Structured Programming with Python

Donaldson Tan
maths_and_programming@outlook.sg

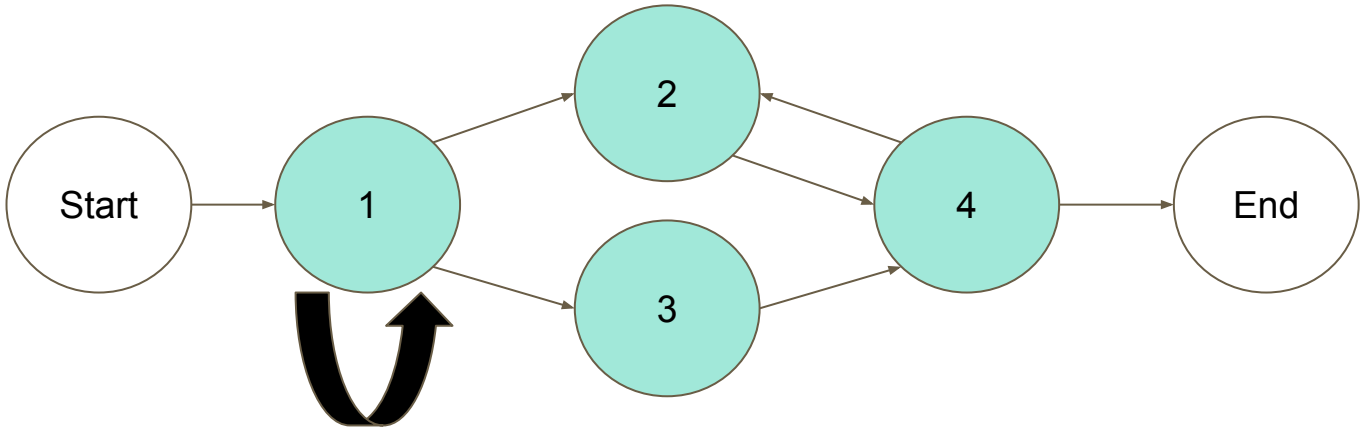
Lesson Plan 1:

Structured Programming with Python

1. What is a Computer Program?
2. Structured Programming Paradigm
3. Expression Construct
4. Subroutine Construct
5. Block Construct
6. Selection Construct
7. Repetition Construct
8. Python data types
9. From Natural Language to Python
10. Problem Solving
11. Exercise

What is a Computer Program?

- A computer program consists of data and a set of instructions.
- The data and instructions can be organised as a sequence of states.
- Control Flow determines the transition between 2 states.



Structured Programming

- Structured Programming is a programming paradigm
- It involves the following Control Flow constructs:
  - Block Constructs
  - Subroutine Constructs
  - Selection Constructs
  - Repetition Constructs

Expression Construct

- An expression
  - is a combination of operands and operators that yields a result.
  - is the basic building block of a programming language statement.
- Examples of Operand
  - Variable
  - Constant
- Examples of Operator
  - +, -, *, /, >, <, ==
- Example of Expression
  - `1 + 1 + 5`
  - `len(string1) > 5`
- We can chain expressions using operators and, or
  - `(x > 2) or (y > 2)`
  - `firstname == “John” and secondname == “Tan”`

Block Construct

- The Block Construct is a group of statements that perform as a single logical statement.
- A simple block of code shares the same level of indentation.
- Blocks may be nested.
- Python uses the LEGB Rule to resolve a variable name in a nested Block:
  - **L**ocal: The block where the variable name is invoked.
  - **E**nclosing: A local block may be enclosed by 1 or more external blocks.
  - **G**lobal: The top-level scope of the Python program
  - **B**uilt-in: A special scope that is created by the Python runtime system.
- A block cannot access variables created inside a nested block.

