PDSP 2024, Lecture 06, 27 August 2024

List membership

- `v in l` returns `True` iff value `v` is in `l`
- Implicit iteration, same as

```
def element(l,v):
  for x in l:
    if x == v:
      return(True)
  return(False)
```

- Linear scan of the list, examine all elements (worst case) if `v` is not in `l`

- Find unique elements in a list

In [1]:
```
def uniq(l):
    newl = []
    for x in l: # newl is unique elements to the left of x
        if not (x in newl):
            newl.append(x)
    return(newl)
```

In [2]:
```
uniq([1,3,5,3,8,5,7,9])
```

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

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 [3]:
```
t = ("Visakhapatnam","DC","CSK","DC","DC",192)
```

- Positional indexing, slicing like other sequences

In [4]:
```
t[4],t[-1]
```

Out[4]:
```
('DC', 192)
```

In [5]:
```
t[1:4]
```

Out[5]:
```
('DC', 'CSK', 'DC')
```

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

In [6]:
```
for x in t:
    print("x is", x, ", to repeat", x)
```

```
x is Visakhapatnam , to repeat Visakhapatnam
x is DC , to repeat DC
x is CSK , to repeat CSK
x is DC , to repeat DC
x is DC , to repeat DC
x is 192 , to repeat 192
```

'RR' in t

In [7]:
```
print(t)
```

```
('Visakhapatnam', 'DC', 'CSK', 'DC', 'DC', 192)
```

- Unlike lists, cannot update a component of a tuple

In [8]:
```
t = ("Visakhapatnam","DC","CSK","DC","DC",192) # Change Team 2 to 'RR' from 'CSK'
```

In [9]:
```
t[2] = 'RR'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[9], line 1
----> 1 t[2] = 'RR'

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 [10]:
```
u = t[0:2]+('RR')+t[3:]  # No difference between 'RR' and ('RR')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[10], line 1
----> 1 u = t[0:2]+('RR')+t[3:]  # No difference between 'RR' and ('RR')

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

In [16]:
```
u = t[0:2]+('RR',)+t[3:]  # ('RR',) is recognized a singleton tuple
```

In [17]:
```
u
```

Out[17]:
```
('Visakhapatnam', 'DC', 'RR', 'DC', 'DC', 192)
```

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

In [18]:
```
list(t)
```

Out[18]:
```
['Visakhapatnam', 'DC', 'CSK', 'DC', 'DC', 192]
```

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

In [19]:
```
list(range(5))
```

Out[19]:
```
[0, 1, 2, 3, 4]
```

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

In [20]:
```
list(7)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[20], line 1
----> 1 list(7)

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

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

In [21]:
```
(primelist,p) = ([],2)
```

In [22]:
```
# 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 [23]:
```
(x,y) = (5,[])
(x,y) = (y,x)
```

In [24]:
```
x,y
```

Out[24]:
```
([], 5)
```

- When we say `x,y` we mean `(x,y)` --- brackets may be omitted
- Python inserts them to display the value to us

In [25]:
```
x,y = 5,[]
```

In [26]:
```
x,y
```

Out[26]:
```
(5, [])
```

Strings

- Also sequences, of characters
- String values can be enclosed in single or double quotes
  - `'Chennai'` or `"Chennai"`
- Allows a value that has a quote to be easily embedded
  - `"Fermat's Last Theorem"`
  - `'He said, "Thank you!"'`

- If you need to use both single and double quotes inside the string, use a triple quote!
  - `'''He said, "That's great!"'''`

In [27]:
```
s = '''He said, "That's great!"'''
```

In [28]:
```
s
```

Out[28]:
```
'He said, "That\'s great!"'
```

- Python prefers to render strings using single quote
- Embedded quotes are "escaped" using `\` to remove their special meaning

- Like lists and tuples, can access elements of a string by position, or by slices

In [29]:
```
s = "hello"
```

In [30]:
```
s[1]
```

Out[30]:
```
'e'
```

In [31]:
```
s[2:4]
```

Out[31]:
```
'll'
```

- Some languages have a separate type `char` for a single character
  - A string is then a sequence of `char`
- In Python, there is only the string type `str`
  - A single character is the same as a string of length 1

