E-MODULE

COMPUTATIONAL PROBLEM SOLVING USING PYTHON

SUBMITTED BY:
RANI CHANDI
(DEPARTMENT OF COMPUTER SCIENCE AND IT)
CLASS:-BSC(IT)-SEM III
SUBJECT:-INTRODUCTION TO PYTHON

INTRODUCTION TO PYTHON

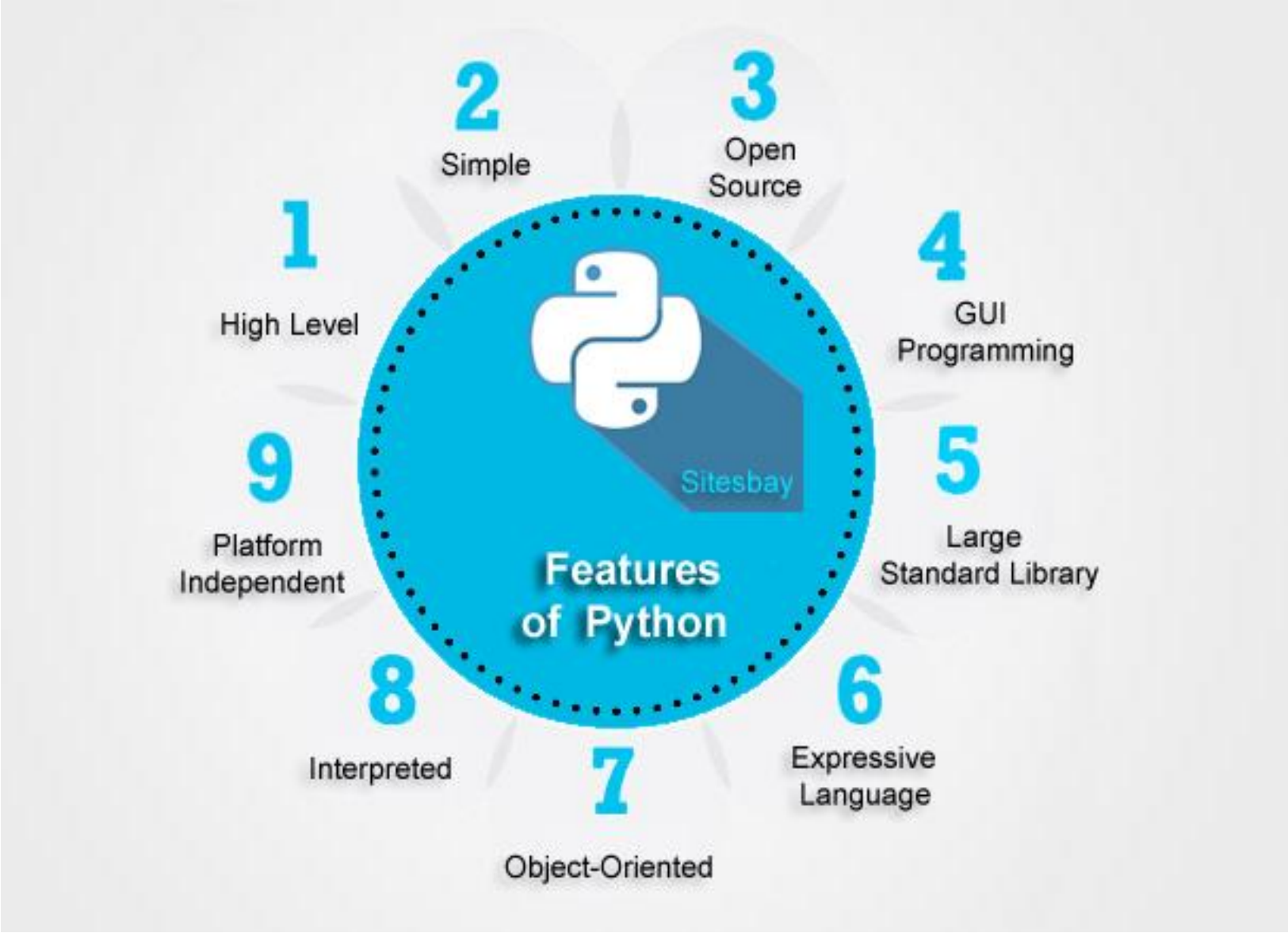
1) **Python** is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991.
2) An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java.

python
powered

```
print("Hello, world!")
```

1) Python features a dynamic type system and automatic memory management.
2) It supports multiple programming paradigms, including object-oriented, imperative, functional and procedu ral, and has a large and comprehensive standard library.
3) Python interpreters are available for many operating systems.
4) CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations.
5) CPython is managed by the non-profit Python Software Foundation.

