UNIT-I

Sl.No-1 Structured programming paradigm – Sl.No 2 programming Language Theory – Sl.No 3 Behavior Jacopini, Structured program theorem – Sl.No 4 Sequence, selection, decision, iteration, recursion, Sl.No 5 Other Languages: C, C++, Java, C#, Ruby. Sl.No 6 Demo: Structured programming in Python. Sl.No 7 procedural programming paradigm, Routines, Subroutines, functions, Using Functions in Python, logical view, Control flow of procedural programming in various aspects, Sl.No 8 Other languages: Bliss, Chuck, Matlab, Demo: Creating routines and subroutines using functions in Python Sl.No 9 Objected Oriented programming paradigm - class, objects, Instances, Methods, Encapsulation, OOP in Python.

Structured programming paradigm

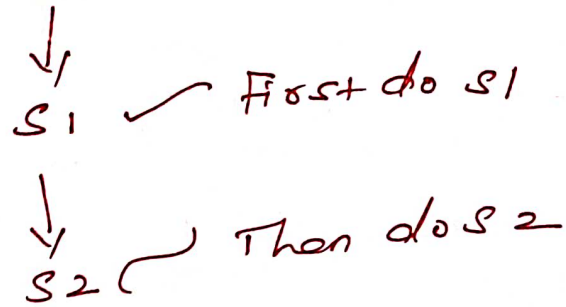
The types of programs written so far are for the Turing machine are what are usually called non-structured. This type of code is generally very difficult to read and understand.

. Bohm and Jacopini in 1966 showed that any non-structured program can be rewritten by combining three techniques: sequence, selection and repetition (or loop).

This is something we're familiar with and we've been writing our algorithms since we started, but how we can see the connection between the Turing m/c and what we've been doing.

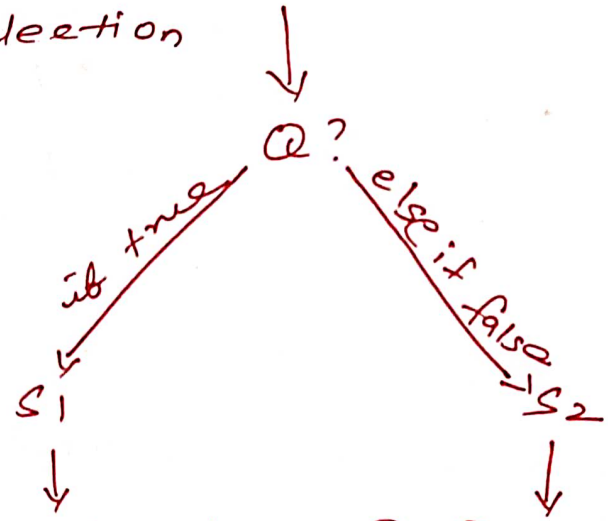
To series, Sequence performs some of
S, S2. meaning perform S1; then perform

Sequence



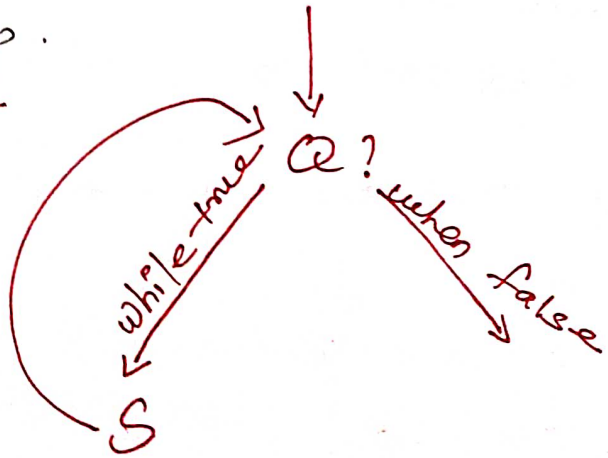
Selection says, if Q is true, then perform
else perform S2.

Seleetion



Loop says while Q is true, do S.

Loop.



