Appendix A.

Python

A.1. Basic Properties

1. Python is a dynamically-typed language. We do not need to declare types of variables, and their values can be of any type.

```python
a = 15
a = "Hello, Python"
```

2. Python is a strongly-typed language. Python requires an *explicit* conversion when we work with multiple types.

```python
a = 15
b = " files"
c = a + b      # Incorrect
c = a + str(b) # Correct
```

A.2. Data Structures

1. List is a basic data structure in Python to manage a ordered collection of data items.

```python
a = []           # "a" is an empty list of three elements
b = [1, 2, 3]    # "b" is a list of three elements: 1, 2, 3
c = 5            # "c" is an integer
print(b[0])      # display the first element of the list "b"
a.append(c)      # append "c" to the list "a"
print(a)         # display the list i.e. [5]
d = a + b        # "d" is the concatenation of "a" and "b"
e = b.pop()      # remove the last element of "b"
print(b)         # display [1, 2, 3]
```

2. Set is similar to list but it is for the unordered collection.

```
f = set()          # "f" is an empty set
g = {1,2}          # "g" is a set of two elements
h = 2
print(h in g)      # display "True"
print(h not in f) # display "True"
```

3. Deque (double-ended queue) is a structure provided in the `collections`. It allows values to be efficiently appended and removed at the both ends.

```
from collections import deque
q = deque()
q.append(2)      # append 2 to the end of "q"
q.appendleft(5) # append 5 to the beginning of "q"
print(q)         # display "deque([5,2])"
t = q.popleft() # remove the first element of "q"
u = q.pop()      # remove the last element of "q"
print(t, u)      # display "(5, 2)"
```

A.3. Flow Control Statements

Similar to many high-level languages, Python provides flow control statements e.g. `if`, `for`, `while`, etc. Python uses "indentation" to group statements

```
a = 5
b = 6
if a == b:
  print("Equal")
  print("=====")
elif a+1 == b:
  print("+1")
else:
  print("Not Equal")

for i in range(10):
  print(i)         # display 0 1 2 .. 9

l = [5, 4, 3, 1, 2]
for e in l:
  print(e)         # display each element of the list
```

A.4. Functions

Python uses a keyword **def** to define a function.

```
def f(x, y):
  print(x, end=",")
  print(y)
  return (x+y)/2

t = f(5, 3)         # display "5,3"
print(t)            # display "4"
```

A.5. Classes and Objects

Similar to object-oriented programming languages, Python allows us to define a class to encapsulate related values and operations. We can then create objects from the class.

```
""" point.py """
import math
class Point:
    # Define a constructor called when we create an object
    # of this class
    def __init__(self, x, y):
        self.x = x
        self.y = y
        # Use "self." to define and access each member of object

    # Define how to create a string for displaying the values of
    # this object
    def __str__(self):
        return "(%d, %d)" % (self.x, self.y)

    # Define how to convert this object into a string
    # This is useful for printing a list containing this object
    def __repr__(self):
        return str(self)

    def distance(self, p):
        return math.sqrt((self.x - p.x)**2 + (self.y - p.y)**2)

    def __eq__(self, p):
        return isinstance(p, self.__class__) and \
                (self.x == p.x) and (self.y == p.y)
    def __hash__(self):
        return hash((self.x, self.y))

if __name__ == '__main__':
    # The following statements won't be called when this file
    # is imported by the other python file.
    p1 = Point(0, 2)            # create an object of Point
    print(p1)                   # display "(0, 2)"
    l = []
    for i in range(10):
        p = Point(i, i+1)       # create an object of Point
        l.append(p)             # add the object into the list
    for i, p in enumerate(l):
        # "p" is a member of list "l"
        # "i" is the order of "p" in the list
        print(i, p1.distance(p))
```

A.5.1. Yield Statement

The **yield** statement is similar to **return**, but it causes the function to return a *generator* and **yield** returns a value for the generator.

```
""" gen.py """
from point import Point

def mygenerator():
    for i in range(5):
        yield Point(i, i)

def mylist():
    l = []
    for i in range(5):
        l.append(Point(i, i))
    return l

for j in mygenerator():
    print(j, end=" ")
print()
for j in mylist():
    print(j, end=" ")
print()
```