Discrete structure using Python

Credits to:

Ali Farahani, National University, San Diego

Why python is used?

- intuitive syntax
- easy to learn
- close similarity with mathematics notation
- close similarity with algorithmic psudocode
- has interactive window for code testing and simple calculations
- has rich standard library and many other modules
- used widely by both novice and expert developers

Set operations Example 1

- For example, the set of the first twenty even numbers

- set builder notation is denoted by

  $S = \{x \mid x = 2n; 0 \leq n \leq 19\}$

- In python:

  *S = [2*x for x in range(19)]*

Set operations
Example 2

- Make a function in python to check whether N (input from user) is a prime number
- Simply applies the definition of a prime number to N by looping through a set of integers less that N to see is N has a factor there.

```
def func():
   N = int(input("Input a number you want to check whether it is prime or not."))
   for i in [x+1 for x in range(N)]:
      if N % i==0 and (i!=1 and i!=N):
         return False
      return True
print (func())
```

Set operations
Example 3

- In Python set membership can be examined by the command “in”.
- Here is an example of set equality; suppose we want to verify that the sets A and B given below are equal

  $A = \{x \mid x^2 + x - 6 = 0\}$ *and* $B = \{2,-3\}$ *then* $A = B$

- In python:

  *A=set ([x for x in range (-50,50) if x**2+x-6==0])*

where the *set* command is applied to convert the *list* to a *set* object. The set is constructed by searching for integer solutions of the quadratic equation in a specified range.

Set B is simple to construct in Python, B*=set ([2,-3]).* Now we use the command *A==B* to verify the equality. The system returns “True”.

product of sets

A set product, or a set of ordered pair is easily built by a single line of code assuming that sets A and B have been defined then *[(a,b) for a in A for b in B]* produces the product of the two sets.

The *set* union, intersection and difference operations are all available in Python thus enabling one to verify Demorgan's law for sets.

Logic
Example 1

- Python supports conjunction and disjunction by *and* and *or,* the negation is done using *not.*
- A nested loop can be constructed to generate the truth table for standard logic operators such as the *and*

```
for p in (True, False):
    for q in (True, False):
        print "%10s %10s %10s" % ( p, q, p and q)
```

Logic
Example 2

- expression – $p \wedge (q \vee r)$

```
for p in (True, False):
        for q in (True, False):
                for r in (True, False):
                        print "%10s %10s %10s %10s" % ( p, q, r, not p and (q or r))
```

Logic
Example 3

- The implication statement $p \rightarrow q$ can either be looked at as equivalent to $-p \vee q$ or encoded as a function

```
def implies(p,q):
    if a:
        return b
    else:
        return True
```

- To express the logical statement $x \geq 0$ and $y \geq 0 \rightarrow x * y \geq 0$, we can write

```
implies ( x >= 0 and y >= 0 , x * y >= 0 )
```

Quantifiers Example 1

- A propositional or a predicate function $f(d)$ is a Boolean function whose truth value depends on the value of d which is restricted to a properly defined domain.
- Most of the statements in mathematics and computer science use terms such as "for every (∀)" or "there exists(∃)"
- Universal: $\forall x \in Z(x^2 \geq 0)$ in python

```
for x in range(-100,100):
    if not (x**2 >=0):
        return False
return True
```

The code only test integer numbers between -100 and 100; if any of the values assigned to the
loop variable fails the stated proposition i.e. $x^2 \geq 0$, then the for loop is terminated with a "False" output, otherwise the for loop will iterated exhaustively and outputs True

Quantifiers Example 2

- Existential: $\exists x(x\text{-}2 = 0)$
- In python:

```
for x in range(-100,100)]:
    if x - 2 == 0:
        return True
    return False
```

Quantifiers Example 3

- nested quantifiers
- $\forall x$ ∃y (x − y = 0) in python

```
for x in range(-100,100)]:
    for y in range(-100,100)]:
        if i - j == 0:
            return True
return False
```

Combinatorial selection Example 1:

- The combinatorial problems can easily be formulated in terms of loops
- An example to print all possible ways to select 3 out of the N first non-negative integers if order is relevant and repetition is allowed.

```
for i1 in range(0,3):
    for i2 in range(0,3):
        for i3 in range (0, 3):
            print i1,i2,i3
```

- By simple adjustments of the range in the inner loops one can generate a selection without repetition or a larger selection can be obtained with an appropriate number of nested loops.