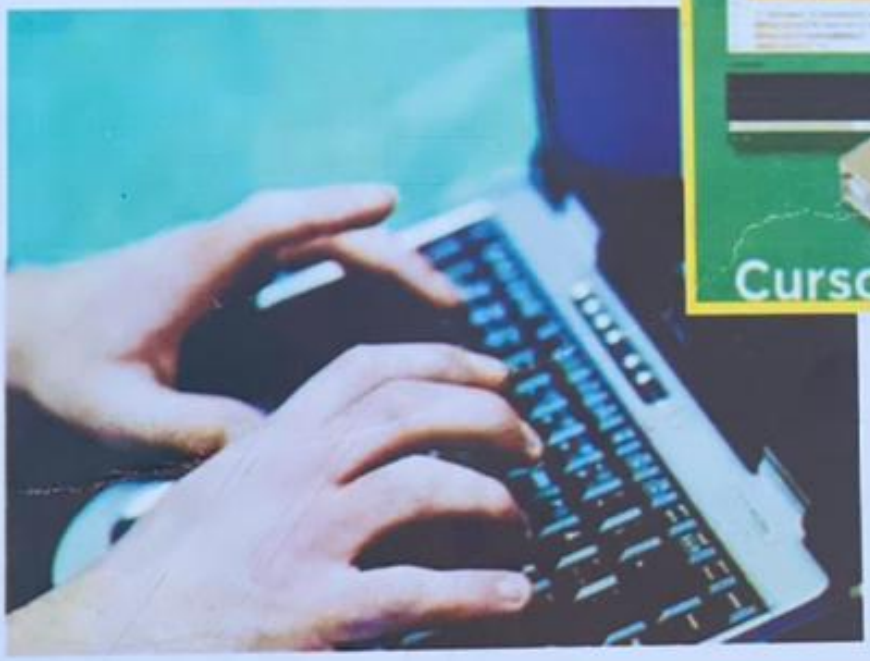
STRUCTURED PROGRAMMING

(CSC 301)

TUTORIAL WORKBOOK



DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF PHYSICAL SCIENCES
FEDERAL UNIVERSITY OF AGRICULTURE, ABEOKUTA

Lecture One: Introduction to Structured Programming

Structured programming is a programming approach where a program is modularized for the purpose of clarity, efficiency and modification. This is mostly achieved in structured programming through the use of program flow constructs such as selection statements; if ….statements, for…statements, do….while, and block structures,. Structured programming approach is one where a problem is solved by dividing the problem into modules and solutions proffered to those modules, so that the solutions could be in turn combined to serve as the solution to the overall problem. Examples of Structured programming languages include C Language, Pascal, ALGOL, PL1, ADA

Features of Structured programming approach

- Breaking of problems into smaller modules or procedures or functions.
- Absence of GOTO statements
- Structured programs are usually composed of If , For, Do While statements
- Availability of expanded set of operators like bit manipulation, shift and part word operators in addition to logical, relational, arithmetic and comparison operators.

Exercise 1.1 List 10 examples of Structured programming Languages:

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

Lecture Two: Introduction to Python programming

Python a programing language created by Guido van Rossum is a structured programming language. It has a simple syntax, hence, it is extremely easy to get started. It is an example of a Free/Libré and Open Source Software (*FLOSS)*. In simple terms, you can freely distribute copies of this software, read its source code, make changes to it, and use pieces of it in new free programs. Python is an interpreted language where there is no need to compile python source code to object but rather to byte code.

This means that when python programs are written, programmer does not have to bother about the low-level details such as managing the memory used by his program. Furthermore, python is a ported programming language. This shows that python programs can work on any platform without change to the program. Python also allows part of code written in C/C++ to be used in python. Programmers can embed python code in C/C++ as a script language. In python, there is huge standard library to make development of cryptography, databases, GUI (Graphics User Interface), and HTML files possible.

Exercise 2.1: Give a brief description of Python programming language

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

Lecture Three: Structure of a Python programs

A typical Python program structure is made up of statements for invoking library files, function/modules definition, comments blocks of statements. The idea of whitespace and indentation is also very important in the structure of a python program. Use of whitespace at the beginning of the line is called indentation and very important in python programming structure. Leading whitespace at the beginning of the logical line is used to determine the indentation level of the logical line, which in turn is used to determine the grouping of statements. In python, statements in the same block must have the same indentation. Indentation that is not right would give rise to errors in a program.