FEATURES AND PHILOSOPHY OF PYHTON

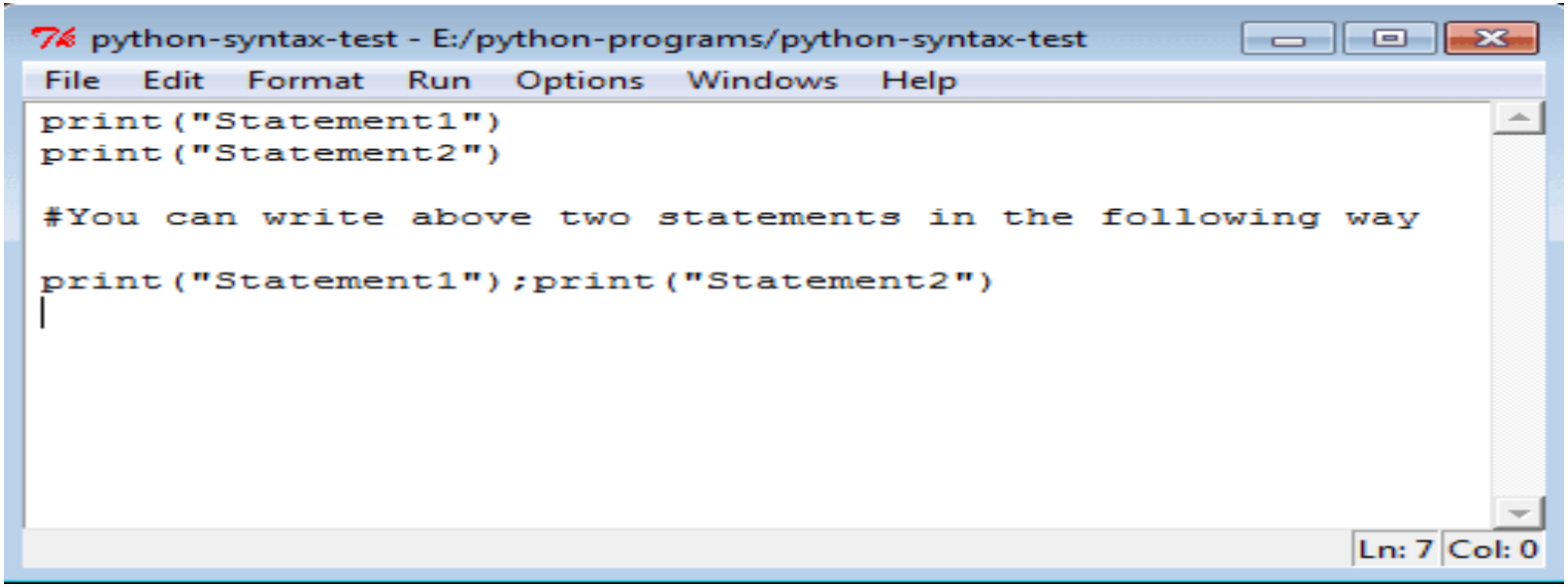


1) Python is a multi-paradigm programming language.
2) Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)).
3) Many other paradigms are supported via extensions, including design by contract and logic programming.
4) Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management.
5) It also features dynamic name resolution(late binding), which binds method and variable names during program execution.
6) Python's design offers some support for functional programming in the Lisp tradition.
7) It has filter(), map(), and reduce() functions; list comprehensions, dictionaries, and sets; and generator expressions.
8) The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

Features of Python

- A script language
- Interpreted
  - No compile/link stage
  - Write and run
  - Slow compared to C, C++
- Elegant design; "tight"
- Designed for
  - Quick-and-dirty scripts
  - Reusable modules
  - Very large systems
- Object-oriented
  - Very well-designed
  - But you don't have to use it
- Useful error messages
- Automatic memory handling
- Independent of operating system
  - Works on both Unix and Windows
  - Even file system code can be made os-independent
- Large library of standard modules
- Large number of third-party modules
- Integration with external C code
- Well documented

