**Introducción**

**1.-Definición de lenguaje procedural**

Está basado en estructurar el código de un programa en componentes que se les llaman procedimientos, subrutinas o funciones. Este tipo de programación puede ser ejecutada tantas veces sea necesario dentro de su programa.

Indicar lenguajes de programación y ejemplos de código:

C

#include *<stdio.h>*

int main (int argc,char \*\*argv)

{

  printf("Hola mundo**\n**");

**return** 0;

}

Pascal

**program** Project1;

**begin**

writeln('¡Hola mundo!');

**end**.

Fortran

program hello

write(6,\*)'Hello world'

stop

end

Basic

CLS

PRINT "Hola Mundo"

**2.- Definición de lenguaje orientada a objetos**

Se le llama así a cualquier lenguaje de programación que implementa los conceptos definidos por la OOP

Indicar lenguajes de programación y una reseña de creación y evolución (C++, Java, Python, C#)

**\*C#**

Durante el desarrollo del .NET Framework, sus bibliotecas de clases fueron escritas mediante un compilador de código administrado denominado Simple Managed C (SMC). En enero de 1999, [Anders Hejlsberg](http://en.wikipedia.org/wiki/Anders_Hejlsberg), creador de lenguajes como Turbo Pascal y Delphi, formó un equipo para diseñar un nuevo lenguaje, cuyo nombre inicial fue Cool (C-like Object Oriented Language). Microsoft quiso mantener dicho nombre como definitivo pero por razones de registro de marca tuvo que echarse para atrás.

Con el C# se pretendió que incorporar las ventajas o mejoras que tiene el lenguaje JAVA. Así se consiguió que tuviese las ventajas del C, del C++, pero además la productividad que posee el lenguaje JAVA y se le denomino C#.

Algunas de las características del lenguaje de programación C# son: Su código se puede tratar íntegramente como un objeto. Su sintaxis es muy similar a la del JAVA. Es un lenguaje orientado a objetos y a componentes. Armoniza la productividad del Visual Basic con el poder y la flexibilidad del C++. Ahorramos tiempo en la programación ya que tiene una librería de clases muy completa y bien diseñada.

A pesar que el lenguaje C# forma parte de la plataforma .NET, que es una interfaz de programación de aplicaciones. C# es un lenguaje independiente que originariamente se creó para producir programas sobre esta plataforma .NET.

Esta plataforma se creó, entre otras razones, porque el Visual Basic era uno de los lenguajes de programación que se encargaban de desarrollar estas aplicaciones. Pero el Visual Basic es un lenguaje orientado a objetos algo pobre, porque se quiso que fuese desde su creación un lenguaje fácil de aprender para los programadores novatos. Por esto, surgió el C#, para suplir esta deficiencia del Visual Basic.

El Visual Basic no tiene algunas de las características necesarias como la herencia, los métodos virtuales, la sobrecarga de operadores, etc. Que se han conseguido con el C# y la plataforma .NET.

**\*C++**

**C++** es un [lenguaje de programación](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) diseñado a mediados de los años 1980 por [Bjarne Stroustrup](http://es.wikipedia.org/wiki/Bjarne_Stroustrup). La intención de su creación fue el extender al [lenguaje de programación](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) [C](http://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)) mecanismos que permiten la manipulación de [objetos](http://es.wikipedia.org/wiki/Objeto_(programaci%C3%B3n)). En ese sentido, desde el punto de vista de los [lenguajes orientados a objetos](http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos), el C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de [programación genérica](http://es.wikipedia.org/wiki/Programaci%C3%B3n_gen%C3%A9rica), que se sumaron a los paradigmas de programación [estructurada](http://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada) y [programación orientada a objetos](http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos). Por esto se suele decir que el C++ es un [lenguaje de programación multiparadigma](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_multiparadigma).

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes, tales como ROOT.

Una particularidad del C++ es la posibilidad de redefinir los [operadores](http://es.wikipedia.org/wiki/Operadores_de_C_y_C%2B%2B), y de poder crear nuevos [tipos](http://es.wikipedia.org/wiki/Tipo_de_datos) que se comporten como tipos fundamentales.

