Lecture 07, 05 September 2023

Operating on dictionaries

- How do we run through all entries in a dictionary - the equivalent of `for x in l`?
- `d.keys()`, `d.values()` generate sequences corresponding to the keys and values of `d`, respectively
- Like `range()` these are not directly lists, use `list(d.keys())` if you want a list

In [1]:
```
marks = {'Physics': 75, 'Maths': 88}
```

In [2]:
```
for k in marks.keys():
    print(k, marks[k])
```

```
Physics 75
Maths 88
```

- Shortcut, can omit `.keys()` when iterating

In [3]:
```
for k in marks:
    print(k)
```

```
Physics
Maths
```

- Likewise can iterate over the values using `d.values()`
- Iterate over the keys in the same order as `d.keys()` but use `d[k]` rather than `k`

In [4]:
```
for k in marks.values():
    print(k)
```

```
75
88
```

- In what order does `d.keys()` list the keys?
- In theory, this order is arbitrary and you should not make any assumptions
- In practice, from some recent version of Python (3.6?) keys are listed in the order added
- If dictionary keys are of the same type, use `sorted(d.keys())` to get them in sorted

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

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

- `sorted(l)` leaves `l` unchanged
- elements of `l` should be of a uniform type to allow comparisons
- strings are sorted in lexicographic (dictionary) order

In [6]:
```
l = [3,1,8,2,9]
```

In [7]:
```
sorted(l)
```

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

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

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

In [9]:
```
sorted(["A",2])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [9], line 1
----> 1 sorted(["A",2])

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

In [10]:
```
sorted(["B","A"])
```

Out[10]:
```
['A', 'B']
```

In [11]:
```
sorted(["1","2","3","10"])
```

Out[11]:
```
['1', '10', '2', '3']
```

In [12]:
```
for k in sorted(marks.keys()):
    print(k,marks[k])
```

```
Maths 88
Physics 75
```

In [13]:
```
for k in sorted(marks):
    print(k,marks[k])
```

```
Maths 88
Physics 75
```

Accumulating values

- We have a list of pairs (name,runs) of runs scored by players in a series
- We want to report the total runs of each student
- Create a dictionary `totalruns` whose keys are names and whose values are total runs for that name
- How would we do this?
  - Check if the current key already exists in `totalruns`
  - If sp, add the current runs to the existing entry
  - Otherwise, create a new entry with the current pair

In [14]:
```
runlist = [("A",10),("C",20),("A",44),("B",33),("B",77)]
```

In [15]:
```
totalruns = {}
for p in runlist:
    name = p[0]
    runs = p[1]
    # totalruns[name] = runs  # Replaces the value
    if name in totalruns:  # Same as --- if name in totalruns.keys():
        totalruns[name] = totalruns[name] + runs
    else:
        totalruns[name] = runs
```

- We can accumulate other quantities for each key -- for instance, the list of runs scored

In [16]:
```
totalrunlist = {}
for p in runlist:
    name = p[0]
    runs = p[1]
    if name in totalrunlist:
        totalrunlist[name] = totalrunlist[name] + [runs]  # or totalrunlist[name].append(runs)
    else:
        totalrunlist[name] = [runs]
```

In [17]:
```
totalruns, totalrunlist
```

Out[17]:
```
({'A': 54, 'C': 20, 'B': 110}, {'A': [10, 44], 'C': [20], 'B': [33, 77]})
```

Basic input and output

- Take input from the keyboard
- Print output to the screen

In [19]:
```
x = input()
```

```
3443
```

- `input()` always returns a string

In [20]:
```
x, type(x)
```

Out[20]:
```
('3443', str)
```

Type conversion

- we have seen `list(range(n))`
- int(s) converts a string s to an int, if it is possible

In [21]:
```
list(range(10)), range(10)
```

Out[21]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], range(0, 10))
```

In [22]:
```
int(x),x
```

Out[22]:
```
(3443, '3443')
```

- Any type name can be used as a type converter

In [23]:
```
str(7888)
```

Out[23]:
```
'7888'
```

- Type conversion works only if argument is of a sensible type
- For instance, `int(x)` requires `x` to be a valid integer

In [24]:
```
int("abcd")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [24], line 1
----> 1 int("abcd")

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

- Optional second argument indicates the base for `int` conversion
- For instance, in base 16, `a` to `f` are legal

In [25]:
```
int("abcd",16)
```

Out[25]:
```
43981
```

In [26]:
```
13 + 12*16 + 11*16*16 + 10*16*16*16
```

Out[26]:
```
43981
```

In [27]:
```
list(7)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [27], line 1
----> 1 list(7)

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

In [28]:
```
list((7,8))
```

Out[28]:
```
[7, 8]
```

- Can provide a "prompt" to `input()` by passing it as a string

In [29]:
```
x = input("Please type a number: ")
```

```
Please type a number: 343
```

In [30]:
```
x = float(x)
```

In [31]:
```
x
```

Out[31]:
```
343.0
```

Output

- `print(x1,x2,...,xn)`
- Implicitly each `xi` is converted to `str(xi)`
- Use `sep=` to modify default space separating values
- Use `end=` to modify default new line after each print

In [32]:
```
for k in marks.keys():
    print(k, marks[k])
```

```
Physics 75
Maths 88
```

In [33]:
```
for k in marks.keys():
    print(k, marks[k], sep=":")
```

```
Physics:75
Maths:88
```

In [34]:
```
for k in marks.keys():
    print(k, marks[k], sep=":",end="")
    print()
```

```
Physics:75
Maths:88
```

