1)

The languages considered here are C and Python

Simplicity and Readability

In terms of simplicity and readibility Python is the best langauage in the field of programming.It has simple and easy syntax and it is very easy to understand even from the view of an outsider and hence it is way ahead in terms of readability and simplicity than C.

Clarity about Binding

In terms of binding C is much more well defined and better than Python.C is a static typed language and hence it is much more clear in binding and everything is resolved in compile time whereas Python is a dynamic typed language and it uses an interpreter and everything is resolved in run time and hence the running time of C program is far less than running time of Python.

Reliability

In terms of reliability Python is a much more reliable language than C because in C we have a lot of problems fundamentally like the dangling else and memory leak which may change the output for the same input when run again and again (mainly memory leak does this).Support program verification and testing is furthermore much more easy and reliable in Python than in C.Python has a much simpler syntax and easier to design parser than C because there are some special cases in C with a lot of exceptions(for example one line loop doesn't require braces).

Support

In terms of support both C and Python have considerable support in the internet but in case of more complicated and realtime scenarios, Python has much larger support across domains and is of atmost ease for any programmer to refer to and build his solution towards a problem considered.

Abstraction

In terms of abstraction it is easy to conclude that Python is far more abstract language than C language.

The presence of Classes and Polymorphism are very powerful concepts which provide abstract features to Python.
The presence of name mangling provides data abstraction and the above mention concepts provide function abstractions.

Orthogonality

In terms of Orthogonality Python is more orthogonal than C.

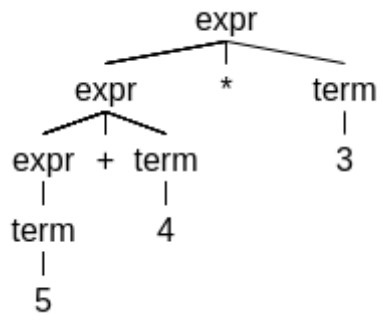
Efficient Implementation

In terms of efficiency of implementation there are a few factors that need to be considered.
If we consider runtime, then C program runs 3 times faster than an equivalent Python program.
In terms of program load time, C is faster than Python.
In terms of compile time,C is faster than Python because any compiler is faster than any interpreter.
In terms of program writing time, Python is easier than C.
On the whole we can say that Python is easier to type but the more efficient language in terms of runtime is C.

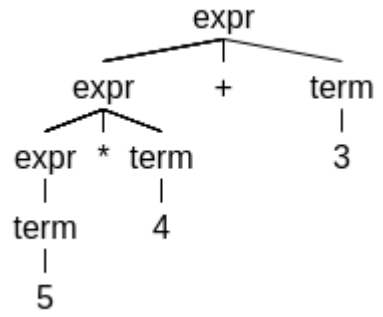
---

2)

(a) 5 + 4 * 3

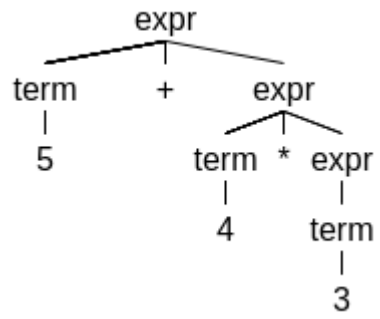


(b) 5 * 4 + 3

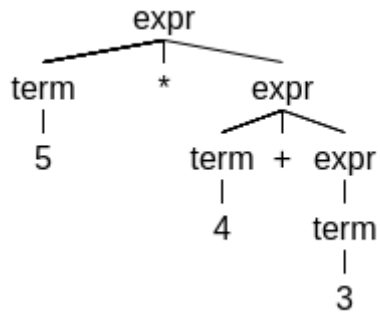


3)

