Python

1. Introducción

Python es un lenguaje interpretado orientado a objetos. En un lenguaje interpretado, el ordenador interpreta el lenguaje al ejecutar el programa, mientras que, en un lenguaje compilado, previamente lo ha convertido a código máquina, por lo que será más rápido, pero depende más de la máquina.

Python es un lenguaje de tipo dinámico. Esto significa que no hay que decir el tipo de variable que hacemos, a diferencia de otros lenguajes como Pascal, Java o C a los que hay que especificar si las variables son texto, números etc. Python está estructurado en objetos y clases que se incluyen en los scripts para que nos sea más fácil su código.

2. Empezando. Primer programa

Para instalar Python, descarga la última versión desde *www.python.org/download/*

El primer programa:

Existen **dos** formas de ejecutar código en Python:

1ª. Escribir líneas de código en la consola, una a una.

La consola la puedes ejecutar desde el programa Python.exe o desde el Shell Python GUI (interface del Usuario)

Ejercicio 1: Tras el símbolo >>> escribe : `print ("hola mundo")` y pulsa ↵

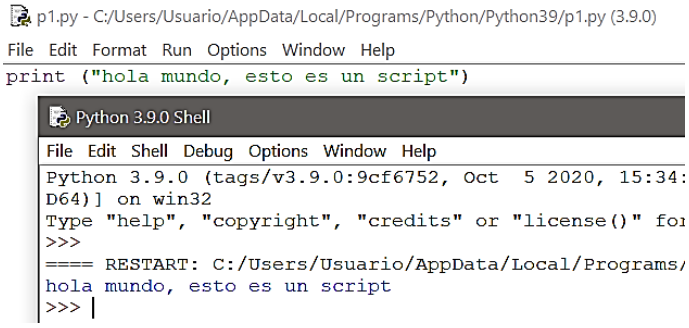
2ª. Ejecutar un archivo de texto .py llamado *script*.

Ejercicio 2: Crea un archivo desde el bloc de notas o desde el GUI (***File*** – ***New file***) con el nombre " *script1*.py".

- Escribe: `print ("hola mundo, esto es un script")` – Guárdalo en tu carpeta con el nombre: *script1.py*
- Desde la consola puedes escribir: Python c:\Python***script1**.py

- Desde el *Shell* ***IDLE*** (*Integrated Development Environment for Python*) es el entorno de desarrollo que permite editar y ejecutar los programas.

  Escoge: ***File – Open*** – Se abrirá el archivo en una ventana aparte, donde hay un menú con el elemento: *Run*.
- Para probar/ejecutar el script: Escoge del menú: ***Run*** – ***Run module*** o F5.

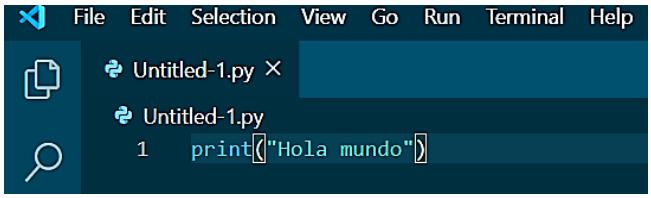


- Utiliza un IDE o editor de terceros como, por ejemplo: ***Visual Studio Code***:

  Una vez instalado Visual Studio Code, debes añadir el depurador de *Python* desde el menú: *Run – Install additional debuggers…*

  Para ejecutar el archivo, menú:
  Run – Start debugging – Currently Python file.

  Verás el resultado en el panel *Terminal – Consola* inferior.



Tipos y variables

En *Python*, los tipos básicos se dividen en

1. **Números:** se dividen en enteros (int 5), números de coma flotante (float 5.53), o complejos (7+5i). Recuerda que, en programación, la coma decimal española se convierte en un punto.
2. **Cadenas** de texto o strings, van siempre entre comillas: c = "hola mundo"
3. **Boolean** o lógicas (verdadero (true)/falso (false))

Como vemos, en *Python* no hay que indicar de qué tipo de variable se trata.