PYTHON SYNTAX AND SEMANTICS



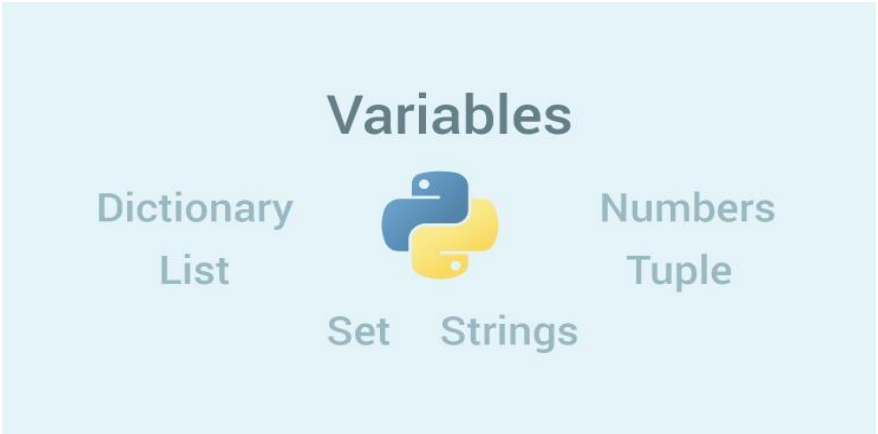
Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Indentation

Python uses whitespace indentation, rather than curly braces or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.This feature is also sometimes termed the off-side rule

DATA TYPES OF PYTHON

Type	Example
• Numeric: Integer, Float	x = 10 x = 1.0
• String	x= 'Mike'
• Boolean	y = True x = False
• List	my_list = [10, 20, 30]
• Tuple	my_tuple = ('Brett', 'Cisco', 'Cary',2015)
• Dictionary	my_dict = {"one":1, "two":2}
• Lists in Lists	my_list2=[[10,20,30], ['Cisco Live', 'May', 2016]]



Built-in Data types

Python's built-in (or standard) data types can be grouped into several classes. Sticking to the hierarchy scheme used in the official Python documentation these are **numeric types, sequences, sets and mappings** (and a few more not discussed further here). Some of the types are only available in certain versions of the language as noted below.

1. **BOOLEAN**: the type of the built-in values True and False. Useful in conditional expressions, and anywhere else you want to represent the truth or falsity of some condition. Mostly interchangeable with the integers 1 and 0. In fact, conditional expressions will accept values of any type, treating special ones like boolean False, integer 0 and the empty string "" as equivalent to False, and all other values as equivalent to True. But for safety's sake, it is best to only use boolean values in these places.
2. **NUMERIC TYPES:**

   int: Integers; equivalent to C longs in Python 2.x, non-limited length in Python 3.x

   long: Long integers of non-limited length; exists only in Python 2.x

   float: Floating-Point numbers, equivalent to C doubles

   complex: Complex Numbers

Sequences:

str: String; represented as a sequence of 8-bit characters in Python 2.x, but as a sequence of Unicode characters (in the range of U+0000 - U+10FFFF) in Python 3.x

bytes: a sequence of integers in the range of 0-255; only available in Python 3.x

byte array: like bytes, but mutable (see below); only available in Python 3.x

list

Tuple

Some *immutable* types	Some *mutable* types
• int, float, long, complex • str • bytes • tuple • frozen set • Boolean • array	• byte array • list • set • dict

COMPUTATIONAL PROBLEM SOLVING USING PYTHON

- **The Process of Computational Problem Solving**

Computational problem solving does not simply involve the act of computer programming. It is a process , with programming being only one of the steps. Before a program is written, a design for the program must be developed. And before a design can be developed, the problem to be solved must be well understood. Once written, the program must be thoroughly tested.