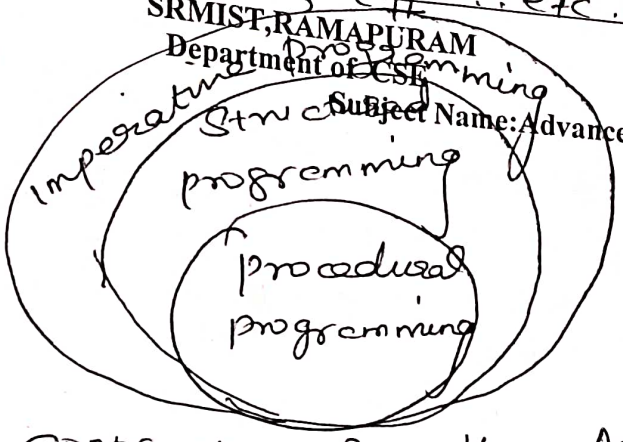
These are all the tools we need for
Structured programming.

Structured Programming Approach.

* C, C++, Java, C#, ..etc.

Course Code: 18CSC207J

SRMIST, RAMAPURAM
Department of CSE
Subject Name: Advanced Programming Practice



on the contrary, in the Assembly languages like Microprocessor 8085, etc, the stmts do not get executed in a structured manner. It allows GOTO. So pgm flow might be random.

Programming Paradigms.

Imperative programming paradigm

Declarative Programming Paradigm

- Procedural programming paradigm (C, C++, Java)
- OOP (C++, Java)
- parallel processing Approach.
- Logic programming (Prolog) Paradigm
- Functional Programming Perl, JS, Lisp
- DB processing Approach (SQL)

Python is considered as an object-oriented language rather than a procedural programming language.

It is identified by looking at python packages like Scikit-learn, Pandas and Numpy.

These are all Python packages limit with Object-Oriented programming.

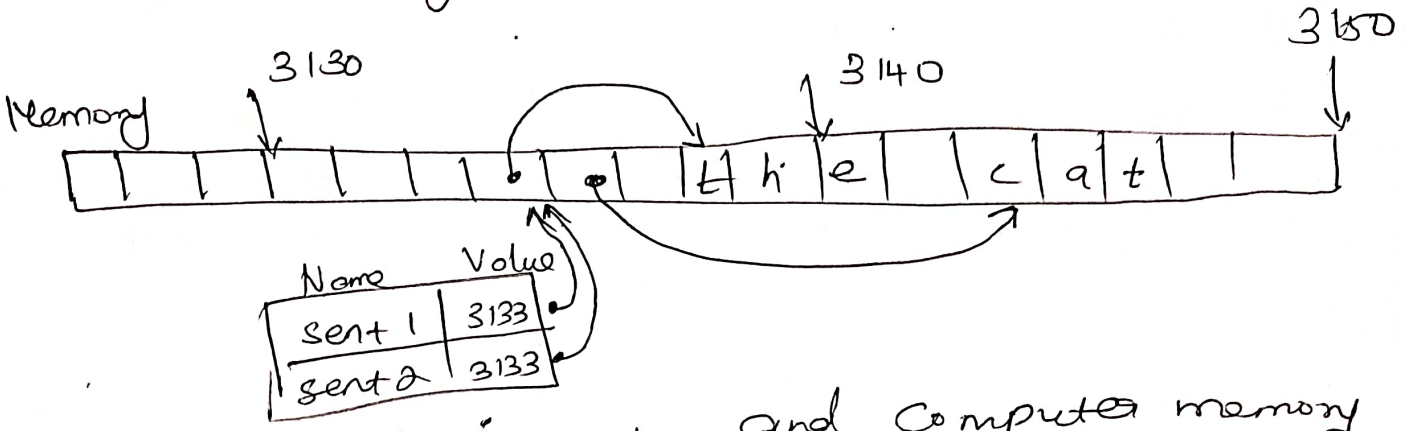
Structured programming (SP) is a technique devised to improve the reliability and clarity of programs. In SP, control of program flow is restricted to three structures, Sequence, IF THEN ELSE, And DO WHILE.

```
>>> Sent = ['I', 'am', 'the', 'walrus']
>>> i=0
>>> while i < len(sent):
...     print sent[i].lower(),
...     i=i+1
...     i+=1
i am the walrus.
```

Assignment

```
>>> word1 = 'Monty'
>>> word2 = word1
>>> word1 = 'Python'
>>> word2
'Monty'
```

```
>>> list 1 = ['Monty', 'Python']
>>> list2 = list1
>>> list1[1] = 'Bodkin'
>>> list2
['Monty', 'Bodkin']
```

List Assignment and computer memory

procedural programming Paradigm

A Function is a structuring element in programming languages to group a bunch of statements so they can be utilized in a pgm more than once.

* It enhances the Comprehensibility and quality of a pgm. ↓ the cost for devpt and maintanance of the s/w.