Comentarios: Los comentarios en *Python* van precedidos del signo # (en C o Delphi van con //)

En Python se pueden sumar o multiplicar variables de texto y numéricas pero no juntas a la vez.

Por ejemplo: meses = edad *12 o Nombrecompleto = nombre + apellidos

Ejercicio 3:

```
# script 3 de Python
# Voy a crear una variable de texto o cadena
nombre = "Alberto"
# Ahora voy a escribir una variables numérica:
edad=18
meses=edad*12
print ("Hola, " + nombre)
print (meses)
print ("tienes " + str(meses) + " meses")
```

3. Entrada de usuario input

Pedir un dato al usuario: ***input()*** y ***raw_input()*** Para obtener lo que el usuario escriba en pantalla.
La variable entrada contendrá lo que el usuario escribió hasta pulsar Enter.

Ejercicio script4 (versión 3.X):

```
print("¿Cómo te llamas?")
nombre = input()  # nombre es la variable que recoge la entrada de texto
print("Me alegro de conocerte,", nombre)
```

De forma predeterminada, la función input() convierte la entrada en una cadena.

- Entrada de un *número entero*: se debe utilizar la función int() de la siguiente manera:

```
cantidad = int(input("Dígame una cantidad en euros: "))
print(cantidad, "euros en dólares son ", cantidad / 1.25, "dólares")
```

- Entrada de *número decimal*: se debe utilizar la función float() de la siguiente manera:

```
cantidad = float(input("Dígame una cantidad en euros (hasta con 2 decimales): "))
print(cantidad, "euros son", cantidad * 166.386, "pesetas")
```

Si se quiere que los dos puntos (:) salgan pegados al nombre habría que utilizar la concatenación (operador +):

```
nombre = input("Dígame su nombre: ")
print("Dígame su apellido,", nombre + ": ", end="")
apellido = input()
print("Me alegro de conocerle,", nombre, apellido)
```

```
Dígame su nombre:   Armando
Dígame su apellido, Armando:   Guerra
Me alegro de conocerle, Armando Guerra
```

Pero también se puede incluir la pregunta en la recogida de la respuesta, utilizando la concatenación (operador +):

4. Variables lógicas y operadores

Las *boolean* de verdadero y falso son útiles para las expresiones condicionales (**if**) y los **while** (bucle).

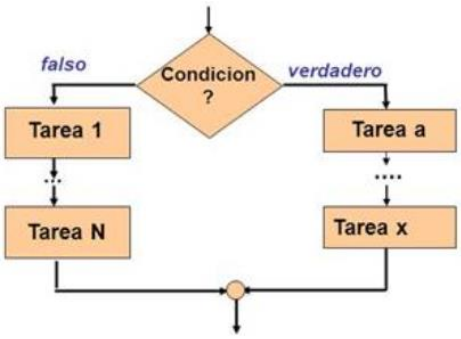
Los operadores lógicos Sirven para trabajar con boolean y números en expresiones condicionales.

```
Igual: ==; Mayor: > , menor<  , mayor o igual que >= , menor o igual que <=, distinto: != ; and se
cumple a y b -  or: se cumple a o b;  not:  not True es False
```

5. Condicionales.

Los condicionales permiten comprobar condiciones y hacer que nuestro programa ejecute un fragmento de código u otro dependiendo de la condición.