(a) 5 + 4 * 3



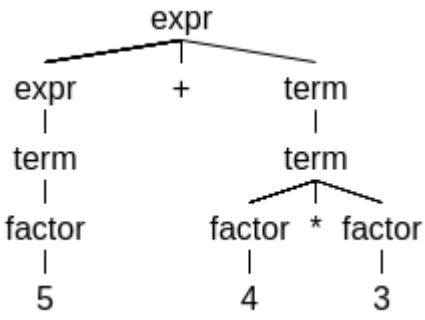
(b) 5 * 4 + 3



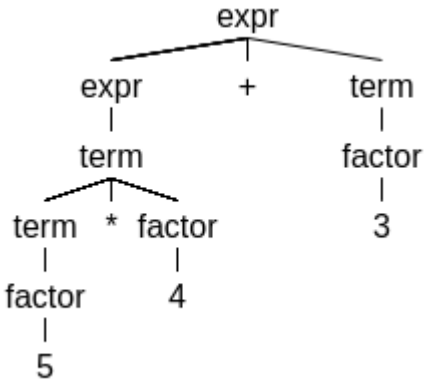
---

4)

(a) 5 + 4 * 3

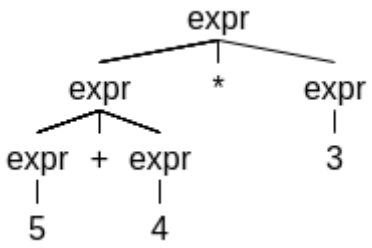


(b) 5 * 4 + 3

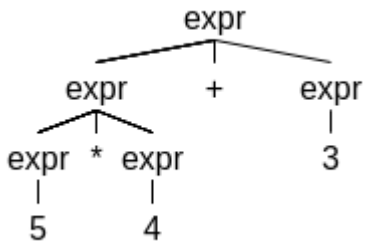


5)

(a) 5 + 4 * 3



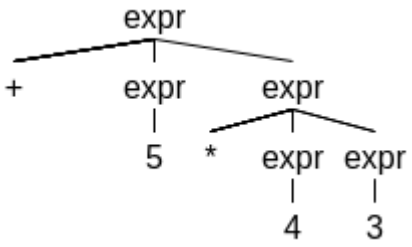
(b) 5 * 4 + 3



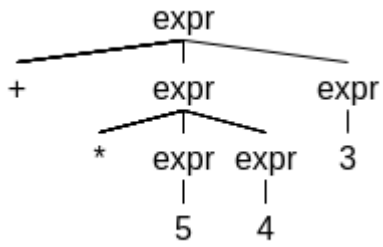
---

6)

(a) + 5 * 4 3



(b) + * 5 4 3



---

7)

Grammar for if else statement

*S -> M | U
M -> iMeM | a
U -> iM | iMeU

Perl

```
$a = 100;
if( $a < 20 ) {
   printf "a is less than 20\n";
}
else {
   printf "a is greater than 20\n";
}
```

Python

```
a=100
if a<20:
    print("a is less than 20")
else:
    print("a is greater than 20")
```

Ada

```
procedure check is
    i: integer := 2;
begin
    if i = 2 then
        i := i + 1
    else
        i := i + 2
    end if;
end check;
```

---

8)

Declaring an entity

When you declare a variable, a function, or even a class all you are doing is saying: there is something with this name, and it has this type.
The compiler can then handle most (but not all) uses of that name without needing the full definition of that name.

Declaring a value without defining it allows you to write code that the compiler can understand without having to put all of the details.
This is particularly useful if you are working with multiple source files, and you need to use a function in multiple files. You don't want to put the body of the function in multiple files, but you do need to provide a declaration for it.

```
int func();
```

This is a function declaration; it does not provide the body of the function, but it does tell the compiler that it can use this function and expect that it will be defined somewhere.

Defining an entity

Defining something means providing all of the necessary information to create that thing in its entirety.
Defining a function means providing a function body; defining a class means giving all of the methods of the class and the fields. Once something is defined, that also counts as declaring it; so you can often both declare and define a function, class or variable at the same time.But you don't have to.

```
int func();

int main()
{
    int x = func();
}

int func()
{
    return 2;
}
```

Since the compiler knows the return value of func, and the number of arguments it takes, it can compile the call to func even though it doesn't yet have the definition. In fact, the definition of the method func could go into another file

Conclusion

A declaration provides basic attributes of a symbol: its type and its name.
A definition provides all of the details of that symbol if it's a function, what it does; if it's a class, what fields and methods it has; if it's a variable, where that variable is stored.

Often, the compiler only needs to have a declaration for something in order to compile a file into an object file, expecting that the linker can find the definition from another file.
If no source file ever defines a symbol, but it is declared, you will get errors at link time complaining about undefined symbols.