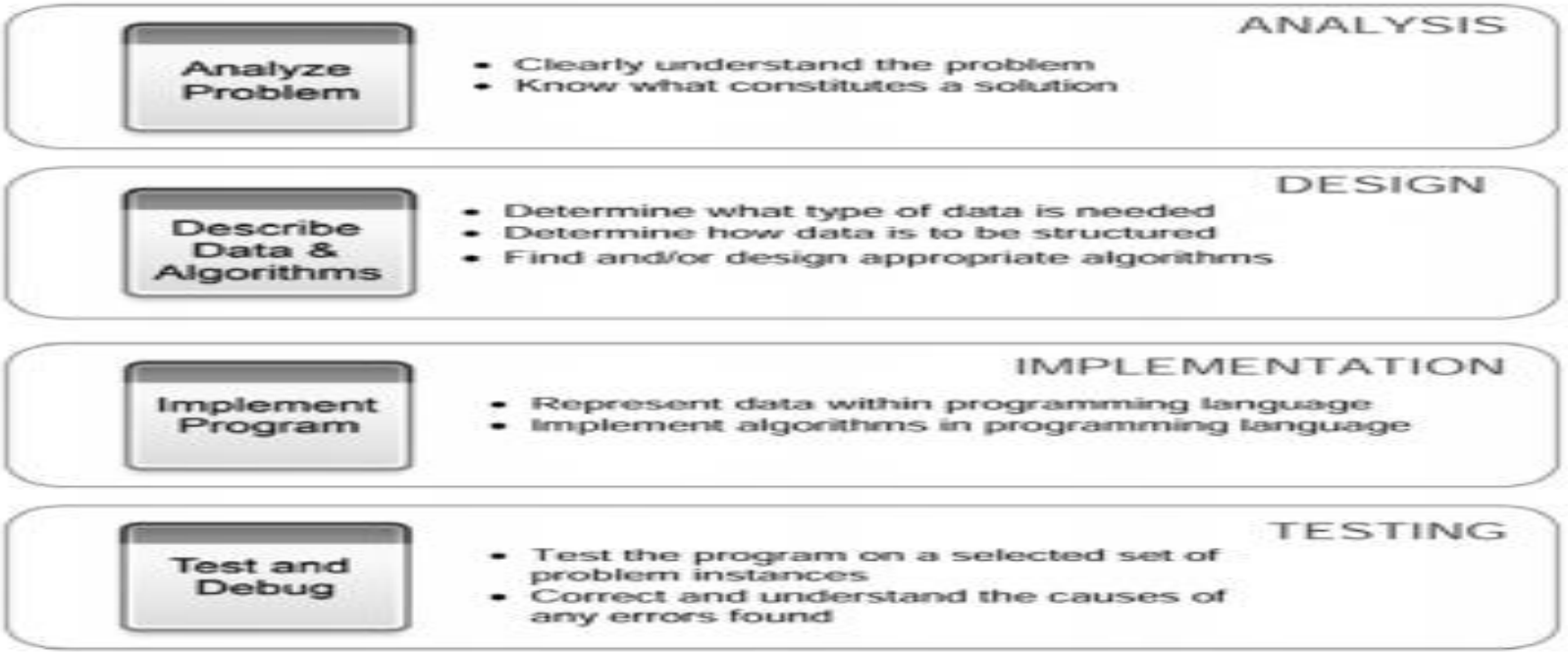


FIGURE 1-22 Process of Computational Problem Solving

1. Many systems in the natural sciences can be represented by means of a mathematical expression, known as a mathematical model.
2. This expression may be fairly complex in form, and thus it may not be possible to make a reasonable application of the standard methods of evaluation to the expression.
3. If this is true, then it is standard practice to make an attempt at evaluating the expression by various approximation methods. In particular, by methods that involve computational algorithms.
   For any problem that we must consider there are a number of steps that are suggested for one to take in the process of finding a computational solution of the problem, and then for assessing the viability of this solution. We suggest the following five steps:

The Five Step Process

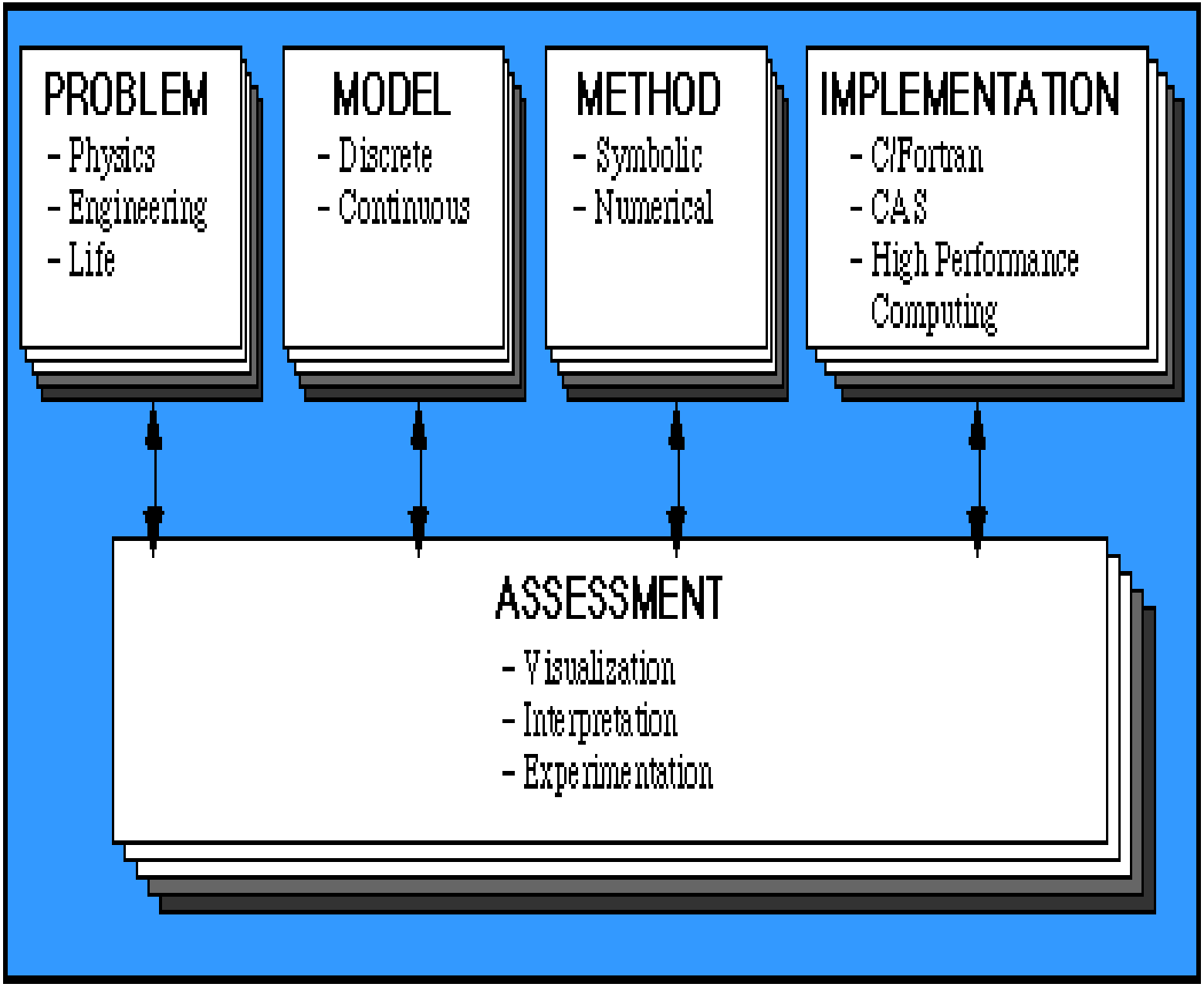
1) Identify the problem.

2) Express the problem in terms of a mathematical model.

3) Construct a computational method for solving the model.

4) Implement the computational method on a computer.

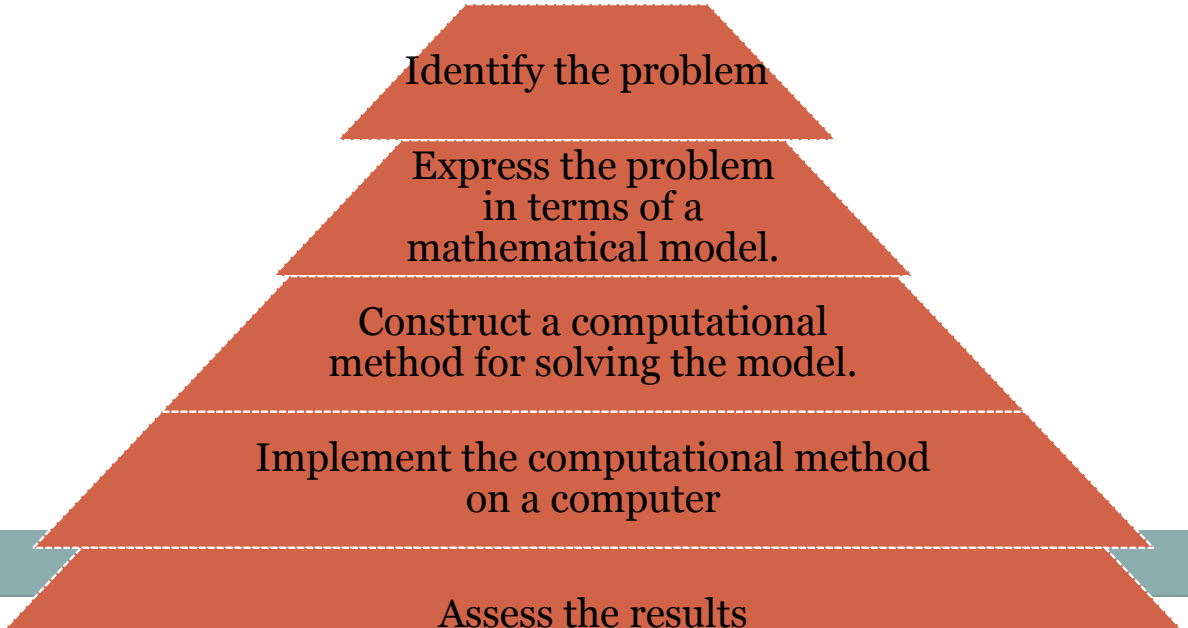
5) Assess the results; if there is a relatively large disparity between your solution and the actual solution, or what you know can actually be true for the problem, then reassess the problem and try the process again.