If: if (del inglés si) seguido de la condición a evaluar, dos puntos (:) y en la siguiente línea y sangrado, el código a ejecutar en caso de que se cumpla dicha condición:



Ejercicio script5

```
favorito = "ofimega.es"
if favorito == "ofimega.es":   # si favorito es "ofimega.es"
     print ("Tienes buen gusto!")
     print ("Gracias")
```

Importante las tabulaciones:

En Python, el código tabulado (sangrado o *indentado*) se utiliza para designar un bloque. No hace falta símbolos para agrupar un bloque { … } como en C o Java, porque Python busca si hay espacios antes del código y considera que están dentro de otro bloque.

else: if … **else**: se utiliza en el caso de que la condición no se cumpla

Ejercicio script6

```
gusto = ""
print ("Crees que Edu es guapo?")
gusto = input("> ")
if gusto == "si":
    print ("Tienes buen gusto")
else:
    print ("Tienes el gusto estropeado")
```

¿Qué pasa si el usuario contesta "SI" o "sí" con acento? Para que acepte varias condiciones a la vez utilizamos el operador lógico OR:

Amplía el ejercicio:

Puedes poner el texto de la pregunta dentro del *input* pero si la variable es numérica debes convertir antes el input a *integer*.


```
#Ejercicio script7
nombre = input ("Cómo te llamas?: ")          # pregunta el nombre
print ("Me alegro de conocerle,", nombre)
edad = int(input ("Cuántos años tienes?: ")) # pregunta la edad y la convierte a numérica
if edad >=18:                                 # evalúa si se cumple la condición
       print ("Oye, "+nombre+" con "+ str(edad)+ " ya eres viejo")
else:
       print ("Oye, "+nombre+" con "+ str(edad)+ " todavía eres joven")
```

#Ejercicio script7 mejorado

```
print ("Hola, bienvenido")
nombre=input("¿Cómo te llamas?:  ")
print("Encantado, ", nombre)
anos=int(input("cuantos años tienes?  "))
if anos>65:
   print ("Pues con "+str(anos)+" años, ya eres jubilado")
elif anos>18:
   print ("Pues con "+str(anos)+" años, ya eres mayor de edad")
else:
   print ("Pues con "+str(anos)+" años, aún eres menor de edad")
```

elif.(Y si… else if)

elif es una contracción de else if, por lo tanto elif puede leerse como "y si". Es decir, primero se evalúa la condición del if. Si es cierta, se ejecuta su código y se continúa ejecutando el código posterior al condicional; si no se cumple, se evalúa la condición del elif. Si se cumple la condición del elif se ejecuta su código y se continúa ejecutando el código posterior al condicional; si no se cumple y hay más de un elif se continúa con el siguiente en orden de aparición.

```
if numero < 0:
       print "Negativo"
elif numero > 0:
       print "Positivo"
else:
       print "Cero"
```

Si no se cumple la condición del if ni de ninguno de los elif, se ejecuta el código del else. (en cualquier otro caso)

A **if** C **else** B

También existe una construcción similar al operador ? de otros lenguajes, que no es más que una forma compacta de expresar un if else. En esta construcción se evalúa el predicado C y se devuelve A si se cumple o B si no se cumple: A if C else B. Veamos un ejemplo:

```
var = "par" if (num % 2 == 0) else "impar"
```

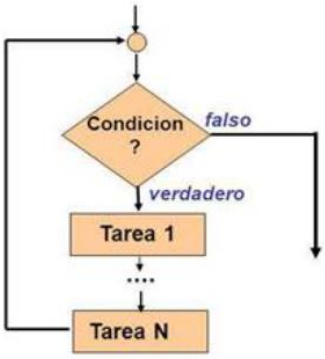
switch o case: En Python no existe esta construcción, que podría emularse con un simple diccionario.

6. Bucles

Mientras que los condicionales nos permiten ejecutar distintos fragmentos de código dependiendo de ciertas condiciones, los bucles nos permiten ejecutar un mismo fragmento de código un cierto número de veces, mientras se cumpla una determinada condición.

While bucle infinito

El bucle while (mientras) ejecuta un fragmento de código mientras se cumpla una condición.