---

9)

Header files contain definitions of Functions and Variables, which is imported or used into any C++ program by using the preprocessor #include statement. Header file have an extension ".h" which contains C++ function declaration and macro definition. When we want to use any function in our C++ program then first we need to import their definition from C++ library, for importing their declaration and definition we need to include header file in program by using #include. Header file include at the top of any C++ program.

For example if we use clrscr() in C++ program, then we need to include, conio.h header file, because in conio.h header file definition of clrscr() (for clear screen) is written in conio.h header file

Why Java doesn't use header files ?

Source code written in Java is simple.
There is no preprocessor, no #define and related capabilities, no typedef, and absent those features, no longer any need for header files.
Instead of header files, Java language source files provide the declarations of other classes and their methods.
A major problem with C and C++ is the amount of context you need to understand another programmer's code: you have to read all related header files, all related #defines, and all related typedefs before you can even begin to analyze a program.
In essence, programming with #defines and typedefs results in every programmer inventing a new programming language that's incomprehensible to anybody other than its creator, thus defeating the goals of good programming practices.

In Java, you obtain the effects of #define by using constants.
You obtain the effects of typedef by declaring classes--after all, a class effectively declares a new type.
You don't need header files because the Java compiler compiles class definitions into a binary form that retains all the type information through to link time.

By removing all this baggage, Java becomes remarkably context-free.
Programmers can read and understand code and, more importantly, modify and reuse code much faster and easier.

---

10)

Variable names and types:

a) Whenever a compiler compiles a code it stores the variables as {name, datatype} pairs. Each of the different pairs have separate memory allocated i.e. int x; float x; get stored as {x, int} and {x, float} respectively and hence compiler sees them as different variables and does the operation respectively. Further most datatypes differ in the amount of memory required for this datatype and hence they can be easily distinguished.

b) Overloading is a concrete example of runtime polymorphism. If such a restriction that same variable names cannot be used for overloading functions even though they are of different types exists, then it would become tedious for the user to remember a lot of variable names and hence the concept of overloading becomes tarnished in a large scale.
Allowing to differentiate function overloading by no of parameters and type of parameters helps the user write a more logically relatable code and it is easy for the user to manage variable names as they are reduced to a small subset.

---

11)

Examples of R-values that cannot be L-values

1.Expressions Eg- a+b,a-5
2.Constant values Eg- 1,2,3.5
3.Any function thata does not return a pointer.

Examples of L-values

1.The name of a variable that points to a memory location
2.The square bracket [] always yields an L-values
3.const object (specifically using the const keyword)

All l-values can be treated as r-values because l-values serve as an identity and anything that has an identity must evaluate to a value.
Since it evaluates to a value, we can say that any l-value is a r-value but the converse doesn't hold good.

Eg

```
    int a = 10;
    int b = a;  // here the lvalue a can be treated as a r-value
```

---

12)

Language: C

Type	Storage size	Value range
Char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
Int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
Short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
Long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295
Float	4 bytes	1.2E-38 to 3.4E+38(precision of 6 decimal places)
Double	8 bytes	2.3E-308 to 1.7E+308(precision of 15 decimal places)
long double	10 bytes	3.4E-4932 to 1.1E+4932(precision of 19 decimal places)

Language: C#

Type	Storage size	Value range
short	2 bytes	-32,768 to 32,767
Sbyte	1 byte	-128 to 127
int	4 bytes	-2,147,483,648 to 2,147,483,647
long	8 bytes	(-)9,223,372,036,854,775,808 to

		9,223,372,036,854,775,807
byte	1 byte	0 to 255
ushort	2 bytes	0 to 65,535
uint	4 bytes	0 to 4,294,967,295
ulong	8 bytes	0 to 18,446,744,073,709,551,615
float	4 bytes	1.5 × 10-45 to 3.4 × 1038(7-digit precision)
double	8 bytes	5.0 × 10-324 to 1.7 × 10308(15-digit precision)
decimal	16 bytes	–7.9 × 10-28 to 7.9 × 1028(min 28-digit precision)
bool	1 byte	holds true or false
char	2 bytes	U +0000 to U +ffff (follows UTF-16)