PROBLEM
- Physics
- Engineering
- Life
MODEL
- Discrete
- Continuous
METHOD
- Symbolic
- Numerical
IMPLEMENTATION
- C/Fortran
- CAS
- High Performance Computing
ASSESSMENT
- Visualization
- Interpretation
- Experimentation

Applying the Five Step Process

Gaining a better understanding of this five step process can best be accomplished by applying it to an actual problem. One such problem would be that of finding the area of a region bounded within a set of known curves.

Example: Consider the area of the region in the xy-plane that is above the x-axis and below the curve $y = x^2$ (a parabola) for those values of x that range over the interval from 0 to 4. By implementing this five step process, we shall attempt to find an approximation of the area of this region.



1. Identify the problem. We wish to find the area of the region that lies above the x-axis and below the parabola $y = x^2$ for values of x between 0 and 4. We try to incorporate all directly or indirectly supplied information such as the equation $y = x^2$ for the parabola, the equation $y = 0$ for the x-axis, and the interval $[0, 4]$ on which the values of x are restricted.

2. Express the problem in terms of a mathematical model. This problem can be approached by finding the definite integral of the function x^2 from 0 to 4, in which we will obtain the exact value of the area of the region, or it can be approached by approximating the region defined by our curves with a collection of rectangles and then by approximating the area of the region by summing the areas of the rectangles

3. Construct a computational method

1. Since integration is more of an analytical method, and one for which the reader may be unfamiliar, we are going to take the numerical approach of approximating the area with the sum of areas of rectangles.
2. To do this we will need to construct a series of rectangles that not only approximate the region, but also we need rectangles whose areas are easy to compute.
3. This can be accomplished by constructing rectangles, lying side by side across the region, whose sides are parallel to the coordinate axes--rectangles with vertical and horizontal sides.

4. Implement the computational method

1. Let us take the graph of our curves and build a series of four rectangles of equal width, whose bases lie along the line $y = 0$ (the x-axis), and whose heights reach the curve $y = x^2$ at the point above the right endpoint of each base (which also happens to be the highest point on $y = x^2$ above each base).
2. This can best be accomplished with rectangles of width 1 unit having bases along the intervals [0, 1], [1, 2], [2, 3], and [3, 4]. The resulting graph gives you a series of rectangles of width 1 and heights 1, 4, 9, and 16.
3. Finding the sum of their areas, one obtains a value of 30 square units.