Example 3.1: A program to illustrate the wrong use of whitespaces and indentation.

```
x = 10
# Program to illustrate indentation error
 print('What is your  name’)
print('Asking my name is an ERROR’)
```

Output:

Error message:

```
  File "wrongspace.py", line 3
    print('What is your name’)
    ^
IndentationError: unexpected indent
```

Exercise 3.1: What number of whitespaces is officially recommended by python for indentation. Illustrate your answer with one example.

Lecture Four: Writing Python programs

Python programs could be written and tested using the interactive interpreter prompt or source file in code editor. The standard procedure however, is to use a code editor (like Pycham or Vim) to create the program as a file to be stored for later use. Integrated Development Environment (IDE) such as Jupyter Notebook could also be used.

Programming with Python Interpreter Prompt

- Open your operating system prompt (i.e. command prompt in Windows)
- Navigate to the python working directory
- Type "python" and press Enter key to see the prompt character like >>>
- Type command statement such as print("Good to see my first python Code") and press enter key

Programming in Python with Code Editor

The procedure to save and run a Python program is as follows:

1. Open the installed text editor i.e PyCharm
2. Click on Create new Project
3. Select Pure Python and Change untitled to helloworld (in front of for the location of the project).
4. Click the Create button.
5. Right-click on the helloworld in the sidebar and select New -> Python File
6. When prompted to type the name, type hello
7. After an empty file is opened, type code statement i.e print("Welcome to my first code")
8. Right-click on the code statement and click on Run 'hello'

Exercise 4.1: Describe the functionality of any other code editor (apart from PyCharm) for python programs.

Exercise 4.2: Compare and contrast between Pycharm and Vim code editors.

Lecture Five: Input and Output in Python

One common way to write and test programs is for user to interact with the system by supplying input values to get output from the program. This can be achieved using functions such as: input ( ) and print( ) functions, str (string) e.t.c. Another method of dealing with input/output is through files. In programming, files must be created, read, and written (using create read, write).

Example 5.1: Program to check whether a word is a palindrome.

```
def reverse(word):
    return word[::-1]


def  palindrome(word):
    return word == reverse(word)


something = input("Type your word: ")
if palindrome(something):
   print("Yes, word entered is a palindrome")
else:
   print("No, word entered is not a palindrome")
```

When the code statement is executed with different input values the outputs would look like the following:

Output:
Type your word: sir
No, word entered is not a palindrome

Type your word : madam
Yes, word entered is a palindrome

Type your word: racecar
Yes, word entered is a palindrome

Exercise 5.1: Write a Python program that would allow user to enter name, firstscore, secondscore, thirdscore, totalscore, and averagescore using input ( ) and print ( ).

Exercise 5.2: Write a Python program to read a file line by line and store it into a variable.

Exercise 5.3: Write a Python program to read a file line by line and store it into a list

Exercise 5.4: Write a Python program to read last n lines of a file.

Exercise 5.5: Write a Python program to read an entire text file

Lecture Six: Literal constant, String, Variables and Identifiers

Literal constant is a character or collection of characters that do not change in a program. Examples of literal constants are 7, 10.5. When literal constant is in form of alphabet, they are called string literals. Examples of string literals are 'Computer Science' and "Federal University of Agriculture, Abeokuta".

Variables are data names or identifiers that hold different values in the life of a program. In python, some rules are used to guide the formulation of variables (or identifiers). These include:

- ✓ An identifier or variable must start with a letter of the alphabet (uppercase ASCII or lowercase ASCII or Unicode character) or an underscore (_).
- ✓ Identifiers must not be a python reserved word like print, str, e.t.c
- ✓ The rest of the identifier name can consist of letters (uppercase ASCII or lowercase ASCII or Unicode character), underscores (_) or digits (0-9).
- ✓ Identifier must include space or special character.
- ✓ Identifier names are case-sensitive. e.g. firstname and firstName are *not* the same.

Examples of *valid* identifier include j, average, total, my_salary, sum, abc. Examples of *invalid* identifier names are 7name, my salary, print, and >abc_c3.

Exercise 6.1: List 30 examples of reserved words in Python.

Lecture Seven: Data types in Python

Data types refer to the different forms of data that can be assigned to variables (or identifiers) in python. These include strings, numbers, and objects.

Exercise 7.1: Illustrate with program example the use of data types in Python

Exercise 7.2: Determine what would be the output of the following program snippet.

```
def string_length(str1):
    count = 0
    for char in str1:
        count += 1
    return count
print(string_length('www.unaab.edu.ng'))
```

Exercise 7.3: Write a Python program to get a string from a given string where all occurrences of its first char have been changed to '+', except the first char itself.