El nombre **C++** fue propuesto por [Rick Mascitti](http://es.wikipedia.org/w/index.php?title=Rick_Mascitti&action=edit&redlink=1) en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases".

En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

**\*PYTHON**

Python fue creado a finales de los ochenta por Guido van Rossum en CWI en los Países Bajos como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.

En 1991, van Rossum publicó el código (versión 0.9.0) en alt.sources. En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones, funciones, y los tipos modulares: list, dict, str y así sucesivamente. Además en este lanzamiento inicial aparecía un sistema de módulos adoptado de Modula-3; van Rossum describe el módulo como "uno de las mayores unidades de programación de Python". El modelo de excepciones en Python es parecido al de Modula-3, con la adición de una cláusula else . En el año 1994 se formó comp.lang.python, el foro de discusión principal de Python, marcando un hito en el crecimiento del grupo de usuarios de este lenguaje.

Python 2.0 introdujo además un sistema de recolección de basura capaz de recolectar referencias cíclicas. Las adiciones a la biblioteca estándar de Python y las decisiones sintácticas fueron influenciadas fuertemente por Java en algunos casos.

**\*JAVA**

Java nace en 1991 con el nombre "OAK", posteriormente cambiado por Green por problemas legales, y finalmente con la denominación actual JAVA.

El objetivo de java era crear un lenguaje de programación parecido a C++ en estructura y sintaxis, fuertemente orientado a objetos, pero con una máquina virtual propia. Esto se hizo bajo el principio, de poder ser usado bajo cualquier arquitectura "Write Once, Run Anywhere (escríbelo una vez, ejecútalo en cualquier sitio)".

En 1995 finalmente, es presentada la versión alpha de java, y un año después en 1996 es lanzado el primer JDK (JDK 1.0). El desarrollo de java a partir de entonces es imparable, se van presentando nuevos paquetes y librerías hasta la actualidad.

A día de hoy, podemos decir, que Java es uno de los lenguajes más importantes del mundo. Con una comunidad extendida en todos los componentes y más de 4 millones de desarrolladores, existen millones de dispositivos que lo usan. Además, tras el surgimiento de android, java se establecido como el lenguaje de programación para móviles más extendido del planeta.

**3.- Cuadro comparativo de estos dos paradigmas**

|  |  |
| --- | --- |
| **Paradigma Procedural** | **Paradigma Orientada a Objetos** |
| -Es un tipo de programación imperativa, donde las declaraciones se ponen procedimientos, que se pueden volver a llamar cuando sea necesario.  -No necesita objetos, tiene procedimientos que podrían ser estructuras de datos, rutinas y subrutinas.  -Las funciones se denominan PROCEDIMIENTOS.  -Usa una llamada de procedimiento para llamar una función.  -No admite la herencia ya que carece de objetos y la herencia sólo es aplicable a estos mismos.  -Son más fáciles de entender.  -Puede ser leído en secuencia, de arriba hacia abajo, sin tener que saltar de un sitio a otro.  -la estructura del programa es más clara puesto que las instrucciones están más relacionadas entre sí.  -Consta de una estructura donde se ejecuta paso a paso y se debe tener una secuencia y una lógica para ser eficiente.  -Pretende resolver un problema de principio a fin en una sola estructura de código..  -Orientada a los procesos del sistema.  -consta de: ANÁLISIS, DISEÑO, CODIFICACIÓN, PRUEBAS, INTEGRACIÓN. | -La OOP es un estilo que trata los datos como objetos con atributos y  métodos que pueden aplicarse a estos objetos y también ser heredados por otros objetos.  -Se centra en clases y objetos. Al representar las variables dadas como objetos, se le puede pasar una función o método. Un objeto que pertenece a una clase en particular se puede tratar independientemente.  -Las funciones se le conoce también como métodos.  -Se utiliza llamada de mensaje para llamar o  solicitar acciones a un objeto.  -Permite el uso de la herencia, ya que permite una facilidad general a través del cual el código se puede reutilizar y extender sin cambiar el código que ya existe. Los objetos nuevos son capaces de heredar las propiedades de los objetos más antiguos.  -Reusabilidad: cuando hemos diseñado adecuadamente las clases, se pueden usar en distintas partes del programa y en varios proyectos.  -Mantenibilidad: Es más sencillo de leer y comprender, nos permiten ocultar detalles de implementación dejando visibles sólo los detalles que sean más relevantes.  -Modificabilidad: Facilidad de añadir, suprimir o modificar nuevos objetos nos permite hacer modificaciones de una forma muy sencilla.  -Se puede dividir  el problema en partes más pequeñas para probarlas de manera independiente y más fácilmente poder ver los errores que puedan surgir.  -Se basa en una nueva forma de pensar los problemas, declarando como variables o los tipos de datos los objetos del problema, y que a su vez, cada objeto tiene anidadas variables que hacen referencia al dato.  -Es más moderna, es una evolución.  -Incorpora polimorfismo y el envío de mensajes entre los objetos.  -Resuelve el problema identificando los actores que tienen participación e identificando sus acciones.  -Está orientada a objetos  -Consta de: ANÁLISIS, DISEÑO, EVOLUCIÓN Y MODIFICACIÓN. |