Language: Java

Type	Storage size	Value range
short	2 bytes	-32,768 to 32,767
byte	1 byte	-128 to 127
int	4 bytes	-2,147,483,648 to 2,147,483,647
long	8 bytes	(-)9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
char	2 bytes	0 to 65535
float	4 bytes	-3.40282347E+38F to +3.40282347E+38F(6 to 7 digit precision)
double	8 bytes	-1.79769313486231570E+308 to +1.79769313486231570E+308(15 digit precision)
boolean	1 bit (technically	holds true or false

	but not well known)	

Language : Python

Type

Numeric
(a) int - holds signed integers of non limited length
(b) float - holds floating precision upto 15 decimal digits
(c) complex - holds complex numbers
String
List
Set
Dictionary
Tuple

---

13)

Type-checking and type requirements are much tighter in Java.
For example:
1. Conditional expressions can be only boolean, not integral.
2. The result of an expression must be used.

Reason

All conditional expressions are boolean by default.
However,some languages like C,C++ allows us to use numerical expressions or pointers as conditional expressions.
When a non-boolean expression is used as a conditional expression, they are implicitly converted into comparisions with zero.
This feature is not supported in many high level languages like Java and C# because it does not produce a clear code.
Furthermore, it results in another set of comparisons which slows down the rate of execution due to type conversions.

Example
1.To check variable is equal to zero

```
if(i == 0){
}
```

2.To check variable is null or not

```
if(i!=null){
}
```

---

14)

```
#include<bits/stdc++.h>
using namespace std;
typedef long double ld;
string toBool(int n,bool state){
    string ans = "";
    while(n){
        if(n % 2 == 0)
            ans = "0" + ans;
        else
            ans = "1" + ans;
        n/=2;
    }
    if(state){
        int size = ans.size();
        for(int i = 31-size;i>0;i--)
            ans = "0" + ans;
    }
    return ans;
}
string toBoolFrac(ld val){
    string ans = "";
    while(ans.length() < 30){
        val = val * 2;
        if(floor(val) == 1){
            ans+="1";
            val = val - 1;
        }
        else{
            ans+="0";
        }
    }
    return ans;
}

int main(){
    string ans1="";
    string ans2="";
    int n = -10;
    int val = pow(2,31)+1;
    if(n>0)
        ans1 = "0" + toBool(n,true);
    else
        ans1 = "1" + toBool(val-n,false);
    cout<<"32 bit representation of "<<n<<" is "<<ans1<<endl;
    ld m = 0.3;
    if(m >= 0)
        ans2+="0";
```

```
else
    ans2+="1";
int temp = floor(m);
ld temp1 = m - temp;
string tt = toBool(temp,false);
string tt1 = toBoolFrac(temp1);
int exponent;
if(tt.size()>1){
    exponent = 127 + tt.size() - 1;
    string x = toBool(exponent,false);
    ans2+=x;
    for(int i = 1;i<tt.size();i++){
        if(ans2.size() == 32){
            break;
        }
        if(tt[i] == '0')
            ans2+="0";
        else
            ans2+="1";
    }
    if(ans2.size()!=32){
        for(int j = 0;j<tt1.size();j++){
            if(ans2.size() == 32){
            break;
        }
        if(tt1[j] == '0')
            ans2+="0";
        else
            ans2+="1";
        }
    }
}
else if(tt.size() == 1 && tt[0] == '1'){
    exponent = 127;
    ans2+="0"+toBool(exponent,false);
    for(int j = 0;j<tt1.size();j++){
        if(ans2.size() == 32){
            break;
        }
        if(tt1[j] == '0')
            ans2+="0";
        else
            ans2+="1";
    }
}
else{
    int index;
    for(int i = 0; i < tt1.size();i++){
        if(tt1[i] == '1'){
            index = i;
```

```
                break;
            }
        }
        index++;
        exponent = -index + 127;
        ans2+="0"+toBool(exponent,false);
        for(int j = index;j<tt1.size();j++){
                if(ans2.size() == 32){
                break;
            }
            if(tt1[j] == '0')
                ans2+="0";
            else
                ans2+="1";
        }
    }
    cout<<"32 bit representation of "<<m<<" is "<<ans2<<endl;
    return 0;
}
```

OUTPUT

```
32 bit representation of -10 is 11111111111111111111111111110110
32 bit representation of 0.3 is 00111110100110011001100110011001
```