In [35]:
```
for k in marks.keys():
    print(k, marks[k], sep=":",end=",")
```

```
Physics:75,Maths:88,
```

- Can intersperse strings in `print()` to make output more readable

In [37]:
```
for k in marks.keys():
    print("Name:", k, ", Marks:", marks[k])
```

```
Name: Physics , Marks: 75
Name: Maths , Marks: 88
```

Multiple assignment using tuples

Useful to initialize many things at the start of a function

In [38]:
```
count = 0
marksdict = {}
namelist = []
```

In [39]:
```
(count,marksdict,namelist) = (0,{},[])
```

Exchange the values of two variables

- Can we write a function `swap(a,b)` that exchanges the values of its (immutable) arguments?
- Can only do something like `a,b = swap(a,b)`

In [40]:
```
def myswap(x,y):
    (y,x) = (x,y)
```

In [41]:
```
m = 5
```

In [42]:
```
n = 10
```

In [43]:
```
myswap(m,n)
```

In [44]:
```
m,n
```

Out[44]:
```
(5, 10)
```

In [45]:
```
def swap(x,y):
    return((y,x))
```

In [46]:
```
swap(m,n)
```

Out[46]:
```
(10, 5)
```

In [47]:
```
(m,n) = swap(m,n)
```

In [48]:
```
m,n
```

Out[48]:
```
(10, 5)
```

- To swap `x` and `y`, normally we need an intermediate temporary value `tmp`

```
tmp = y
y = x
x = tmp
```

- In Python, tuple assignment works!

```
(x,y) = (y,x)
```

- Parentheses are optional!

In [49]:
```
m,n = 5,10
m,n = n,m
```

In [50]:
```
m,n
```

Out[50]:
```
(10, 5)
```

- Can also use tuples to implicitly decompose pairs, triples

In [51]:
```
runlist = [("A",10),("C",20),("A",44),("B",33),("B",77)]
```

In [52]:
```
totalrunlist = {}
for (name,runs) in runlist:  # Use tuple to implicitly decompose
    if name in totalrunlist:
        totalrunlist[name] = totalrunlist[name] + [runs]  # or totalrunlist[name].append(runs)
    else:
        totalrunlist[name] = [runs]
```

In [53]:
```
totalrunlist
```

Out[53]:
```
{'A': [10, 44], 'C': [20], 'B': [33, 77]}
```

Mutable and immutable values

What is the value of `y` after the following code fragment?

In [54]:
```
y = 17
x = y
x = 15
```

In [55]:
```
y
```

Out[55]:
```
17
```

What are the values of `l1` and `l2` after the following code fragment?

In [56]:
```
l1 = [1,2,3]
l2 = l1
l2[0] = 4
```

In [57]:
```
l1,l2
```

Out[57]:
```
([4, 2, 3], [4, 2, 3])
```

- When we assign `y = x`, the value is copied - *immutable value*
- When we assign `l2 = l1`, both names point to the same value - *mutable value*

- Lists are dictionaries are mutable
- All other values are immutable

In [58]:
```
marks
```

Out[58]:
```
{'Physics': 75, 'Maths': 88}
```

In [59]:
```
newmarks = marks
```

In [60]:
```
newmarks['Maths'] = 100
```

In [61]:
```
marks, newmarks
```

Out[61]:
```
({'Physics': 75, 'Maths': 100}, {'Physics': 75, 'Maths': 100})
```

- Tuples are immutable

In [62]:
```
p1 = (5,2,3)
p2 = p1
```

In [63]:
```
p1, p2
```

Out[63]:
```
((5, 2, 3), (5, 2, 3))
```

In [64]:
```
p2[1] = 7
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [64], line 1
----> 1 p2[1] = 7

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

In [65]:
```
p2 = (7,9,4)
```

In [66]:
```
p1, p2
```

Out[66]:
```
((5, 2, 3), (7, 9, 4))
```

How can we "safely" copy a list?

- Make a copy of `l1` in `l2` that does not point to the same value

- Any slice `l[i:j]` creates a new list
- Assign a full slice `l[:]`

In [67]:
```
l3 = l1[:]
```

In [68]:
```
l1,l3
```

Out[68]:
```
([4, 2, 3], [4, 2, 3])
```

In [69]:
```
l3[0] = 17
```

In [70]:
```
l1,l3
```

Out[70]:
```
([4, 2, 3], [17, 2, 3])
```

Pitfalls with mutability

- Multiple references to same list value

In [71]:
```
zeros = [0,0,0]
zeromat = [zeros,zeros,zeros]
```

In [72]:
```
zeros, zeromat
```

Out[72]:
```
([0, 0, 0], [[0, 0, 0], [0, 0, 0], [0, 0, 0]])
```

In [73]:
```
zeromat[1][1] = 1
```

In [74]:
```
zeros, zeromat
```

Out[74]:
```
([0, 1, 0], [[0, 1, 0], [0, 1, 0], [0, 1, 0]])
```

- One solution is to use a full slice when reusing a list

In [75]:
```
zeros = [0,0,0]
zeromat2 = [zeros[:],zeros[:],zeros[:]]
```

In [76]:
```
zeros, zeromat2
```

Out[76]:
```
([0, 0, 0], [[0, 0, 0], [0, 0, 0], [0, 0, 0]])
```

In [77]:
```
zeromat2[1][1] = 1
```

In [78]:
```
zeros, zeromat2
```

Out[78]:
```
([0, 0, 0], [[0, 0, 0], [0, 1, 0], [0, 0, 0]])
```