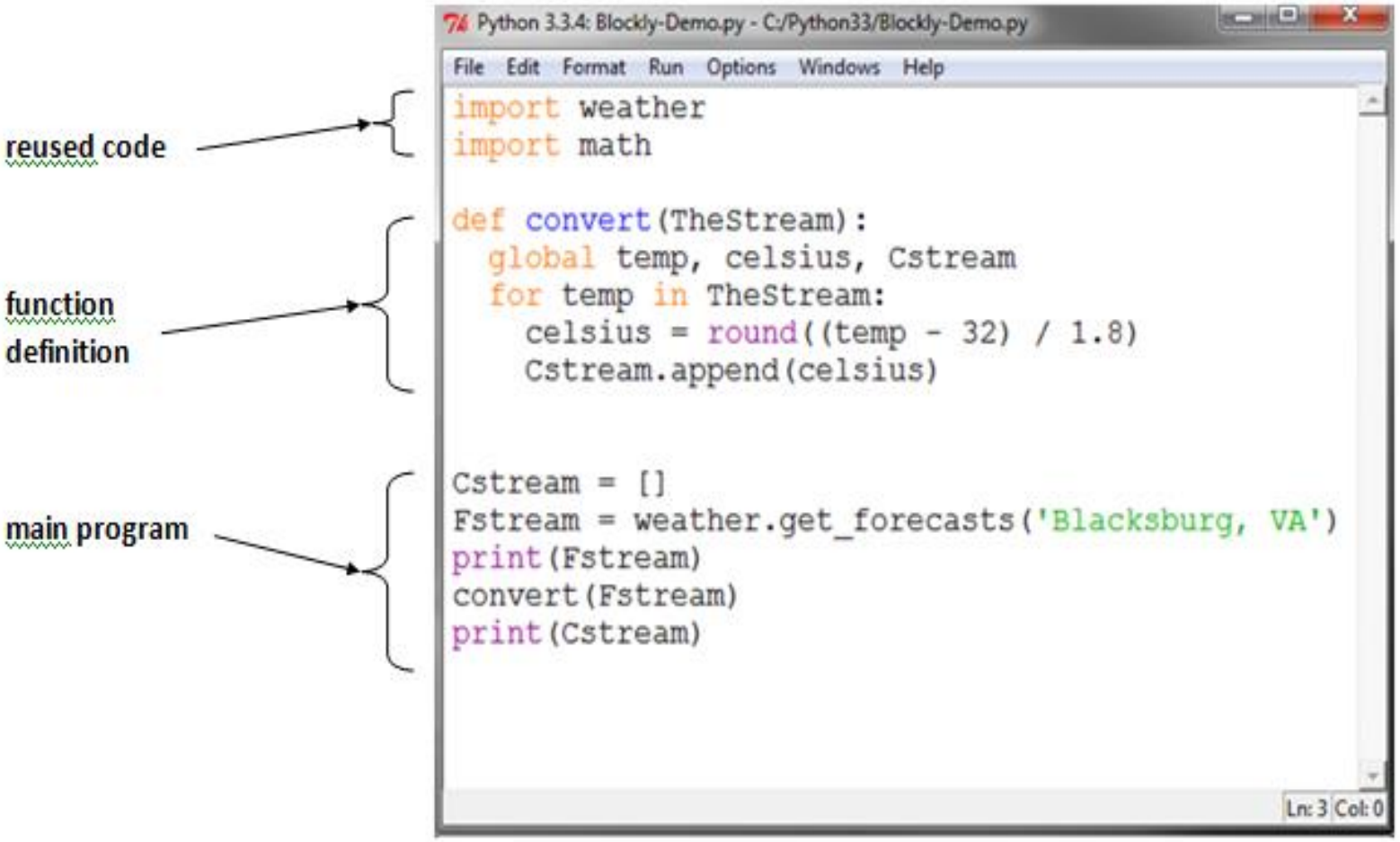
5. Assess the results

1. With this computational method providing an answer of 30 square units, we must now consider how this compares with the actual area of the region.
2. Since the area of our region lies within a triangle of base 4 and height 16, we see that our region must have an area that is greater than 0 and less than the area of this triangle, which is 32 square units.
3. Since our approximation of 30 square units does lie within these bounds (although our approximation is necessarily greater than the actual area), we see that our model yields a value that is consistent with what we would expect of the solution of the problem, although it may not yield a satisfactory approximation.
4. Assuming that there is no fault in our model, how can we obtain a more accurate approximation? Increase the number of rectangles (say, eight rectangles, each of width 0.5 units), and see how the corresponding solution is affected. (This should yield a better approximation.)

