CMSC 245 – Principles of Programming Languages
Lab#2: Python Basics

Structure of a Python Program

The general structure of a Python program is shown below:

```
1  <imports>
2
3  <function definitions>
4
5  def main():
6     <code for main() goes here>
7
8 main()
```

Here are some salient features:

- **Line 1:** All imports go here (for example, math, random, etc).
- **Line 3:** All functions are defined here.
- **Lines 5-6:** Defines the `main()` function. Technically, Python does not require a `main()` function. This is just to transfer you over coming from languages like Java, C, etc.
- **Line 8:** Where the execution of your program starts is dictated by this function call.

Types and Operators

Python provides several built-in basic data types: numbers (integers, floating-point numbers), Booleans, strings, and collections (list, tuple, set, dict):

- **Numbers-integers:** The built-in type **`int`** represents integers that are represented in 32-bit 2's complement. **`int`** is the type you will use for most common applications. The standard operators defined on integers are: **+**, **-**, *****, **/**, and **%**. The operator ** is the exponentiation operator. It is a binary operator, written as x ** y (i.e. it computes x^y).
- **Numbers-floating point numbers:** The built-in type **`float`** is used for floating point values. The standard operators defined on floating-point numbers are: **+**, **-**, *****, **/, ****.
- **Booleans:** The type **`bool`** is used to define Boolean values (**`False`**, **`True`**). The three standard logical operators available are: **and** (and), **`or`** (or), and **`not`** (not). Besides these, the comparison operators (**==**, **!=**, **>**, **>=**, **<**, **<=**) also yield Boolean values. Python also has an **`in`** operator that tests for membership. For example,

  ```
  evens = [2,4,6,8,10]
  2 in evens        -> returns True
  3 in evens        -> returns False
  ```

- **Strings:** The type `str` is used to define strings. String literals are written enclosed in singe, double, or triple quotes: `‘Bryn Mawr’`, `"Bryn Mawr"`, `‘‘‘Bryn Mawr’’’`, `"""Bryn Mawr"""`, etc. Triple quotes are generally reserved for multi-line quotes. The operations on string type include: **+** (concatenation), **`s[i]`** (i-the character of a string **`s`**), and **`len(s)`** for finding out the length of a string **`s`**. Strings in Python can also be compared using **<**, **<=**, **>**, **>=**, **==**, and **!=**.

- **Collections:** These represent a collection of values and can be a **`list`**, **`tuple`**, **`set`**, or **`dict`** (dictionary).

  ```
  colleges = [("Bryn Mawr", "Haverford", "Swarthmore"]   # list
  data = [12, 54, 367, 89, -1, 23]                       # list
  ```

  Lists can be indexed (`colleges[0], data[3]`), and are changeable (colleges[1] = `“Villanova”, data[2] = 99`).

  ```
  person = ("Deepak", "Kumar", 2, "CMSC245", "CMSC325")  # tuple
  ```

  Like lists, tuples can be indexed (`person[2]`).

  ```
  primes = {2, 3, 5, 7, 9}                          # set of int
  ```

  Sets cannot be indexed, are unordered, and are unchangeable. Sets do not contain any duplicates. Lists, tuples, and sets can contain a mix of values of any type.

  ```
  # Dictionary
  namesDictionary = {1: "uno", 2: "dos", 3: "tres", 4: "quatro"}
  ```

  Dictionary contains a bunch of **`key:value`** pairs. A dictionary can be indexed using a key (`namesDictionary[3] = "tres"`). Keys and values can be of any type.

Variables and Declarations

Variable names: All variable names must start with a letter, including _ (underscore), and may contain letters, numbers, or _ (underscore).

Scope: Variables can be used anywhere in the program without first declaring them. Python uses lexical scoping. A variable is visible after it is defined, in the block that it is defined, and available in any enclosing blocks. In Python, a variable is a container of value(s) of any type. That it, it could be assigned values of different types in different places in a program (though this is not a good practice!).

Some unusual ways variables can be used:

1. **Assigning multiple variables at once**

   ```
   i, j = 23, 42
   f, s = 3.1415, "Bryn Mawr"
   ```

2. **Swapping two variables**

   ```
   i, j = j, i
   ```

   Exchanges the values of `i` and `j`.

