54, 55, 63, 64, 65, 25, 26, 27, 28, 29]
```

In [7]:
```
l = list(range(20,30))
l[3:5] = []
```

In [8]:
```
l
```

Out[8]:
```
[20, 21, 22, 25, 26, 27, 28, 29]
```

Operations on lists

- Recall that `+` concatenates two lists
- Returns a new list
- Original lists are unchanged

In [9]:
```
l1 = [1,2,3]
l2 = [4,5,6]
l3 = l1 + l2
```

In [10]:
```
l3, l1, l2
```

Out[10]:
```
([1, 2, 3, 4, 5, 6], [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 [11]:
```
l3[:-1]+l3[-1:]
```

Out[11]:
```
[1, 2, 3, 4, 5, 6]
```

In [12]:
```
l3[:2]+l3[2:]
```

Out[12]:
```
[1, 2, 3, 4, 5, 6]
```

In [13]:
```
l3[:9]+l3[9:]
```

Out[13]:
```
[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 [14]: l3.append(7)

In [15]: l3

Out[15]: [1, 2, 3, 4, 5, 6, 7]

In [16]: l4 = l3 + [8]

In [17]: l4

Out[17]: [1, 2, 3, 4, 5, 6, 7, 8]

In [18]: l4 = l4 + [9]

In [19]: l4

Out[19]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- What happens if we reassign a list after an `append()`?

```
In [20]: l3 = l3.append(8)

In [21]: l3
```

- It is a mistake to reassign a list after an `append()`

- Assignment `v = e` stores return value of `e` in `v`
- Return value of `l.append()` is empty

- Did not say `NameError`

```
In [22]: p
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[22], line 1
----> 1 p

NameError: name 'p' is not defined
```

- More specifically, `l.append()` returns a special value `None`, that stands for "no value", but assigning to `None` is different from not assigning at all
- Jupyter notebook does not display `None` as the return value, but try `print(l3.append(8))`

Other functions

- `l.insert(pos,val)` inserts `val` at position `pos`
  - Similar to `l = l[:pos] + [val] + l[pos:]`
- `l.extend(newl)` extends `l` with a list of values `newl`
  - Similar to `l = l + newl`
- Like `l.append(v)`, these update the list in place, do not reassign return value

```
In [23]: l3 = [1,2,3,4,5,6,7]
```

```
In [24]: l3.insert(0,0)
```

```
In [25]: l3
```

```
Out[25]: [0, 1, 2, 3, 4, 5, 6, 7]
```

```
In [26]: l3.extend([8,9,10])
```

```
In [27]: l3
```

```
Out[27]: [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 [28]: l = [1,15,3,7,9,2]
```

```
In [29]: l.sort()
```

```
In [30]: l
```

```
Out[30]: [1, 2, 3, 7, 9, 15]
```

- Python lists can have values of different types

```
In [31]: badl = [1,'CSK',True,7.5]
```

- However, functions like `sort()` require all values to have compatible types -- in this case, to compare values sensibly

```
In [32]: badl.sort()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[32], line 1
----> 1 badl.sort()

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

- Can sort values of type `bool`, for example

```
In [33]: blist = [True,False,True]
```

```
In [34]: blist.sort()
```

```
In [35]: blist
```

```
Out[35]: [False, True, True]
```

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

```
In [36]: l = [15, 1000, 9, 7, 3, 2, 1]
```

```
In [37]: l
```

```
Out[37]: [15, 1000, 9, 7, 3, 2, 1]
```

```
In [38]: sorted(l)
```

```
Out[38]: [1, 2, 3, 7, 9, 15, 1000]
```

```
In [39]: l
```

```
Out[39]: [15, 1000, 9, 7, 3, 2, 1]
```

- Can store a copy of the sorted list in another list

```
In [40]: newl = sorted(l)
```

```
In [41]: newl, l
```

```
Out[41]: ([1, 2, 3, 7, 9, 15, 1000], [15, 1000, 9, 7, 3, 2, 1])
```

- Can we, instead, first copy `l` and sort the copy in place using `sort()`?

```
In [42]: newl = l
```

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

In [44]:
```
newl
```

Out[44]:
```
[1, 2, 3, 7, 9, 15, 1000]
```

In [45]:
```
l
```

Out[45]:
```
[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 [46]:
```
y = 7
```

In [47]:
```
x = y
```

In [48]:
```
x, y
```

Out[48]:
```
(7, 7)
```

In [49]:
```
x = 17
```

In [50]:
```
x, y
```

Out[50]:
```
(17, 7)
```

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

Tuples

- Sequence of values in round brackets - `(v1,v2,...,vk)`
- Typically values are not of a uniform type
  - Collect together different attributes of an item
  - Row in a table; each column is an attribute

In [51]:
```
t = ("Mexico City","MEX","RSA",2,0,0,0,1.84,0.52)
```

- Positional indexing, slicing like other sequences

In [52]:
```
t[2],t[-1]
```

Out[52]:
```
('RSA', 0.52)
```

In [53]:
```
t[1:4]
```

Out[53]:
```
('MEX', 'RSA', 2)
```

- Iterate over a tuple
  - `print()` prints its arguments --- either a message (string) or value of a variable

In [54]:
```
for x in t:
    print("x is", x, ", to repeat", x)
```

```
x is Mexico City , to repeat Mexico City
x is MEX , to repeat MEX
x is RSA , to repeat RSA
x is 2 , to repeat 2
x is 0 , to repeat 0
x is 0 , to repeat 0
x is 0 , to repeat 0
x is 1.84 , to repeat 1.84
x is 0.52 , to repeat 0.52
```

In [55]:
```
'BRA' in t, 'MEX' in t
```

Out[55]:
```
(False, True)
```

In [56]:
```
print(t)
```