---

15)

```
Number: 0.2
Sign Bit: 0
Exponent: 01111100
Fraction: 10011001100110011001100

Overall Representation: 00111110010011001100110011001100

Number: 0.5
Sign Bit: 0
Exponent: 01111110
Fraction: 00000000000000000000

Overall Representation: 00111111000000000000000000000000

Number: 0.3
Sign Bit: 0
Exponent: 01111101
Fraction: 00110011001100110011001

Overall Representation: 00111110100110011001100110011001

Number: 1.0
```

Sign Bit: 0
Exponent: 01111111
Fraction: 00000000000000000000000

Overall Representation: 00111111100000000000000000000000

---

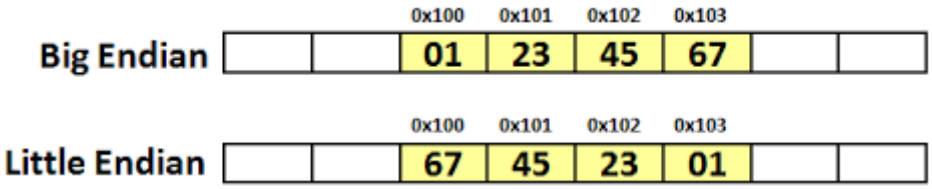
16)

Big Endian and Little Endian

Big Endian and Litte Endian are two methods of storing multi-byte data types like int,float,etc.

In Little Endian last byte of binary representation are stored first of the word and in Big Endian first byte of binary represenation is stored as first byte of the word.

Suppose an integer is stored as 4 bytes(32-bits), then a variable y with value 0x01234567(Hexa-decimal representation)is stored as four bytes 0x01, 0x23, 0x45, 0x67, on Big-endian while on Little-Endian (Intel x86), it will be stored in reverse order as 0x67, 0x45, 0x23, 0x01.



Advantages and Disadvantages

In “Little Endian” form, assembly language instructions for picking up a 1, 2, 4, or longer byte number proceed in exactly the same way for all formats: first pick up the lowest order byte at offset 0. Also, because of the 1:1 relationship between address offset and byte number (offset 0 is byte 0), multiple precision math routines are correspondingly easy to write It is easier to add and multiply.

In “Big Endian” form, by having the high-order byte come first, you can always test whether the number is positive or negative by looking at the byte at offset zero. You don’t have to know how long the number is, nor do you have to skip over any bytes to find the byte containing the sign information. The numbers are also stored in the order in which they are printed out, so binary to

decimal routines are particularly efficient.It is easy to compare numbers and also divide numbers.

Examples

Little Endian: IBM x86

Big endian: IBM S/390

Example 1:

Little Endian - 0000 0000 0011 0010

Big Endian   - 0011 0010 0000 0000

Example 2:

Little Endian - 0000 0000 0000 0010

Big Endian   - 0010 0000 0000 0000

Example 3:

Little Endian - 0000 0000 0000 0000 0000 0000 0000 0010

Big Endian   - 0010 0000 0000 0000 0000 0000 0000 0000

---

17)

Let us consider the language C++

Let us assume the integer data type has 32 bits(4 bytes).

For any n bit representation the max value that can be represented is 2^n -1.

In that case the integer data type can hold values upto (2^32)-1.

The range for this integer datatype is from -(2^31) to (2^31)-1.

If we consider unsigned integer datatype the range is from 0 to (2^32)-1.

In both the cases of signed and unsigned integer the datatype can hold only upto factorial of 12.
Beyond 12 it cannot hold values.

In that case the integer datatype cannot hold those values.
In case of unsigned integer some garbage value within the range gets thrown up.
In case of signed integer negative value or some positive garbage value get thrown up.

Incase of long long int, we can compute upto factorial of 20.

If we use Python we can compute factorial of any number because the integer in python can be arbitrarily large.

---

18)

Enums in C/C++

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants.
These names make a program easy to read and maintain.

Rules

1.The value assigned to enum names must be some integeral constant, i.e., the value must be in range from minimum possible integer value to maximum possible integer value.
2.All enum constants must be unique in their scope
3.Two enum names can have same value
4.If we do not explicitly assign values to enum names, the compiler by default assigns values starting from 0.
5.We can assign values to some name in any order. All unassigned names get value as value of previous name plus one.