Pide clave (modo 1)

```
#Ejercicio script8
clave = input("Introduzca la clave: ")
while clave != "ofimega":
       clave = input("clave incorrecta, repita de nuevo: ")
print ("correcto")
```

Esto es lo que se conoce como un bucle infinito. Este bucle se podría haber escrito también, de la forma:

Pide clave (modo 2)

```
while True:
    clave = input("introduzca la clave: ")
    if clave == "ofimega":
        print ("correcto")
        break
    else:
        print (clave + " no es una clave correcta. Pruebe de nuevo." )
```

La palabra clave break (romper) sale del bucle en el que estamos.

Tabla de multiplicar

```
print("La tabla de multiplicar del 5:")
NUMERO=5   #variable NUMERO como constante
i = 1
while i <= 10:
    resultado=NUMERO*i;
    print(str(NUMERO) + " por " + str(i) +" = " + str(resultado))
    i=i+1    #incrementamos su valor
print("Programa terminado")
```

Usamos la variable NUMERO como una **constante**: simplemente escribimos el nombre en mayúsculas para dar a entender que esa variable no debe cambiar.

For bucle (finito)

```
secuencia = ["uno", "dos", "tres"]
for elemento in secuencia:
print (elemento)
```

- los ***for*** se utilizan para recorrer secuencias: para cada elemento que tengamos en la secuencia, ejecuta sus líneas de código.

- *range* (rango) es una secuencia definida desde el primer al último valor

Ejercicio con sus variantes:

```
secuencia = [1, 2, 3]
for oportunidad in secuencia:
    print ("oportunidad " + str(oportunidad))
    clave=input("clave: ")
    if clave=="ofimega":
        print("Clave correcta")
        break
    print("Clave incorrecta")
print("Fin")
```

Explicación

Lo que hace la cabecera del bucle es obtener el siguiente elemento de la secuencia y almacenarlo en la variable oportunidad, que irá cambiando en cada ciclo.

Ejercicio variante 2: Cambia el principio del código

```
for x in range(1,4);
    print ("oportunidad " + str(x))
clave=input("clave: ")
       . . .
```

Python proporciona una función llamada ***range*** (rango) que permite generar una lista que vaya desde el primer número que le indiquemos al segundo.:

range(primero, último)

Ejercicio variante 3: Cambia el principio del código

```
for x in range(3);
print("tienes ", str( 3-x), " oportunidades"
clave=input("clave: ")
       . . .
```

Si no se especifica el primero, empieza a contar desde cero el número de veces:

range(veces)

Ejemplos:

```
anchura = int(input("Anchura del rectángulo: "))
altura = int(input("Altura del rectángulo: "))

for i in range(altura):
    for j in range(anchura):
        print("* ", end="")
    print()
```

Programa que pida la anchura y altura de un rectángulo y lo dibuje con caracteres *

```
anchura = int(input("Anchura del triángulo: "))

for i in range(1, anchura + 1):
    for j in range(i):
        print("* ", end="")
    print()

for i in range(1, anchura):
    for j in range(anchura - i):
        print("* ", end="")
    print()
```

Programa que pida la anchura de un triángulo y lo dibuje con caracteres *

Ejercicio propuesto: Calcule el factorial de 5

Ejercicio factorial usando while

```
# Factorial con while
x=int(input("Introduzca numero
factorial: "))
y=x
while x>1:
    x=x-1
    y = y* x
print("Resultado: ", str(y))
```

Ejercicio de potencia usando For

```
# potencia con for
x = int(input("Introduzca la base: "))
y = int(input("Introduzca el exponente: "))
z = x
for i in range(1,y):
    z = z* x
print("Resultado: ", str(z))
```

7. Funciones

Una función es un fragmento de código con un nombre asociado que realiza una serie de tareas y devuelve un valor. A los fragmentos de código que tienen un nombre asociado y no devuelven valores se les suele llamar procedimientos. En Python no existen los procedimientos, ya que cuando el programador no especifica un valor de retorno la función devuelve el valor None (nada), equivalente al null de Java.