Lecture Eight: Operators in Python

Operators are used to define operation in a programming language. In most logical expression (or code statement), we use operators to define what operation(s) to be performed on the operands. Operators in python can be classified into:

(1) Arithmetic operators: + (plus), - (minus), * (multiply), ** (power)
(2) Relational/Comparison operators: (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), == (equal to), != (not equal to)
(3) Logical operators: not (boolean NOT), and (boolean AND), or (boolean OR)
(4) Assignment operators: =, +=, *=, /+, and -= Examples of its use are a = 2, a += 3, a *= 3, a /= 3, a -= 3
(5) Shift operators: << (left shift) and >> (right shift).

Exercise 8.1: Write a python program to perform arithmetic operations such as addition, subtraction, multiplication and division (+, -, *, /) on four numbers.

Exercise 8.2 Write a python program to generate the bitwise operator of X and Y, given that X=00111100 and Y= 00001101

Lecture Nine: Control Flow Statements in Python

There are situations when programs are written to be executed by obeying series of statements from beginning to the end. In the same vein, there are situations where problems can only be solved in programming by taking certain decision or repeating certain part of the program a number of times. These are usually carried out using control flow statements. There are three control flow statements in Python namely; if, for and while.

If Statement

The if statement is used to check a condition: *if* the condition is true, we run a block of statements (called the *if-block*), *else* we process another block of statements (called the *else-block*). The *else* clause is optional.

While Statement

The while statement allows you to repeatedly execute a block of statements as long as a condition is true. A while statement is an example of what is called a *looping* statement. A while statement can have an optional else clause.

For Loop

The for..in statement is another looping statement which *iterates* over a sequence of objects i.e. go through each item in a sequence. We will see more about sequences in detail in later chapters. What you need to know right now is that a sequence is just an ordered collection of items.

Break Statement

The break statement is used to *break* out of a loop statement i.e. stop the execution of a looping statement, even if the loop condition has not become False or the sequence of items has not been completely iterated over.

Continue Statement

The continue statement is used to tell Python to skip the rest of the statements in the current loop block and to *continue* to the next iteration of the loop.

Exercise 9.1: Using for…in statement write a Python program to generate the number of characters in the word "Computer Science Department"

Exercise 9.2: Using while ….. statement write a python program to generate integer values between 1 and 1000.

Exercise 9.3: Using break statement write a python program to display prime numbers between 2 and 10.

Exercise 9.4: Using if ….statements write a python program to determine whether an integer supplied by a user is greater than 1000 or not.

Exercise 9.5: Write a python program to illustrate continue statement

Lecture 10: Arrays in Python programming

An array is a collection of data values represented a variable name or identifier. In python a number of data items can be defined, represented, and referenced by an identifier.

Example 10.1: A Python program to create and display an array of 10 numbers.

```
array_num = array('i', [3,5,7,9,1,7,9,10,4,3])
for i in array_num:
print(i)
print("Displaying only five of the numbers")
print(array_num[0])
print(array_num[4])
print(array_num[3])
print(array_num[1])
print(array_num[9])
```

Output:

```
3
5
7
9
1
7
9
10
4
3
Displaying only five of the numbers
3
1
9
5
3
```

Exercise 10.1: Write a python program to create an array of 10 integers. Display the elements with index values of 2, 5, and 7 from the array.

Exercise 10.2: Write a python program to create and print an array of strings values.

Exercise 10.3: Write a python program to search for a given element in an array of INTEGER values.

Exercise 10.4: Given an array dept = ["CSC", "STS", "MTS", “CHM”, “PHS”] write a python program to sort the array “dept” ascending order.

Lecture Eleven: Functions in Python programming

A function is a block of code statements that is defined once but can be used many times. They are therefore referred to as reusable pieces of programs. It is usually defined using a name with which it can be referenced and called. Functions can be users-defined or built-in. Examples of some built-in functions in python are len and range.

Users-defined functions can be defined in python using the def keyword followed by the *identifier* name for the function and then by a pair of parentheses which may enclose some names of variables, and by the final colon that ends the line. Then this is followed by the block of statements that would constitute the function.

Mathematical Functions: Python supports a wide variety of mathematical functions. The followings are some of the examples of mathematical functions and their roles;

- ✓ abs(x) - Returns the absolute value of x.
- ✓ cmp(x,y) - Returns -1 if x < y returns 0 if x equals to y returns 1 if x > y.
- ✓ exp(x) - Returns the exponential of x. The usage is: import math x = 6 print(math.exp(x)) log(x) - Returns natural logarithm of x.
- ✓ log10(x) - The base 10 logarithm of x
- ✓ pow(x,y) - The result of x**y
- ✓ sqrt(x) - The square root of x

Example 11.1: A Python program to define a function

```
def  birth_greeting():
# block belonging to the function
  print('Happy Birthday to you !!!')
# End of function

   birth_greeting()  # call the function
```

Output:

Happy Birthday to you!!!