Examples
1.

```
enum state  {working, failed};
enum result {failed, passed};

int main()  { return 0; }
```

This shows compilation error as failed has a previous declaration.

2.

```
#include<stdio.h>
enum year{Jan, Feb, Mar, Apr, May, Jun, Jul,
          Aug, Sep, Oct, Nov, Dec};
int main()
{
   int i;
   for (i=Jan; i<=Dec; i++)
      printf("%d ", i);

   return 0;
}
```

OUTPUT

```
0 1 2 3 4 5 6 7 8 9 10 11
```

3.
```
#include <stdio.h>
enum day {sunday = 1, monday, tuesday = 5,
          wednesday, thursday = 10, friday, saturday};

int main()
{
    printf("%d %d %d %d %d %d %d", sunday, monday, tuesday,
            wednesday, thursday, friday, saturday);
    return 0;
}
```

OUTPUT
1 2 5 6 10 11 12

Enum in Java

Java Enum is a type like class and interface and can be used to define a set of Enum constants.
Enum constants are implicitly static and final and you can not change their value once created.
Enum in Java provides type-safety and can be used inside switch statement like int variables.
Enums in Java are type-safe and has their own namespace.
It means your enum will have a type and you can not assign any value other than specified in Enum Constants.
Enum can implement an interface in Java

Examples

1.Not user defined values
```
public enum Currency {
    PENNY, NICKLE, DIME, QUARTER
};
Currency coin = Currency.PENNY;
```

2.User defined values for enum
```
public enum Currency{
    PENNY(1), NICKLE(5), DIME(10), QUARTER(25)
    private int value;
    private Currency(int value){
        this.value = value;
    }
};
Currency coin = Currency.PENNY;
```
Here new method should not be used for the construction of enums.

3.Enum can be used with switch statements
```
Currency usCoin = Currency.DIME;
    switch (usCoin) {
```

```
            case PENNY:
                    System.out.println("Penny coin");
                    break;
            case NICKLE:
                    System.out.println("Nickle coin");
                    break;
            case DIME:
                    System.out.println("Dime coin");
                    break;
            case QUARTER:
                    System.out.println("Quarter coin");
    }
```

Differences between Enum in C++ and Java

C++	Java
In C++ enum is a named set of integral constants.	In Java enum is like a named instance of a class.
In C++ the conversion of enum to integer values must be implicit.	In Java conversion must be explicit.
In C++ enum does not have special support systems in its library.	Java has support for enum like EnumSet and EnumMap for organizing enums internally.
There is no provison to define methods for enum	We can define methods for enum in Java.

---

19)

Like Python Ada also supports array slicing

Example

```
Ada
with Text_IO;

procedure Main is
   Text : String := "ABCDE";
begin
   Text_IO.Put_Line (Text (2 .. 4));
end Main;
```

OUTPUT

```
BCD
```

Python

```
a="ABCDE"
print(a[2:4])

OUTPUT
CD
```

Difference with Python
1.Python uses 0 based indexing while Ada uses 1 based indexing.
2.Python doesn't include the last index while slicing while Ada includes the last index.

Applications
1.Finding top 10 ranks in a class.
2.Finding last 5 ranks in a class.
3.Finding a substring in a string.
4.Computations that involve ranges uses slicing.

---

20)

C/C++

In C/C++,dynamic arrays can be created using pointers and some functions like malloc and calloc
Example

```
int *a;
int n;
scanf("%d",&n);
a = (int*)malloc(n*sizeof(int));
    (or)
a = (int*)calloc(n,sizeof(int));
```

In C++ alone the following implementaion exists and not in C

```
int n;
cin>>n;
int a[n];
```

JAVA

In Java,dynamic arrays can be created using ArrayList.
Arrays have fixed size because they work on built-in datatypes but the ArrayList works on Objects so dynamic size is achieved.
Example

```
List<Integer> list = new ArrayList<Integer>();
list.add(1);
list.add(2);
list.add(3);
```

PYTHON

In Python,dynamic arrays can be created using list datatype.

The size of a list is not restricted in Python.

Example

```
a = []
a.append(1)
print(a)
[1]
a.append(2)
print(a)
[1,2]
a.append(3)
print(a)
[1,2,3]
```