Ejercicio de acción o subrutina:

```
def licencia():
    print("Copyright 2013 ofimega")
    print("Licencia CC-BY-SA 3.0")
    return
print("Créditos del programa:")
licencia()
```

```
Créditos del programa:
Copyright 2013 ofimega
Licencia CC-BY-SA 3.0
```

Además de ayudarnos a programar y depurar dividiendo el programa en partes, las funciones también permiten reutilizar código. En Python las funciones se declaran de la siguiente forma:

```
def mi_funcion(param1, param2):
print param1
print param2
```

Es decir, la palabra clave **def** seguida del nombre de la función y entre paréntesis los argumentos separados por comas. A continuación, en otra línea, tabulado y después de los dos puntos tendríamos las líneas de código que conforman el código a ejecutar por la función.

Ejercicio de Procedimiento: Recoge parámetros pero no devuelve nada

```
def escribe_media(x, y):
    media = (x + y) / 2
    print("La media de", x, "y", y, "es:", media)
    return

a = 3
b = 5
escribe_media(a, b)
```

```
La media de 3 y 5 es: 4.0
```

El número de valores que se pasan como parámetro al llamar a la función tiene que coincidir con el número de parámetros que la función acepta según la declaración de la función. En caso contrario Python se quejará:

Ejercicio de Función: recoge parámetros y devuelve un valor con Return

```
def calcula_media(x, y):
    resultado = (x + y) / 2
    return resultado

a = 3
b = 5
media = calcula_media(a, b)
print("La media de", a, "y", b, "es:", media)
```

```
La media de 3 y 5 es: 4.0
```

También podemos encontrarnos con una cadena de texto como primera línea del cuerpo de la función. Estas cadenas se conocen con el nombre de *docstring* (cadena de documentación) y sirven, como su nombre indica, a modo de documentación de la función.

```
def mi_funcion(param1, param2):
"""Esta funcion imprime los dos valores pasados como parametros"""
print param1
print param2
```

Al declarar la función lo único que hacemos es asociar un nombre al fragmento de código que conforma la función, de forma que podamos ejecutar dicho código más tarde referenciándolo por su nombre. Es decir, a la hora de escribir estas líneas no se ejecuta la función. Para llamar a la función (ejecutar su código) se escribiría:

```
mi_funcion("hola", 2)
```

Los valores por defecto para los parámetros se definen situando un signo igual después del nombre del parámetro y a continuación el valor por defecto:

```
def imprimir(texto, veces = 1):
print veces * texto
```

Para definir funciones con un número variable de argumentos colocamos un último parámetro para la función cuyo nombre debe precederse de un signo *****:

```
def varios(param1, param2, *otros):
for val in otros:
print val
varios(1, 2)
varios(1, 2, 3)
varios(1, 2, 3, 4)
```

Los argumentos de la función son variables locales a la función que contienen los valores. Es decir, en realidad lo que se le pasa a la función son copias de los valores y no las variables en sí.

Ejercicio propuesto: función que calcule el factorial de cualquier número y devuelva su resultado.

8. Listas o arrays

Conocidas también como arreglos o matrices unidimensionales en otros lenguajes, las listas son conjuntos ordenados de elementos (números, cadenas, listas, etc). Las listas se delimitan por corchetes ([ ]) y los elementos se separan por comas.

Las listas pueden contener elementos del mismo tipo:

```
>>> primos = [2, 3, 5, 7, 11, 13]
>>> diasLaborables = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes"]
```

O pueden contener elementos de tipos distintos:

```
>>> fecha = ["Lunes", 27, "Octubre", 1997]
```

O pueden contener listas:

```
>>> peliculas = [ ["Senderos de Gloria", 1957], ["Hannah y sus hermanas", 1986]]
```

Concatenar listas: Las listas se pueden concatenar con el símbolo de la suma (+):

```
>>> vocales = ["E", "I", "O"]
>>> vocales
['E', 'I', 'O']
>>> vocales = vocales + ["U"]
>>> vocales
['E', 'I', 'O', 'U']
>>> vocales = ["A"] + vocales
>>> vocales
['A', 'E', 'I', 'O', 'U']
```