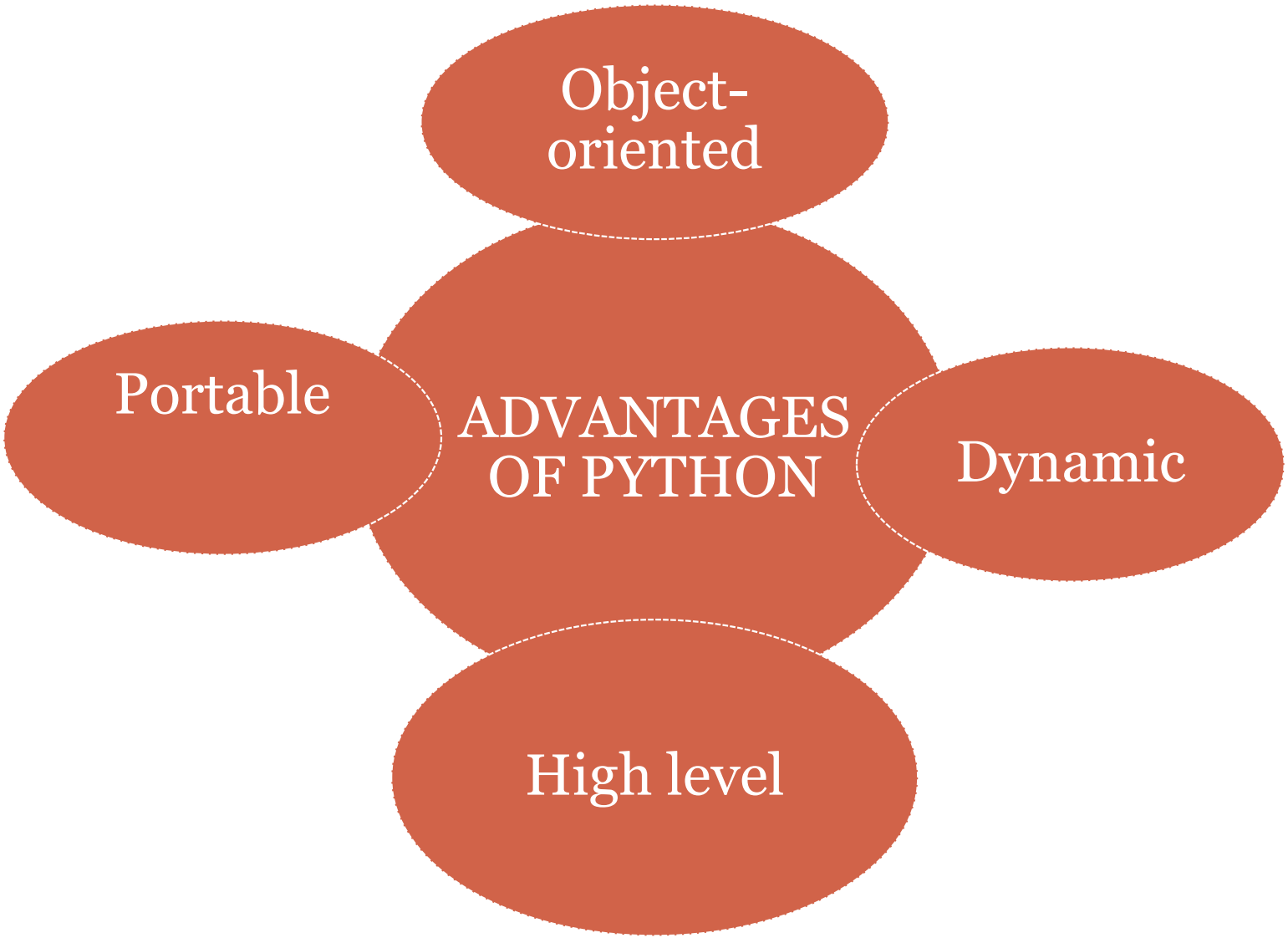
VERIFICATION OF COMPUTATIONAL MODEL

1) Verification of a computational method can be difficult. There are a number of things that we must consider with computational methods.
2) A computational method may yield a solution that fits reasonably well with what we would expect from our mathematical model, but it may do so in error--as a fluke. A slight change in the model or in some of the parameters that make up the model may bring out evidence of a fault in the computational method.
3) Thus it is always best to test a computational method by applying it to model whose solution is known exactly, and even, if possible, to apply the method to a model in which the method should yield the exact solution (in the above case, find the area of a large rectangle by summing the areas of smaller rectangles). In contrast, a computational method may be designed well in theory, but not work well in practice.
4) Error from the round-off of various numerical values may propagate through subsequent computations and affect the solution, although the method of deriving the solution is without fault.
5) Human error may also make a deceptive contribution to the application of a computational method.

STRUCTURE OF A PYTHON PROGRAM



ADVANTAGES OF PYTHON PROGRAMMING LANGUAGE



Productivity

With its strong process integration features, unit testing framework and enhanced control capabilities contribute towards the increased speed for most applications and productivity of applications. It is a great option for building scalable multi-protocol network applications

Improved Programmer's Productivity

The language has extensive support libraries and clean object-oriented designs that increase two to ten fold of programmer's productivity while using the languages like Java, VB, Perl, C, C++ and C#.

Integration Feature

Python integrates the Enterprise Application Integration that makes it easy to develop Web services by invoking COM or COBRA components. It has powerful control capabilities as it calls directly through C, C++ or Java via Jython. Python also processes XML and other markup languages as it can run on all modern operating systems through same byte code.

Extensive Support Libraries

It provides large standard libraries that include the areas like string operations, Internet, web service tools, operating system interfaces and protocols. Most of the highly used programming tasks are already scripted into it that limits the length of the codes to be written in Python

Thank You