PDSP 2025, Lecture 06, 26 August 2025

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 [1]:
```
t = ("Visakhapatnam","DC","CSK","DC","DC",192)
```

- Positional indexing, slicing like other sequences

In [2]:
```
t[4],t[-1]
```

Out[2]:
```
('DC', 192)
```

In [3]:
```
t[1:4]
```

Out[3]:
```
('DC', 'CSK', 'DC')
```

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

In [4]:
```
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
```

In [5]:
```
'RR' in t, 'DC' in t
```

Out[5]:
```
(False, True)
```

In [6]:
```
print(t)
```

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

- Unlike lists, cannot update a component of a tuple

In [7]:
```
t = ("Visakhapatnam","DC","CSK","DC","DC",192) # Change Team 2 to 'RR' from 'CSK'
```

In [8]:
```
t[2] = 'RR'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[8], 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 [9]:
```
u = t[0:2]+('RR')+t[3:]  # No difference between 'RR' and ('RR')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[9], line 1
----> 1 u = t[0:2]+(     )+t[3:]  # No difference between 'RR' and ('RR')

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

In [10]:
```
u = t[0:2]+('RR',)+t[3:]  # ('RR',) is recognized a singleton tuple
```

In [11]:
```
u
```

Out[11]:
```
('Visakhapatnam', 'DC', 'RR', 'DC', 'DC', 192)
```

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

In [12]:
```
list(t)
```

Out[12]:
```
['Visakhapatnam', 'DC', 'CSK', 'DC', 'DC', 192]
```

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

In [13]:
```
list(range(5))
```

Out[13]:
```
[0, 1, 2, 3, 4]
```

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

In [14]:
```
list(7)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[14], 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 [15]:
```
(primelist,p) = ([],2)
```

In [16]:
```
# 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 [17]:
```
(x,y) = (5,[])
(x,y) = (y,x)
```

In [18]:
```
x,y
```

Out[18]:
```
([], 5)
```

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

In [19]:
```
x,y = 5,[]
```

In [20]:
```
x,y
```

Out[20]:
```
(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 [21]:
```
d = {'a':1,'b':17,'c':0}
```

In [22]:
```
d['b']
```

Out[22]:
```
17
```

In [23]:
```
d['d']  # Invalid key
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[23], line 1
----> 1 d[   ]  # Invalid key

KeyError: 'd'
```

In [24]:
```
d['d'] = 17
```

In [25]:
```
d['d']
```

Out[25]:
```
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 [26]:
```
d = {'a':1,'b':17,'c':0}
```

In [27]:
```
list(d.keys()), list(d.values())
```

Out[27]:
```
(['a', 'b', 'c'], [1, 17, 0])
```

In [28]:
```
d = {'b':17,'c':0,'a':1}
```

In [29]:
```
list(d.keys()), list(d.values())
```

Out[29]:
```
(['b', 'c', 'a'], [17, 0, 1])
```