Manipular elementos individuales de una lista: Cada elemento se identifica por su posición en la lista, teniendo en cuenta que se empieza a contar por 0.

```
>>> fecha = [27, "Octubre", 1997]
>>> fecha[0]
27
>>> fecha[1]
Octubre
>>> fecha[2]
1997
```

Manipular sublistas: De una lista se pueden extraer sublistas, utilizando la notación nombreDeLista[inicio:límite], donde inicio y límite hacen el mismo papel que en el tipo range(inicio, límite).

```
>>> dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"]
>>> dias[1:4] # Se extrae una lista con los valores 1, 2 y 3
['Martes', 'Miércoles', 'Jueves']
>>> dias[4:5] # Se extrae una lista con el valor 4
['Viernes']
>>> dias[4:4] # Se extrae una lista vacía
[]
>>> dias[:4]  # Se extrae una lista hasta el valor 4 (no incluido)
['Lunes', 'Martes', 'Miércoles', 'Jueves']
>>> dias[4:]  # Se extrae una lista desde el valor 4 (incluido)
['Viernes', 'Sábado', 'Domingo']
>>> dias[:]   # Se extrae una lista con todos los valores
['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado', 'Domingo']
```

Recorrer una lista

Se puede recorrer una lista de principio a fin de dos formas distintas:

- Directamente: que la variable de control del bucle tome los valores de la lista que estamos recorriendo:

```
letras = ["A", "B", "C"]
for i in letras:
    print(i, end=" ")
```

- Indirectamente: la variable de control del bucle toma como valores los índices de la lista que estamos recorriendo (0,1 ,2 , etc.). En este caso, para acceder a los valores de la lista hay que utilizar letras[i]:

```
letras = ["A", "B", "C"]
for i in range(len(letras)):
    print(letras[i], end=" ")
```

Encontrar valor en una lista:

```
personas_autorizadas = ["Alberto", "Carmen"]
nombre = input("Dígame su nombre: ")
if nombre in personas_autorizadas:
  print("Está autorizado")
else:
  print("No está autorizado")
```

El tipo **range**(): crea una lista inmutable de números enteros en sucesión aritmética. El tipo range() puede tener uno, dos o tres argumentos numéricos enteros. El tipo range() con un único argumento se escribe range(n) y crea una lista creciente de n términos enteros que empieza en 0 y acaba antes de llegar a n (los términos aumentan de uno en uno). Es decir, range(n) = range(0, n) = [0, 1, ... , n-1].

Para ver los valores de la lista creada con range(), es necesario convertirla a lista mediante la función list().

```
>>> range(10)
range(0, 10)
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

La función **len**(): devuelve la longitud de una cadena de caracteres o el número de elementos de una lista. El argumento de la función len() es la lista o cadena que queremos "medir".

```
>>> len("mensaje secreto")
15
>>> len(["a","b","c"])
3
>>> len(range(1, 100, 7))
15
```

9. *Gráficos*

Los módulos más conocidos son:

- `Brython: SVG`
- `Tortuga: turtle`
- Biblioteca Pygame

El módulo turtle. Hace falta importarlo:

Si se va a escribir código no orientado a objetos: from turtle import *

Si se va a escribir código orientado a objetos: import turtle

La función setup(ancho, alto, posicionX, posicionY) permite definir el tamaño y la posición inicial de la ventana. Si no se usa la función setup(), la ventana se crea en el centro de la pantalla.

La función **title**() permite definir el título de la ventana.

