More Python 3

Data and File Structures Laboratory

`http://www.isical.ac.in/~dfslab/2019/index.html`

Corrigenda

- Bitwise operators:
- Functions: `return` statement not mandatory for functions that do not return a value

Input/output I

Opening/closing a file

```
fp = open("file.txt", "w") # r, w, a
fp.close()
```

Reading / writing

```
input = f.read(size) # read size bytes, or entire file if size is
    omitted
inputline = f.readline() # read + return the next line
inputlines = f.readlines(n) # read + return next n (or all, if n
    omitted) lines as list

f.write(string)
f.writelines(l) # l is a list of strings
                # (include \n if you need them!)
```

No `f.writeline()` !

Input/output II

Printing to file / I/O redirection

```
print('test', file=f) # f -- file opened in write mode
# For global output redirection #
import sys
sys.stdout = open('file', 'w')
print('test')
```

Line by line handling

```
with open('filename.txt') as fp: # no need of fp.close() !
    for line in fp:
        do_something(line)
```

Other functions

```
f.flush()
f.tell()
f.seek()
```

Simple command line arguments

```
import sys
print("argc = ", len(sys.argv))  # NOTE: multiple arguments allowed
for i, a in enumerate(sys.argv) :
    print(str(i) + "-th argument:", a) # NOTE: use of str() and +
```

For more advanced command line argument processing, see the `argparse` library

Strings

```
s = 'This is an example string, with commas, and no other punctuation'

# Breaking a string into pieces
s.split() # splits on whitespace by default
s.split(sep=",") # splits on commas

# Searching for a substring # CAREFUL: returns -1 if not found
s.find("is")
s.find("is", 10) # starting position = 10
s.find("is", 0, 3) # start = 0, end = 3

s.replace(old, new) # new string is returned; s does not change

# Removing leading / trailing x (optional, whitespace by default)
s.strip(x)

# Padding / alignment / justification: pad s with blanks so that it
# appears left/right/center justified within a string of length n
s.ljust(n), s.rjust(n), s.center(n)
s.zfill(n) # as above, but pad with leading zeros
```

Creating formatted strings

```
'{} {}'.format('Simple', 'example')

'More {1} {0}'.format('example', 'complex')

'{0:2d}|{1:3d}|{2:4d}|{1:.2f}'.format(x, x*x, x*x*x)

'{lang} is {adjective}.'.format(lang='Python', adjective='easy')
```

What about `scanf()`? See `https://pypi.org/project/scanf/`

Regular expressions

```
import re

re.search(pattern, string[, flags]) # match anywhere

re.match(pattern, string[, flags])  # match at beginning
```

Returns None if not found

```
re.sub(pattern, repl, string[, count, flags])

re.split(pattern, string[, maxsplit=0, flags=0])
```

List comprehension, generators I

General format

- List comprehension:

```
[ expr for x in xs if cond1
  for y in ys if cond2
  ...
  for z in zs if condN ]
```

- Generators:

```
( expr for x in xs if cond1
  for y in ys if cond2
  ...
  for z in zs if condN )
```

List comprehension, generators II

Examples

```
squares = [ x**2 for x in range(1, 11) ]
[ x + y for x in range(3) for y in range(5,7) ]

names = ['dipen', 'sudipta', 'arnab', 'subhadip', 'rishi', ... ]
upper_names = [ name.upper() for name in names ]
```

List comprehension, generators III

Generators vs. comprehension

Generators take much less memory (since they generate elements on demand)

```
list_comp = [ x ** 2 for x in range(10) if x % 2 == 0 ]

gen_exp = ( x ** 2 for x in range(10) if x % 2 == 0 )


In [64]: print(list_comp)
[0, 4, 16, 36, 64]

In [65]: print(gen_exp)
<generator object <genexpr> at 0x7fe0d849e3b8>
```

filter, map I

- `filter(f, l)`: returns an iterator yielding those items of iterable `l` for which function `f` is true.

  [If function is None, return the items that are true.]

- `map(f, l1, l2, ...)`: returns an iterator that yields `f(l1[0], l2[0], ...)`, `f(l1[1], l2[1], ...)`, `f(l1[2], l2[2], ...)`; stops when the shortest iterable is exhausted.

```
import math

def is_prime(n) :
    if n == 0 or n == 1 :
        return False
    for i in range(2, 1+int(math.sqrt(n))) :
        if n % i == 0 :
            return False
    return True
```

filter, map II

```
A = list(range(10))
B = list(range(10,20))

P = filter(is_prime, A)
T = filter( lambda x : x % 3 == 0, A)
print(list(P), list(T))

M = map(lambda x,y : x+y, A, B)
print(list(M))
M = map(lambda x,y : x+y, A, B[:3])
print(list(M))
```

Random useful modules

itertools https://docs.python.org/3/library/itertools.html

```
import itertools
# OR import itertools as it
# OR from itertools import *


# return r length subsequences of elements from the input iterable
# combinations(list, 2) is particularly useful
itertools.combinations(iterable, r)
itertools.permutations(iterable, r)
itertools.combinations_with_replacement('ABCD', 2)


# cartesian product of input iterables
itertools.product(A, B, C)     # A x B x C
itertools.product(A, repeat=2) # A x A x A
```