```
('Mexico City', 'MEX', 'RSA', 2, 0, 0, 0, 1.84, 0.52)
```

- Unlike lists, cannot update a component of a tuple

In [57]:
```
t = ("Mexico City","MEX","RSA",2,0,0,0,1.84,0.52)
```

In [58]:
```
t[2] = 'BRA'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[58], line 1
----> 1 t[2] = 'BRA'

TypeError: 'tuple' object does not support item assignment
```

- Can concatenate tuples using `+`
  - Update a tuple by assembling a new tuple
- Be careful, insert a comma to indicate a singleton tuple: `(v,)` vs `(v)`

In [59]:
```
u = t[0:2]+('BRA')+t[3:]  # No difference between 'BRA' and ('BRA')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[59], line 1
----> 1 u = t[0:2]+('BRA')+t[3:]  # No difference between 'BRA' and ('BRA')

TypeError: can only concatenate tuple (not "str") to tuple
```

In [60]:
```
u = t[0:2]+('BRA',)+t[3:]  # ('BRA',) is recognized a singleton tuple
```

In [61]:
```
u
```

Out[61]:
```
('Mexico City', 'MEX', 'BRA', 2, 0, 0, 0, 1.84, 0.52)
```

- Can use `list()` to convert a tuple to a list

In [62]:
```
list(t)
```

Out[62]:
```
['Mexico City', 'MEX', 'RSA', 2, 0, 0, 0, 1.84, 0.52]
```

In [63]:
```
type(u)
```

Out[63]:
```
tuple
```

In [64]:
```
type(list(u))
```

Out[64]:
```
list
```

- In general `list()` works provided its argument is a sequence

In [65]:
```
list(range(5))
```

Out[65]:
```
[0, 1, 2, 3, 4]
```

- If the argument is not a sequence, `list()` generates an error

In [66]:
```
list(7)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[66], line 1
----> 1 list(7)

TypeError: 'int' object is not iterable
```

- Any type name can be used as a function to convert to that type
- However, the argument needs to have a sensible interpretation in the target type

In [67]:
```
int(7.5)
```

Out[67]:
```
7
```

In [68]:
```
int("B")
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
Cell In[68], line 1
----> 1 int("B")

ValueError: invalid literal for int() with base 10: 'B'
```

- Notice the error specifically refers to `base 10`
- `int()` takes an optional second argument, specifying the base. In base 16, the values 10,11,...,15 are denoted A,B,...,F

In [69]:
```
int("B",16)
```

Out[69]:
```
11
```

- Can assign a tuple of variables in one shot
  - Useful for initialising multiple quantities
  - In `nprimes()` we started with `primelist = []` and `p = 2`

In [70]:
```
(primelist,p) = ([],2)
```

In [71]:
```
# Equivalent to
primelist = []
p = 2
```

- `(x,y) = (y,x)` swaps the values of `x` and `y`
  - All values on rhs are old values
  - All values on lhs are new assignments
  - Cannot be done sequentially
    - Not equivalent to `x = y` followed by `y = x` or vice versa
- Normally, swap requires a temporary variable

```
t = y
y = x
x = t
```

- Imagine exchanging the contents of a glass of juice and a glass of milk
  - Need a third empty glass

In [72]:
```
(x,y) = (5,[])
(x,y) = (y,x)
```

In [73]:
```
x,y
```

Out[73]:
```
([], 5)
```

- When we say `x,y` we mean `(x,y)` --- brackets may be omitted

- Python inserts them to display the value to us

In [74]:
```
x,y = 5,[]
```

In [75]:
```
x,y
```

Out[75]:
```
(5, [])
```

Dictionaries

- A list is a collection indexed by position
- A list can be thought of as a function $f : \{0, 1, \ldots, n-1\} \to \{v_0, v_1, \ldots, v_{n-1}\}$
  - A list maps positions to values
- Generalize this to a function $f : \{k_0, k_1, \ldots, k_{n-1}\} \to \{v_0, v_1, \ldots, v_{n-1}\}$
  - Instead of positions, index by an abstract *key*
- **dictionary**: maps keys, rather than positions, to values
- Notation:
  - `d = {k1:v1, k2:v2}`, enumerate a dictionary explicitly
  - `d[k1]`, value in dictionary `d1` corresponding to key `k1`
  - `{}`, empty dictionary (`[]` for lists, `()` for tuples)

In [76]:
```
myd = {'a':1,'b':17,'c':0}
```

In [77]:
```
myd['b']
```

Out[77]:
```
17
```

In [78]:
```
myd['d']  # Invalid key
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[78], line 1
----> 1 myd['d']  # Invalid key

KeyError: 'd'
```

In [79]:
```
myd['d'] = 17
```

In [80]:
```
myd['d']
```

Out[80]:
```
17
```

In [81]:
```
myd
```

Out[81]:
```
{'a': 1, 'b': 17, 'c': 0, 'd': 17}
```

- An assignment `d[k] = v` serves two purposes
  - If there is no key `k`, create the key and assign it the value `v`
  - If there is already a key `k`, replace its current value by `v`
- In a list, we cannot create a value at a new position through an assignment

- If `l` is `[0,1,2,3]`, `l[4] = 4` generate `IndexError`
- If `myd = {'a':1,'b':17,'c':0}`, `myd['d'] = 19` extends `d` with a new key-value pair