```
from turtle import *
setup(450, 150, 0, 0)
title("Ejemplo de ventana")
```

área de dibujo: **screensize**()

```
from turtle import *
setup(250, 100, 0, 0)
screensize(100, 100)
hideturtle()
dot(10, 0, 0, 0)
screensize(200, 100)
```

La tortuga: **showturtle**(), **hideturtle**() muestra y oculta el cursor.

Dibujar líneas: Mover el lápiz: **goto**(), **setx**() y **sety**()

```
from turtle import *
setup(450, 200, 0, 0)
screensize(300, 150)
goto(100, 50)
sety(-50)
setx(50)
```


Levantar y bajar el lápiz: **penup**() y **pendown**()

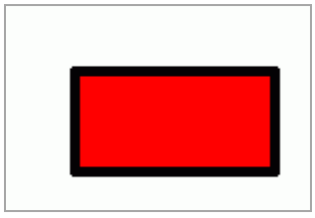
Grosor del trazo: **pensize**()

Color del trazo: **pencolor**() y **colormode**()

Dibujar puntos: **dot**()

Rellenar figuras: begin_fill() - end_fill() - **fillcollor**(color)

```
from turtle import *
setup(450, 200, 0, 0)
screensize(300, 150)
title("www.mclibre.org")
hideturtle()
pensize(5)
fillcolor("red")
begin_fill()
goto(100, 0)
goto(100, 50)
goto(0, 50)
goto(0, 0)
end_fill()
```



Orientación a objetos y ejercicios complementarios:

Véase en http://www.mclibre.org/consultar/python/

Gráficos y Juegos con Pygame

Instalar pygame desde https://bitbucket.org/pygame/pygame/downloads

Véase: http://programarcadegames.com/

Cargar e inicializar la biblioteca Pygame:

```
Importar e inicializar Pygame
# Importa  la librería de funciones llamada 'pygame'
import pygame
# Inicializa el motor de juegos
pygame.init()
```

Definir colores

```
# Definir algunos colores
NEGRO  = (  0,   0,   0)
BLANCO = (255, 255, 255)
VERDE  = (0,   255,   0)
ROJO   = (255,   0,   0)
```

Abrir y establecer las dimensiones de una ventana

```
dimensiones = (700, 500)
pantalla = pygame.display.set_mode(dimensiones)
```

Establecer el título de la ventana

```
pygame.display.set_caption("Super Juego")
```

Muestra de bucle de eventos

```
for evento in pygame.event.get():
    if evento.type == pygame.QUIT:
        print("El usuario solicitó salir.")
    elif evento.type == pygame.KEYDOWN:
        print("El usuario presionó una tecla.")
    elif evento.type == pygame.KEYUP:
        print("El usuario soltó una tecla.")
    elif evento.type == pygame.MOUSEBUTTONDOWN:
        print("El usuario presionó un botón del ratón")
```

Cierre de un programa Pygame: `pygame.quit()`

Limpia la ventana y establece el fondo blanco: `pantalla.fill(BLANCO)`

Actualizar la pantalla: pygame.display.flip()

Dibujar líneas:

```
pygame.draw.line(pantalla, VERDE, [0, 0], [100, 100], 5)
# Dibuja en la pantalla una línea verde desde (0, 0) hasta (100, 100)  y 5 píxeles de grosor.
```

Representar un rectángulo:

```
# Dibuja un rectángulo.
pygame.draw.rect(pantalla, NEGRO, [20, 20, 250, 100], 2)
```

Representar una elipse

```
# Representa una elipse, usando un rectángulo como perímetro exterior
pygame.draw.ellipse(pantalla, NEGRO, [20, 20, 250, 100], 2)
```

Representar arcos

```
# Representa un arco en radianes
pygame.draw.arc(pantalla, VERDE, [100, 100, 250, 200], pi/2, pi, 2)
```

Representar un polígono

```
# Esto dibuja un triángulo usando el comando polygon
pygame.draw.polygon(pantalla, NEGRO, [[100, 100],[0, 200],[200, 200]], 5)
```

texto sobre la pantalla