**Lenguajes de Programación Orientada a Objetos**

En esta sección se deberá focalizar en las diferencias entre los diversos lenguajes de programación que soportan el paradigma de la programación orientada a objetos.

**C++**

Incluir código de como implementar:

**Una clase**

#include <iostream>

#include "Fecha.h"

using namespace std;

Fecha::Fecha(unsigned int d, unsigned int m, int a)

{

   dia = d;

   mes = m;

   año = a;

}

unsigned int Fecha::ganDia()

{

   return dia;

}

unsigned int Fecha::getMes()

{

   return mes;

}

int Fecha::gettAno()

{

   return año;

}

void Fecha::imprimir Fecha()

{

   cout<<"\n"<<dia<<"/"<<mes<<"/"<<anho;

}

void Fecha::cambiar Fecha(unsigned int d, unsigned int m, int a)

{

   if(es Correcta(d,m,a))

   {

       dia = d;

       mes = m;

       año = a;

   }

   else

   {

       cout<<"\nLa fecha introducida es incorrecta. Datos no modificados.";

   }

}

bool es Correcta(unsigned int d, unsigned int m, int a)

{

   unsigned int array Días Mes[12] = {31,29,31,30,31,30,31,31,30,31,30,31};

   if(a==0) return false; //Aporte de susmel en comentarios.

   if(d<=0||d>31) return false;

   if(m<=0||m>12) return false;

   if(m==2 && d>29) return false;

   if(array Días Mes[m-1]<d) return false;

   if(m==2 && d==29 && !es Bisiesto(a)) return false;

   return true;

}

bool Bisiesto(int a)

{

   if ((a%4==0)||(a%400==0)&&(a%100!=0)) return true;

   else return false;

}

**Herencia**

#include <iostream>

#include <cstring>

using namespace std;

//definiciÃ³n de la clase Número

class persona

{

public:

 persona(char[40] = "", int = 1);

 void establecerEdad(int);

 void mostrarEdad(void);

 void establecer Nombre(char[40]);

 void mostrar Nombre(void);

private:

 int edad;

 char nombre[40];

};

class asalariado : public persona {

public:

 asalariado(char[40] = "" , int = 1, double = 0);

 void establecer Sueldo(double = 0);

 void mostrar Sueldo(void);

private:

 double sueldo;

};

class obrero : public asalariado {

public:

 obrero(char[40] = "" , int = 1, double = 0, char[40] = "Funciones Generales");

 void establecer Cargo(char[40]);

 void mostrar Cargo(void);

private:

 char cargo[40];

};

class médico : public asalariado {

public:

 médico(char[40] = "" , int = 1, double = 0, int = 0);

 void establecer Cant Pacientes(int);

 void mostrar Cant Pacientes(void);

private:

 int cant pacientes;