Subroutine Construct

- A Subroutine is a named block construct.
- Python's subroutine is called a function.
- A function is identified by its parametric signature.
  - `def func(p1: Type1, p2: Type2, … ) -> ReturnType`
- A function optionally accepts input and produces an output
- There are 3 types of input:
  - Parameter
  - External variable
  - I/O Stream [e.g. Keyboard input, mouse click]
- There are 3 types of output
  - A Python function always returns a value.
  - If user does not specify a return value, Python assigns `None` as default return value.
  - A function may write data to an I/O stream (e.g. print(), update visualisation)
  - A function may update a variable

Subroutine Construct

```
def findMax(numList: List[int]) -> int:
    m = numList[0]
    for n in numList:
        if n > m:
            m = n
    return m

def toPrint(s: str) -> None:
    print(s)

def update(numList: List[int], n: int) -> None:
    if len(numList) == 0:
        numList.append(n)
    else:
        numList[-1] = n
```

Selection Construct

- It is a type of block construct whereby the block only runs if the specified condition is met.
- This is implemented as if/then/else in Python
- How do I chain multiple conditions?

```
def update(numList: List[int], n: int) -> None:
    if len(numList) == 0:
        numList.append(n)
    else:
        numList[-1] = n
```

Selection Construct

- match/case is another selection construct in Python
  - Good to know, but not required for ICT133.

```
lang = input("What's the programming language you want to learn? ")
match lang:
    case "JavaScript":
        print("You can become a web developer.")
    case "Python":
        print("You can become a Data Scientist")
    case _:
        print("The language doesn't matter, what matters is solving problems.")
```

Repetition Construct

- Repetition construct repeats a block construct as long as the specified conditions are met.

```
for i in range(6):
    print(i)

i = 0
while i < 6:
    print(i)
    i += 1
```

Python Data Types

- How to declare a variable?
  - Assign a value to an unused variable name
  - Create a variable x of integer type and assign it the value of 23

    ```
    x = 23
    ```
- Python is dynamically typed.
  - Change variable x from integer to string

    ```
    x = “hello world”
    ```
- Native Data Type
  - Can be handled directly by the CPU
  - Natively represented on CPU registers.
  - Integer. E.g. 1, 2, 3
  - Float. E.g. 1.234, 3.14159 (Pi), 6.626E-34 (Planck’s Constant)
  - Boolean. E.g. True, False

Python Data Types

- Non-Native Data Type
  - String
    i. Each character of a string has an associated index number
    ii. To read a character, you invoke string1[index].
  - List
    i. Each element is associated with an index number.
    ii. To access a variable in a list, you invoke List[index]
    iii. Index numbers start with 0 and ends with len(List) - 1
    iv. An empty list is declared by `list = [ ]`
  - Dictionary
    i. Each element is associated with key string
    ii. To access a variable in a dictionary, you invoke Dict[key]
    iii. An empty dictionary is declared by `dict1 = { }`

Python Data Types

- Non-native data types
  - requires processing instruction that are not part of the CPU
  - the additional instructions are part of the computer program.
  - List, dictionary and string have built-in functions.
  - For example, List has built-in functions such as append, insert

```
Python 3.11.1 (v3.11.1:a7a450f84a, Dec  6 2022, 15:24:06) [Clang 13.0.0 (clang-1
300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = ["a", "b", "d"]
>>> a.insert(2, "c") #insert "c" at index 2
>>> print(a)
['a', 'b', 'c', 'd']
>>> a.append("e") # add "e" at the end of the list
>>> print(a)
['a', 'b', 'c', 'd', 'e']
>>>
```

Python Data Types

- Reading a list with for-loop

```
for i in numlist:
    print(i)

for i in range(5):
    print(numlist[i])

for i in range(0,len(numlist), 2):
    print(numlist[i])
```

Python Data Types

- Reading a dictionary with for-loop

```
for key in mydict:
    print(mydict[key])

for key in mydict.keys():
    print(mydict[key])

for key, value in mydict.items():
    print(key, value, mydict[key])
```

From Natural Language to Python

- Algorithm:
  - unambiguous description to solve a problem
  - Consists of input, output and procedure.
  - Input: a description of the input
  - Output: a description of the output
  - Procedure: a sequence of step to solve the problem
- How to write an algorithm in Python?
  - Step 1: First formulate the algorithm in Natural Language
  - Step 2: Attempt translate from Natural Language to Python
  - Step 3: If stuck
    - Step 3a: Refine the Natural Language until it is less ambiguous
    - Step 3b: Go back to Step 2
  - Step 4: Stop