Exercise 11.1: Study the program below to determine the output for the python program

```
def max_of_two( x, y ):
    if x > y:
        return x
    return y
def max_of_three( x, y, z ):
    return max_of_two( x, max_of_two( y, z ) )
print(max_of_three(3, 6, -5))
```

Exercise 11.2: Study the program below to determine what the output would be.

```
def sum(numbers):
    total = 0
    for x in numbers:
        total += x
    return total
print(sum((8, 2, 3, 0, 7)))
```

Exercise 11.3: Write a python program to evaluate quadratic equation (use almighty formula)

Exercise 11.4: Study the python program below and determine what would be result of the program.

```
def string_test(s):
    d={"UPPER_CASE":0, "LOWER_CASE":0}
    for c in s:
        if c.isupper():
           d["UPPER_CASE"]+=1
        elif c.islower():
           d["LOWER_CASE"]+=1
        else:
           pass
    print ("Original String : ", s)
    print ("No. of Upper case characters : ", d["UPPER_CASE"])
    print ("No. of Lower case Characters : ", d["LOWER_CASE"])

string_test('The quick Brow Fox')
```

Exercise 11.5: Study the python program below and describe the result of the program.

```
def make_bold(fn):
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped

def make_italic(fn):
    def wrapped():
        return "<i>" + fn() + "</i>"
    return wrapped

def make_underline(fn):
    def wrapped():
        return "<u>" + fn() + "</u>"
    return wrapped
@make_bold
@make_italic
@make_underline
def hello():
    return "hello world"
print(hello()) ## returns "<b><i><u>hello world</u></i></b>"
```

Lecture Twelve: Data Structures in Python programming

Data structure is the programming concept that describes how collection of related data are stored and held together. In python, data structures include list, tuple, dictionary, and set.

List: A list is a data structure that holds an ordered collection of items. In a list, the items are usually enclosed in square brackets. Data items can be added, removed, or searched in a python list. An example of a list is fruit = ['apple', 'mango', 'carrot', 'banana'].
,

Tuple: This is another python data structure that is used to hold multiple objects together. Tuples are defined by specifying items separated by commas within an optional pair of parentheses. Tuples are usually used in cases where a statement or a user-defined function can safely assume that the collection of values (i.e. the tuple of values used) will not change.

Dictionary: A data structure that is used like an address-book where you can find the address or contact details of a person by his name, i.e. by associating *keys* (name) with *values* (details).

Set: Sets are unordered collections of simple objects. These are used when the existence of an object in a collection is more important than the order or how many times it occurs. Set data structure in python is used to test for membership, whether it is a subset of another set, find the intersection between two sets, and so on.

Exercise 12.1: Write a python program to store and display list of books on a shelf.

Exercise 12.2: Write a python program to determine the number of books in (i) Arts shelf (ii) science shelves (iii) all books in arts and science shelves.

Exercise 12.3: Write a Python program to store and display names and email addresses of lecturers in your department.

Exercise 12.4: Given two sets of objects as "Carnivores" and "Herbivores". Write a python program to test if Dog is a member of Carnivores or Herbivores.

Lecture Thirteen: Modules in Python programming

A module is a concept that makes the reuse of functions possible in python programs. One of the ways to write a module is to create a file with a .py extension that contains functions and variables. Another way is to write the modules in the native language in which the Python interpreter itself was written. An example of this is when modules is written using C language and compiled. Then they can be called in your Python code using the standard Python interpreter.

Example 13.1: A python program to create a module

```
def greet():
   print('Happy New month.')

__version__ = '2.0'
```

The module below is an example of a module where the first module is used or called.

```
import greetmodule

greetmodule.greet()
print('Version', greetmodule.__version__)
```

Output:

```
Hapy New month.
Version 2.0
```

Exercise 13.1: Study the program below and explain what its output would be.

```
#mod1
def change(a):
    b=[x*2 for x in a]
    print(b)
#mod2
def change(a):
    b=[x*x for x in a]
    print(b)
from mod1 import change
from mod2 import change
#main
s=[1,2,3]
change(s)
```

Exercise 13.2: Using concept of module write a python program to implement the drawing of a polygon.