};

persona::persona(char nom[40], int ed){

 establecer Edad(ed);

 establecer Nombre(nom);

}

void persona::establecer Edad(int ed) {edad = ed;}

void persona::mostrar Edad(void){ cout << "Edad: "<< edad << endl;}

void persona::establecer Nombre(char nom[40]){strcpy (nombre,nom);}

void persona::mostrar Nombre(void){cout << "Nombre: " << nombre << endl;}

asalariado::asalariado(char nom[40] , int ed, double suel){

 establecer Edad(ed);

 establecer Nombre(nom);

 establecer Sueldo(suel);

}

void asalariado::establecer Sueldo(double sueldo){sueldo = sueldo; }

void asalariado::mostrar Sueldo(void){ cout << "Sueldo: " << sueldo << endl;}

obrero::obrero(char nom[40] , int ed, double suel, char car[40]){

 establecer Edad(ed);

 establecer Nombre(nom);

 establecer Sueldo(suel);

 establecer Cargo(car);

}

void obrero::establecer Cargo(char car[40]) {strcpy (cargo,car);}

void obrero::mostrar Cargo(void) { cout << "Cargo: " << cargo << endl; }

medico::medico(char nom[40] , int ed, double suel, int cant){

 establecer Edad(ed);

 establecer Nombre(nom);

 establecer Sueldo(suel);

 establecer Cant Pacientes(cant);

}

void médico::establecer Cant Pacientes(int cant) { cant pacientes = cant; }

void medico::mostrarCantPacientes(void) { cout << "Cantidad de pacientes: " << cantpacientes << endl; }

//Probamos la clase.

int main()

{

 médico pedro("Pedro Gonzales", 40, 5457.45, 15);

 cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

 pedro.mostrar Nombre();

 pedro.mostrar Edad();

 pedro.mostrar Sueldo();

 pedro.mostrar Cant Pacientes();

 cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

 obrero jose("Jose Alvarez", 54, 1200.32,  "Tornero Calificado");

 jose.mostrar Nombre();

 jose.mostrar Edad();

 jose.mostrar Sueldo();

 jose.mostrar Cargo();

 cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

 médico juan;

 juan.mostrar Nombre();

 juan.mostrar Edad();

 juan.mostrar Sueldo();

 juan.mostrar Cant Pacientes();

 return 0;

}

Polimorfismo

*#include <iostream>*

*#include <string>*

using namespace std;

class Mascota

{

public:

 Mascota(string nombre, int patas);

 string hablar();

private:

 virtual string palabra();

 string nombre;

 int patas;

};

Mascota::Mascota(string nombre, int patas):nombre(nombre),patas(patas)

{

 cout << "Acaba de nacer tu mascota "<<nombre<<" con "<<patas<<"patas"<<endl;

}

string Mascota::palabra()

{

 return "BUUU";

}

string Mascota::hablar()

{

 return nombre" dice: "+this->palabra();

}

class Perro : public  Mascota

{

public:

 Perro(string nombre);

 string palabra();

};

Perro::Perro(string nombre):Mascota(nombre, 4)

{

}

string Perro::palabra()

{

 return "GUAU";

}

class Gato : public  Mascota

{

public:

 Gato(string nombre);

 string palabra();

};

Gato::Gato(string nombre):Mascota(nombre, 4)

{

}

string Gato::palabra()

{

 return "MIAU";

}

int main()

{

 Mascota \*m, \*g;

 m=new Perro("Ramiro");

 g=new Gato("Lancelot");

 cout << m->hablar() << endl;

 cout << g->hablar() << endl;

 return 0;

}

Encapsulamiento

class Fecha

{

  public:

  void tomaFecha (int ancho, int mes, int dia);

  int dameAnho ();

  int dameMes ();

  int dameDia ();

  void metodo Maravilloso 1();

  void metodo Maravilloso 2();

  protected:

  int anho; // El año con cuatro cifras, ej. 2004

  int mes;  // El mes, de 1 a 12

  int dia; // El dia, de 1 a 31

};