```
fuente = pygame.font.Font(None, 25) # Selecciona la fuente. Fuente Default, tamaño 25 pt.
texto = fuente.render("Mi texto", True, NEGRO)
pantalla.blit(texto, [250, 250]) # Coloca la imagen del texto sobre la pantalla en 250 x 250
```

Ejercicio gráfico con pygame:

```
Sencilla demo juego gráfico
import pygame  # Importamos una biblioteca de funciones llamada 'pygame'
pygame.init()  # Inicializamos el motor de juegos
# Definimos algunos colores
NEGRO = (0, 0, 0)
BLANCO = (255, 255, 255)
AZUL = (0, 0, 255)
VERDE = (0, 255, 0)
ROJO = (255, 0, 0)
PI = 3.141592653

dimensiones = (400, 500)     # Establecemos el largo y alto de la pantalla
pantalla = pygame.display.set_mode(dimensiones)

pygame.display.set_caption("Demo juego grafico")

#Iteramos hasta que el usuario haga click sobre el botón de salida.
hecho = False
reloj = pygame.time.Clock()

# Iteramos en el bucle hasta que hecho == False
while not hecho:

    for evento in pygame.event.get(): # El usuario realizó alguna acción
        if evento.type == pygame.QUIT: # Si el usuario hizo click sobre salir
            hecho = True # Marcamos que hemos acabado y abandonamos este bucle
    # Limpiamos la pantalla y establecemos su fondo.
    pantalla.fill(BLANCO)
     # Dibujamos sobre la pantalla una línea verde de 5 píxeles de ancho
    # que vaya desde (0,0) hasta (100,100).
    pygame.draw.line(pantalla, VERDE, [0, 0], [100, 100], 5)
    # Usando un bucle for, dibujamos sobre la pantalla varias líneas rojas de 5 píxeles
    # de ancho, que vayan desde (0,10) hasta (100,110).
    for desplazar_y in range(0, 100, 10):
        pygame.draw.line(pantalla, ROJO, [0, 10 + desplazar_y], [100, 110 + desplazar_y], 5)
    # Dibujamos un rectángulo
    pygame.draw.rect(pantalla, NEGRO, [20, 20, 250, 100], 2)
    # Dibujamos una elipse, usando un rectángulo para fijar sus bordes exteriores
    pygame.draw.ellipse(pantalla, NEGRO, [20, 20, 250, 100], 2)
    # Dibujamos un arco como parte de una elipse.
    # Usamos radianes para determinar qué ángulo tenemos que dibujar.
    pygame.draw.arc(pantalla, NEGRO, [20, 220, 250, 200], 0, PI / 2, 2)
    pygame.draw.arc(pantalla, VERDE, [20, 220, 250, 200], PI / 2, PI, 2)
    pygame.draw.arc(pantalla, AZUL, [20, 220, 250, 200], PI, 3 * PI / 2, 2)
    pygame.draw.arc(pantalla, ROJO, [20, 220, 250, 200], 3 * PI / 2, 2 * PI, 2)
    # Aquí dibujamos un triángulo empleando el comando polygon.
    pygame.draw.polygon(pantalla, NEGRO, [[100, 100], [0, 200], [200, 200]], 5)
    # Elegimos que tipo de fuente usar; fuente por defecto, y de 25 puntos.
    fuente = pygame.font.Font(None, 25)
    # Creamos el texto. "True" significa texto con antialiasing.
    # El color es negro. Esto nos crea una imagen de las letras, pero no las coloca
    # sobre la pantalla.
    texto = fuente.render("Mi texto", True, NEGRO)
    # Coloca la imagen del texto en pantalla sobre las coordenadas 250x250.
    pantalla.blit(texto, [250, 250])
    # Avanzamos y actualizamos la pantalla con lo que hemos dibujado.
    # Esto DEBE suceder después del resto de comandos de dibujo.
    pygame.display.flip()
    # Aquí limitamos el bucle while a un máximo de 60 veces por segundo.
    reloj.tick(60)
pygame.quit()
```