From Natural Language to Python

- Unlike Natural Language, a programming language is
  - Unambiguous [Cannot mean for more than 1 thing]
  - Context Free [No need unspoken information to interpret its meaning]
  - Free of Fault [E.g. grammar must be correct]
- An algorithm formulated in Natural Language needs to be refined until the natural language description is:
  - Unambiguous
  - Context Free
  - Free of Fault

Problem Solving

- List down what you know
  - the input specification
  - the output specification
- Understanding the problem
  - Formulate an example input and infer the correct output
  - If you are given sample input and output, infer unspecified information not given in the problem
- Decomposition
  - Break down the problem into smaller problems which are easier to solve
- Pattern and Generalisation
  - Look for relationship in the data
  - Make use of structural pattern
  - Reverse engineering / Work backwards
- Modify the Problem
  - Simplify the problem
  - Solve a special case of the problem

Exercise 01 - Sorting Integers

- How to sort a list of integers?

Index	0	1	2	3	4	5
Element	10	14	15	17	22	25

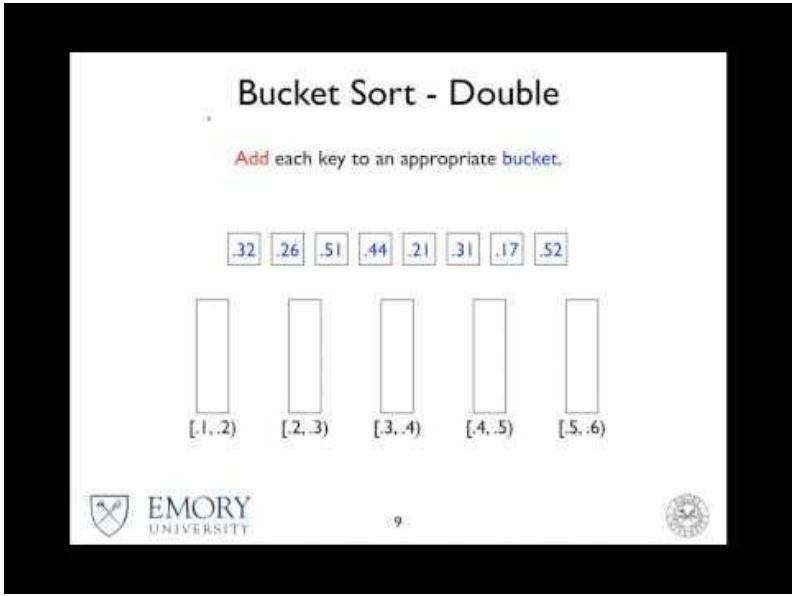
- **Hint**: Given the sorted list L (see above) in ascending order, observe:
  - 10 is the min element for L[0 to 5]
  - 14 is the min element for L[1 to 5]
  - 15 is the min element for L[2 to 5]
  - 17 is the min element for L[3 to 5]
  - 22 is the min element for L[4 to 5]
  - 25 is the min element for L[5 to 5]
- How to sort a list of integers in descending order?

Exercise 01

- Algorithm to sort in Ascending Order
  - Input: L[1…n] is a list of n integers
  - Output: List L sorted in ascending order
  - Procedure:
    - For i = 1 to n
      - Let j = index of smallest element in L[i to n]
      - Swap L[i] and L[j]

Exercise 02 - Detect Repeat Integers

- Given a list of 6 integers between 1 and 49 inclusive, verify there are no repeated integers
- Modify Bucket Sort Algorithm to solve this problem.
- First formulate Bucket Sort in Natural Language
- Then translate Bucket Sort to Python



Exercise 02

- Algorithm to verify 6 unique integers from 1-49
  - Input: A list of 6 integers
  - Output: True if the list has 6 unique integers, otherwise False
  - Procedure:
    - Create buckets 1 to 49 whereby each bucket is an empty list
    - For each integer n in the list
      - a. Add n to the list represented by bucket n
      - b. If the length of bucket n is greater than 1, return False
    - Return True