Functions are known under various names in programming languages, eg as subroutines, procedures, methods or subprograms.

Example of functions

Let's look at the following code:

```
print ("program start")
print ("Hi peter")
print ("Nice to see you again!")
print ("Enjoy our video!")

# Some lines of codes.
the_answer = 42.
print ("Hi Sarah")
print ("Nice to see you again.")
print ("Enjoy our video!")
width, length = 3, 4
area = width * length.
print ("Hi Dominque")
print ("      !")
print ("      !")
print ("Sarah  Dominque  peter")
print ("Hi     ")
print ("      !")
print ("      !")
```

A function — none great. The previous program piece is now a parameter with the name "name":

```
def greet(name):
    print("Hi", name)
    print("Nice to see you again!")
    print("Enjoy our video!")
print("program starts")
greet("peter")
# some lines of codes
the_answer = 42
greet("sarah")
width, length = 3, 4
area = width * length
greet("Domingo")

def f(x,y):
    z = 2*(x+y)
    return z

print("program starts!")
a = 3
res1 = f(a, 2+a)
print("Result of function call:", res1)
a = 4
b = 7
res2 = f(a, b)
print("Result of function call:", res2)
```

An easy way to explain is that "imperative" programming means that the computer get a list of commands and executes them in order, when "procedural progeamming" (which is also imperative) allows splitting those instructions into procedures (or functions).

Object Oriented Programming Paradigm

Class, Objects, Instances, Methods, Encapsulation, Data abstraction in Python.

* Python is an oop language we can easily create and use classes and objects in Python.

Python is a multi-paradigm programming language. It supports different programming approaches.

An object has two characteristics:

* attributes
* behavior.

Eg: Parrot - object.

* name, age, color - attributes.
* singing, dancing - behavior.

*. Reusable code.

```
class Parrot:
    # class attribute.
    species = "bird"
    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age.
# instantiate the parrot class.
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)
# access the class attributes.
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))
```

access the instance attributes

~~print ("{} is {} years old". format (blu.name, blu.age))~~

print ("{} is {} years old". format (blu.name, blu.age))

print ("{} is {} years old". format (woo.name, woo.age))

. Class: Parrot.

Attributes - Characteristics of an object

are defined inside the __init__ method.

It is the initializer method that is first run as soon as the object is created.

Then we create instances of Parrot class. Here, blu and woo are references to new objects.

We can access the class attribute using __class__.species. are same for all instances of a class.

Similarly, we access the instance attributes using blu.name and blu.age. Instance attributes are different for every instance of a class.

Methods.

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

Creating Methods in Python.

```
class parrot:
    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age.
    # instance method
    def sing(self, song):
        return "{} sings {}".format
                (self.name, song)

    def dance(self):
        return ("{} is now dancing".
                format(self.name)

# instantiate the object
    blu = parrot("Blu", 10)
# call instance methods
    print(blu.sing("'Happy'"))
    print(blu.dance())
```

o/p:

Blu sings 'Happy'

Blu is now dancing

Inheritance


```
#parent class
Bird :
    def __init__(self):
        print("Bird is ready")

    def whoisThis(self):
        print("Bird")

    def swim(self):
        print("swim faster")

#child class
class Penguin(Bird):

    def __init__(self):
        #call super() function
        super().__init__()
        print("penguin is ready")

    def whoisThis(self):
        print("Penguin")
    def run(self):
        print("run faster")

peggy = Penguin()
peggy = whoisThis()
peggy.swim()
peggy.run()
```

O/P = Bird is ready
penguin is ready
~~Bird~~ penguin
swim faster

Encapsulation

Using OOP in Python, we can restrict access to methods and variables. This prevents data from direct modification which is called encapsulation. In Python, we denote private attributes using underscore as the prefix i.e single _ or double __.

Data Encapsulation in Python.

```
class Computer:
    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling price: {}".
              format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

Output

```
Selling price : 900
Selling price: 900
Selling price: 1000.
```

Polymorphism

Polymorphism is an ability (in OOPs) to use a common interface for multiple forms (data types).

Suppose, we need to color a shape, there are multiple shape options (rectangle, square, circle). However we could use the same method to color any shape. This concept is called Polymorphism.

```
class parrot:
    def fly(self):
        print("parrot can fly")
    def swim(self):
        print("parrot can't swim")
class penguin:
    def fly(self):
        print("penguin can't fly")
    def swim(self):
        print("penguin can swim")
```