**Java**

***Incluir código de como implementar:***

Una clase

/\* Un programa que usa la clase Vehículo

El archivo se llama DemoVehiculo.java

\*/

class Vehículo {

   int pasajeros;  //números de pasajeros

   int capacidad;  //capacidad del combustible en galones

   int mpg;        //combustible consumido en millas por galón

}

//Esta clase declara un objeto de tipo Vehiculo

class Demo Vehículo {

   public static void main(String[] args) {

       Vehículo minivan = new Vehiculo();

       int rango;

       //asignando valores a los campos de minivan

       minivan.pasajeros = 9;

       minivan.capacidad = 15;

       minivan.mpg = 20;

       //Calcular el rango asumiendo un tanque lleno

       rango = minivan.capacidad \* minivan.mpg;

       System.out.println("La Minivan puede llevar " + minivan.pasajeros + " pasajeros con un rango de " + rango + " millas");

   }

}

**Herencia**

//Clase para objetos de dos dimensiones

class Dos Dimensiones{

   double base;

   double altura;

   void mostrar Dimensión(){

       System.out.println("La base y altura es: "+base" y "+altura);

   }

}

//Una subclase de Dos Dimensiones para Triángulo

class Triangulo extends Dos Dimensiones{

   String estilo;

   double area(){

       return base\*altura/2;

   }

   void mostrar Estilo(){

       System.out.println("Triángulo es: "+estilo);

   }

}

class Lados 3{

   public static void main(String[] args) {

       Triángulo t1=new Triangulo();

       Triángulo t2=new Triangulo();

       t1.base=4.0;

       t1.altura=4.0;

       t1.estilo="Estilo 1";

       t2.base=8.0;

       t2.altura=12.0;

       t2.estilo="Estilo 2";

       System.out.println("Información para T1: ");

       t1.mostrar Estilo();

       t1.mostrar Dimensión();

       System.out.println("Su área es: "+t1.area());

       System.out.println();

       System.out.println("Información para T2: ");

       t2.mostrar Estilo();

       t2.mostrar Dimensión();

       System.out.println("Su área es: "+t2.área());

   }

}

**Polimorfismo**

public class Polimorfismo {

public static void main(String[] args) {

*// Creamos una variable del tipo Mi ClaseA, que será un array de 3 elementos*

MiClaseA [] misClases=new MiClaseA[3];

misClases[0]=new ClaseA("Esther");

*// Asignamos a la variable misClases que son del tipo Mi ClaseA un objeto del*

*// tipo MiClase, ya que hereda de Mi ClaseA*

misClases[1]=new MiClase("Juan", "Azul");

misClases[2]=new ClaseA("Rosa");

for(Mi ClaseA e:misClases) {

*// ejecutará la función info() de la clase que haya sido instanciada.*

*// Esto se llama polimorfismo*

System.out.println(e.info());

}

}

}

class ClaseA {

private String name;

*// constructor*

public Class(String name) {

this.name=name;

}

public String info() {

*// Devolvemos el nombre*

return this.name;

}

}

class MiClaseB extends MiClaseA {

private String color;

*// constructor*

public MiClaseB(String name, String color) {

*// ejecutamos el constructor de la superclase (Mi ClaseA) enviando el nombre*

super(name);

*// guardamos el nombre del color en la variable color de Mi Clase*

this.color=color;

}

public String info() {

*// devolvemos el contenido de Mi ClaseA.info() más la variable color*

*// de la clase MiClase*

return super.info() + " (" + this.color + ")";

}

}

**Encapsulamiento**

public class Persona{

                                           //Atributos

                                           private int altura;

                                           //Métodos

                                           public int get Altura(){

                                           return this.Altura;

                                           }

                                           public void setAltura(int una Altura){

                                           this.altura = una Altura;

                                           }

                        }

Python

Incluir codigo de como implementar:

Una clase

|  |  |
| --- | --- |
| #!/usr/bin/env python3 |  |
|  | # -\*- coding: utf-8 -\*- |
|  | """ |
|  | #Created on Thu Nov  1 17:03:14 2018 |
|  |  |
|  | @author: hsantanam |
|  | """ |
|  | #TSB - Create Class in Python - rocket positions (x,y) |
|  | import matplotlib.pyplot as plt |
|  |  |
|  | class Rocket(): |
|  | def \_\_init\_\_(self, x=0, y=0): |
|  | #each rocket has (x,y) position; user or calling function has choice |
|  | #of passing in x and y values, or by default they are set at 0 |
|  | self.x = x |
|  | self.y = y |
|  |  |
|  | def move\_up(self): |
|  | self.y += 1 |
|  |  |
|  | def move\_down(self): |
|  | self.y -= 1 |
|  |  |
|  | def move\_right(self): |
|  | self.x += 1 |
|  |  |
|  | def move\_left(self): |
|  | self.x -= 1 |
|  |  |
|  |  |
|  | #Make a series of rockets |
|  | rockets=[] |
|  | rockets.append(Rocket()) |
|  | rockets.append(Rocket(0,2)) |
|  | rockets.append(Rocket(1,4)) |
|  | rockets.append(Rocket(2,6)) |
|  | rockets.append(Rocket(3,7)) |
|  | rockets.append(Rocket(5,9)) |
|  | rockets.append(Rocket(8, 15)) |
|  |  |
|  | #Show where each rocket is |
|  |  |
|  | for index, rocket in enumerate(rockets): |
|  | #original position of rockets |
|  | print("Rocket %d is at (%d, %d)." % (index, rocket.x, rocket.y)) |
|  | plt.plot(rocket.x, rocket.y, 'ro', linewidth=2, linestyle='dashed', markersize=12) |
|  | #move the 'rocket' one up |
|  | rocket.move\_up() |
|  | print("New Rocket position %d is at (%d, %d)." % (index, rocket.x, rocket.y)) |
|  | #plot the new position |
|  | plt.plot(rocket.x, rocket.y, 'bo', linewidth=2, linestyle='dashed', markersize=12) |
|  | #move the rocket left |
|  | rocket.move\_left() |
|  | plt.plot(rocket.x, rocket.y, 'yo', linewidth=2, linestyle='dashed', markersize=12) |
|  |  |
|  | #show graph legend to match colors with position |
|  | plt.title("Positions from using Python Class") |
|  | plt.gca().legend(('original position','^ - Moved up', '< - Moved left')) |
|  | plt.show() |
|  | #plt.legend(loc='upper left') |

Herencia

class Polygon:

   def \_\_init\_\_(self, no\_of\_sides):

       self.n = no\_of\_sides

       self.sides = [0 for i in range(no\_of\_sides)]

   def inputSides(self):

       self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in range(self.n)]

   def dispSides(self):

       for i in range(self.n):

           print("Side",i+1,"is",self.sides[i])

class Triangle(Polygon):

   def \_\_init\_\_(self):

       Polygon.\_\_init\_\_(self,3)

   def findArea(self):

       a, b, c = self.sides

       # calculate the semi-perimeter

       s =(a + b + c) / 2

       area =(s\*(s-a)\*(s-b)\*(s-c)) \*\* 0.5

       print('El area del triangulo es %0.2f' %area)

>>> t = Triangle()

>>> t.input Sides()

Enter slide 1 : 3

Enter slide 2 : 5

Enter slide 3 : 4

>>> t.disp Sides()

Side 1 is 3.0

Side 2 is 5.0

Side 3 is 4.0

>>> t.findArea()

El área del triángulo es 6.00

Polimorfismo

def frecuencias(secuencia):

   """ Calcula las frecuencias de aparición de los elementos de

       la secuencia recibida.