In [32]:
```
s[1] == "e"  # Logically speaking, s[1] is a single character
```

Out[32]:
```
True
```

- Concatenate strings using `+`

In [33]:
```
s = "hello"
t = "there"
```

In [34]:
```
s+t
```

Out[34]:
```
'hellothere'
```

- Like tuples, cannot update parts of a string directly

In [35]:
```
s[3] = "p"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[35], line 1
----> 1 s[3] = "p"

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

- Instead, assemble a new string from the old one

In [36]:
```
s = s[0:3] + "p" + s[4:]
```

In [37]:
```
s
```

Out[37]:
```
'helpo'
```

- Can iterative over strings and check membership, like lists
- For iteration, no distinction between a string `"xyz"` and the list `["x","y","z"]`

In [38]:
```
def vowel(c):
    return(c in "aieou")
```

In [39]:
```
vowel("a"),vowel("b")
```

Out[39]:
```
(True, False)
```

In [40]:
```
def countvowels(s):
    count = 0
    for c in s:
        if vowel(c):
            count = count+1
    return(count)
```

In [41]:
```
countvowels("hello")
```

Out[41]:
```
2
```

In [42]:
```
vowel(7)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[42], line 1
----> 1 vowel(7)

Cell In[38], line 2, in vowel(c)
      1 def vowel(c):
----> 2     return(c in "aieou")

TypeError: 'in <string>' requires string as left operand, not int
```

In [43]:
```
vowel("there")
```

Out[43]:
```
False
```

- We have seen that `list()` and `int()` can be use to convert values from one type to another
- Likewise `str()` converts its argument to a string
  - Almost any value converts sensibly into a "readable" representation
  - `print(v)` implicitly converts `v` to `str(v)` to display on screen

In [44]:
```
str([1,2,3])
```

Out[44]:
```
'[1, 2, 3]'
```

In [45]:
```
str(77)
```

Out[45]:
```
'77'
```

- Can convert a string to a number if the contents can be intepreted sensibly

In [46]:
```
int('77')
```

Out[46]: 77

In [47]:
```
int('hello')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[47], line 1
----> 1 int('hello')

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

In [48]:
```
int('77.5')  # 77.5 is a number, but not an int
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[48], line 1
----> 1 int('77.5')  # 77.5 is a number, but not an int

ValueError: invalid literal for int() with base 10: '77.5'
```

In [49]:
```
float('77.5')
```

Out[49]: 77.5

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 [50]:
```
d = {'a':1,'b':17,'c':0}
```

In [51]:
```
d['b']
```

Out[51]: 17

In [52]:
```
d['d']  # Invalid key
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[52], line 1
----> 1 d['d']  # Invalid key

KeyError: 'd'
```

In [53]:
```
d['d'] = 17
```

In [54]:
```
d['d']
```

Out[54]: 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 `d = {'a':1,'b':17,'c':0}`, `d['d'] = 19` extends `d` with a new key-value pair

Iteration

- `d.keys()` generates a sequence of all keys in `d`
  - Iterate over keys using `for k in d.keys():`
  - `for k in d:` also works --- `d` is implicitly intepreted as `d.keys()`
  - Though the keys do not form a sequence, Python will generate them in the order in which they were created
- Similarly, `d.values()` is the sequence of values present

In [55]:
```
d = {'a':1,'b':17,'c':0}
```

In [56]:
```
list(d.keys()), list(d.values())
```

Out[56]:
```
(['a', 'b', 'c'], [1, 17, 0])
```

In [57]:
```
d = {'b':17,'c':0,'a':1}
```

In [58]:
```
list(d.keys()), list(d.values())
```

Out[58]:
```
(['b', 'c', 'a'], [17, 0, 1])
```

Example

- Count frequency of letters in a string
- Maintain a counter for each letter that appears in the string
- Dictionary `freqd` where each key is a letter `c` and `freqd[c]` is a positive integer
  - The first time we see a letter, need to create a key and assign it the value 1
  - If there is already a key for the current letter, increment its count
  - Test if a key `k` is present using `k in d.keys()` (or, shorter, `k in d`)

In [59]:
```
def frequency(s):
    freqd = {}
    for c in s:
        if c in freqd:  # Check if c is already a key
            freqd[c] = freqd[c] + 1
        else:           # Create a new key with count 1
            freqd[c] = 1
    return(freqd)
```

In [60]:
```
d = frequency("hello")
```

In [61]:
```
d
```

Out[61]:
```
{'h': 1, 'e': 1, 'l': 2, 'o': 1}
```