3. **One can assign multiple variables from a list or a tuple (see below for lists, tuples)**

   ```
   a, b, c = [31, 23, 26]
   w, x, y, z = ("Bryn Mawr", "19010", 32, 89)
   ```

4. **Type of a variable**: In Python you can query the type of a vriable.

   ```
   type(w)         # str
   type(z)         # int
   ```

Control Structures

- **Assignment: `<var> = <expression>`**

  ```
  area = width * height
  ```

- **Multiple Assignment:**
  `<var1>, <Var2>, …, <varN> = <expr1>, <expr2>, …, <exprN>`
  `<var1>, <Var2>, …, <varN> := <expr1>, <expr2>, …, <exprN>`

  ```
  n, m := 10, 20
  ```

  `n` and **`m`** are initialized to 10 and 20, respectively.

  Multiple assignment can also be used to swap two variables:

  ```
  n, m = n, n
  ```

  The above swaps the contents of **`n`** and **`m`**.

- **Conditional Statements- If**
  There are several forms of if-statements in Python:

```
if <condition>:                 if <condition>:
     <statement(s)>                       <statement(s)>
                                else:
                                     <statement(s)>

max = b                         if a > b
if a > b:                            max = a
     max = a                   else:
                                     max = b
```

  In Python if- statements, you do not have to surround the `<condition>` with parentheses `()`, but you can. However, a colon (`:`) is used to indicate the start of a block. Indentation below that dictates what is/isn't in the block. Additionally, for nested if statements, we use a special keyword (**`elif`**):

```
if <condition>:
     <statement(s)>
elif <condition>:
     <statement(s)
elif <condition>:
     <statement(s)
…
else:
     <statement(s)>
```

- **Conditional Statements- match/case**

```
match <expression>:
     case <expression1>:
          <statement(s)>
     case <expression2>:
          <statement(s)>
     …
     case _: <statement(s)># default case
```

```
match month:
     case 1 | 3 | 5 | 7 | 8 | 10 | 12:
         daysInMonth = 12
     case 4 | 6 | 9 | 11:
         daysInMonth = 30
     case 2:
       if (leapYear(year):
            daysInMonth = 29
       else:
            daysInMonth = 28
     case _: print(f"Illegal value for month: {month}")
```

The **`<expression>`** in switch is optional. And, cases can refer to any variable in the current scope (see below). Case expressions can be any value, or an expression (see below). There is no break statement required (as in C, C++, Java, etc.). Only the case selected is executed.

```
import random
r = random.random()   // generates a random float in [0.0..1.0)
match r:
     case r if r < 0.5:
          outcome = 'HEADS'
     case r if r >= 0.5:
          outcome = 'TAILS'
```

- **Loops- for**
  The general form of a for-statement is:
  ```
  for <var> in <sequence object>:
       <statements>
  ```

  `<var>` is a loop-control variable. `<sequence object>` (technically called **iterable object**). A sequence object could be a list (like `[1,2,3,4]`, or a string (like `"Bryn Mawr"`), or any other sequence object (usually user-defined). The built-in **`range()`** function (technically a sequence object!) has many forms. In the simplest case it returns a sequence of numbers in [0..n).

  ```
  for i in range(n):
       sum += i
  ```

- **Loops- while**

  The while-loop has the following syntax:

  ```
  while <condition>:
       <statement(s)>
  ```

  For example:

  ```
  while a != b:
       if a > b:
            a = a – b
       else:
            b – b – a
  ```

  All loops can use the **`continue`** and **`break`** statements to skip to the next iteration of the loop, or to exit the loop, respectively.

Exercises

1. The Notorious FizzBuzz Problem: Write a program that prints numbers from 1 to 100. However, for numbers that are multiples of three it prints the word “Fizz” instead of the number, for multiples of five, it prints “Buzz”. And, for numbers that are multiples of both three and five, it prints “FizzBuzz”.

2. Write two programs that simulate the following:

- rolls one six-sided die 100 million times and computes the empirical probability of getting a 6.
- rolls of two six-sided dice 100 million times and computes the empirical probability of getting a sum of 7.

When completed:

Send an e-mail to your instructor indicating you have completed ALL parts of the lab.