       Devuelve un diccionario con elementos: {valor: frecuencia}

   """

*# crea un diccionario vacío*

   frec = dict()

*# recorre la secuencia*

   for elemento in secuencia:

       frec[elemento] = frec.get(elemento, 0) + 1

   return frec

>>> frecuencias(["peras", "manzanas", "peras", "manzanas", "uvas"])

{'uvas': 1, 'peras': 2, 'manzanas': 2}

>>> frecuencias((1,3,4,2,3,1))

{1: 2, 2: 1, 3: 2, 4: 1}

>>> frecuencias("Una frase")

{'a': 2, ' ': 1, 'e': 1, 'f': 1, 'n': 1, 's': 1, 'r': 1, 'U': 1}

>>> ran = xrange(3, 10, 2)

>>> frecuencias(ran)

{9: 1, 3: 1, 5: 1, 7: 1}

>>> frecuencias(4)

Traceback (most recent call last):

   File "<pyshell\#0>", line 1, in <module>

       frecuencias(4)

   File "frecuencias.py", line 12, in frecuencias

       for v in seq:

   TypeError: 'int' object is not iterable

Encapsulamiento

#!/usr/bin/env python

# Nombre de Fichero : encapsulacion.py

class Persona(object) :

   "Clase Persona"

   def \_\_init\_\_(self, Nombre,Edad,Sueldo) :

       self.set Nombre(peNombre)

       self.setEdad(pEdad)

       self.\_\_set Sueldo(Sueldo);

   def setEdad(self, piEdad) :

       self.\_\_ edad = pEdad

   def getEdad(self) :

       return self.\_\_ edad

   def setNombre(self, Nombre) :

       self.\_\_ nombre = Nombre

   def getNombre(self) :

       return self.nombre

   def \_\_setSueldo(self,Sueldo):

       self.\_\_ sueldo = Sueldo

   def getSueldo(self):

       return self.\_\_ sueldo

   nombre = property(getNombre, setNombre)

   edad = property(getEdad, sequEdad)

class Gerente(Persona) :

   def \_\_init\_\_(self, Nombre,Edad):

       Persona.\_\_init\_\_(self, Nombre,Edad,5000)

class Secretaria(Persona) :

   def \_\_init\_\_(self, Nombre,Edad) :

       Persona.\_\_init\_\_(self, Nombre,Edad,500)

if \_\_name\_\_ == '\_\_main\_\_':

   g = Gerente("Mariano", 56)

   s = Secretaria("Rocío", 33)

   print "El Gerente es", g.nombre , " gana ", g.get Sueldo()

   print "La Secretaria es ", s.getNombre(), " gana ",s.get Sueldo()

C#

Incluir codigo de como implementar:

Una clase

public class Puerta

{

   int ancho;     *// Ancho en centímetros*

   int alto;      *// Alto en centímetros*

   int color;     *// Color en formato RGB*

   bool abierta;  *// Abierta o cerrada*

   public void Abrir()

   {

       abierta = true;

   }

   public void Cerrar()

   {

       abierta = false;

   }

   public void Mostrar Estado()

   {

       Console.WriteLine("Ancho: {0}", ancho);

       Console.WriteLine("Alto: {0}", alto);

       Console.WriteLine("Color: {0}", color);

       Console.WriteLine("Abierta: {0}", abierta);

   }

} *// Final de la clase Puerta*

Herencia

using System;

public class A

{

  private int value = 10;

  public class B : A

  {

      public int GetValue()

      {

          return this.value;

      }

  }

}

public class C : A

{

//    public int GetValue()

//    {

//        return this.value;

//    }

}

public class Example

{

   public static void Main(string[] args)

   {

       var b = new A.B();

       Console.WriteLine(b.GetValue());

   }

}

// The example displays the following output:

//       10

Polimorfismo

using System;

using System.Collections.Generic;

public class Shape

{

   // A few example members

   public int X { get; private set; }

   public int Y { get; private set; }

   public int Height { get; set; }

   public int Width { get; set; }

   // Virtual method

   public virtual void Draw()

   {

       Console.WriteLine("Performing base class drawing tasks");

   }

}

class Circle : Shape

{

   public override void Draw()

   {

       // Code to draw a circle...

       Console.WriteLine("Drawing a circle");

       base.Draw();

   }

}

class Rectangle : Shape

{

   public override void Draw()

   {

       // Code to draw a rectangle...

       Console.WriteLine("Drawing a rectangle");

       base.Draw();

   }

}

class Triangle : Shape

{

   public override void Draw()

   {

       // Code to draw a triangle...

       Console.WriteLine("Drawing a triangle");

       base.Draw();

   }

}

class Program

{

   static void Main(string[] args)

   {

       // Polymorphism at work #1: a Rectangle, Triangle and Circle

       // can all be used wherever a Shape is expected. No cast is

       // required because an implicit conversion exists from a derived

       // class to its base class.

       var shapes = new List<Shape>

       {

           new Rectangle(),

           new Triangle(),

           new Circle()

       };

       // Polymorphism at work #2: the virtual method Draw is

       // invoked on each of the derived classes, not the base class.

       foreach (var shape in shapes)

       {

           shape.Draw();

       }

       // Keep the console open in debug mode.

       Console.WriteLine("Press any key to exit.");

       Console.ReadKey();

   }

}

/\* Output:

   Drawing a rectangle

   Performing base class drawing tasks

   Drawing a triangle

   Performing base class drawing tasks

   Drawing a circle

   Performing base class drawing tasks

\*/

Encapsulamiento

///Francys System

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Francys System Encapsulamiento

{

   class Program

   {

       class Salon

       {

           private string nombre = "Omar";

           public string Nombre

           {

               get { return nombre; }

               set { nombre = value; }

           }

           public string nombres = "Carlos";

       }

       class MainClass

       {

           public static void Main()

           {

               Salon nuevo salon = new Salon();

               Console.WriteLine("Encapsulación  en c#");

               Console.WriteLine("Nombre uno : {0}", nuevo salon.nombres); ///Variable Pública

               Console.WriteLine("Nombre dos : {0}", nuevo salon.Nombre); ///Encapsulación

               Console.WriteLine("\n\nfrancysystem.blogspot.com");

               Console.ReadLine();

           }

       }

   }

}

**Conclusiones**

Puedo concluir al final de lo visto e investigado en este documento ,que en los distintos lenguajes de programación hay bastante diferencia en cuestión a líneas de código, es decir, mientras que por ejemplo en python utilizamos dos líneas de código para un hello world, en un c++ o un java se usan cuatro o cinco líneas, esto va dependiendo también el código que se vaya a realizar, también se puede apreciar que hay algunos lenguajes de programación utilizamos objetos ,herencias y encapsulamiento entre otros.

Podemos ver que los distintos lenguajes de programación han evolucionado bastante a partir de su creación y cada vez son más demandantes en la industria, sobre todo el lenguaje java.

# Bibliografía

*Codigo Fuente*. (2018). Obtenido de Polimorfismo en C#: https://codigofuentenet.wordpress.com/2012/07/17/polimorfismo-en-c/

Diaz, C. (s.f.). *Encapsulacion en C#*. Obtenido de Francys Systems: https://francysystem.blogspot.com/2015/02/encapsulacion-en-c.html

Fuente, C. (28 de Octubre de 2018). *Herencia en Python-Simple, Multiple y Multinivel*. Obtenido de https://www.codigofuente.org/herencia-en-python/

informatica, L. r. (2015). Obtenido de Lenguaje de programacion C#: http://www.larevistainformatica.com/C1.htm

IT, E. (21 de Mayo de 2018). *EDUCACION IT*. Recuperado el 07 de Marzo de 2019, de https://blog.educacionit.com/2018/05/21/programacion-orientada-a-objetos-vs-programacion-estructurada/

petrush, R. (s.f.). *Microsoft*. Obtenido de Inheritance in C# and. Net: https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/inheritance

programacion, T. (2013). *Historia de Java*. Obtenido de http://www.tuprogramacion.com/programacion/historia-de-java/

*System.OutOfMemomryException*. (18 de Mayo de 2015). Obtenido de Un poquito de historia de C#: https://sparraguerra.wordpress.com/2015/05/18/un-poquito-de-historia-de-c/

Tecnologiapreu. (14 de agosto de 2017). *Programacion Orientada a Objetos vs Programacion Estructurada*. Obtenido de https://tecnologiapreu.wordpress.com/2017/08/14/programacion-orientada-a-objetos-vs-programacion-estructurada/