

# TRABAJO PRÁCTICO INTEGRAL

## INTEGRANTES

NICOLÁS MACHADO

MATÍAS CARBALLO

SAUL ESPINOSA

GABRIEL MOYANO

Notion:

<https://www.notion.so/Los-Programadores-7911e0a5f1794c64b2866345fd4900b4?pvs=4>

# Trabajo en Git/GitHub

- ▶ En la siguiente captura se puede ver los commits hechos por cada cambio realizado en el proyecto, así cada integrante estaba al día con la ultima actualización/modificación del código
- ▶ <https://github.com/SaulDavid1998/TPI>

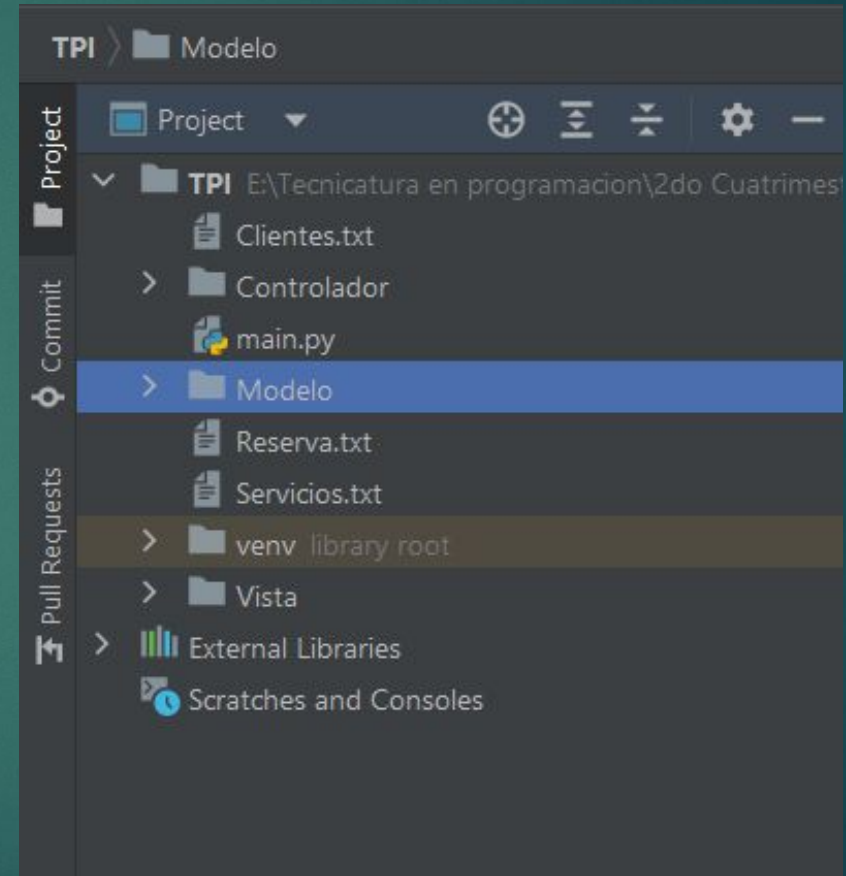
The screenshot shows the GitHub interface for a repository named 'TPI' (Private). At the top, it indicates 'master' branch, '1 branch', and '0 tags'. Below this, there's a commit history table. The table has three columns: a file/folder icon, a description of the commit, and the time since the commit. The commit 'c437a1f' was made '8 minutes ago' and has '15 commits' associated with it. The files listed are 'Controlador', 'Modelo', 'Vista', 'Clientes.txt', 'Reserva.txt', 'Servicios.txt', and 'main.py'. At the bottom, there's a prompt to 'Add a README'.

File/Folder	Commit Message	Time Ago
Controlador	Las reservas devuelven el nombre del cliente y no el ID	8 minutes ago
Modelo	Las reservas devuelven el nombre del cliente y no el ID	8 minutes ago
Vista	Evite que se ingresara una ID de cliente inexistente	22 minutes ago
Clientes.txt	Creamos Mostrar Clientes y Mostrar Fechas	4 days ago
Reserva.txt	Deteccion de dia mas cercano finalizado	2 hours ago
Servicios.txt	Se termino el sistema de reserva, falta cambiar el estado	8 hours ago
main.py	Deteccion de dia mas cercano finalizado	2 hours ago

# Estructura del proyecto

El proyecto, al ser MVC, consta de la siguiente estructura

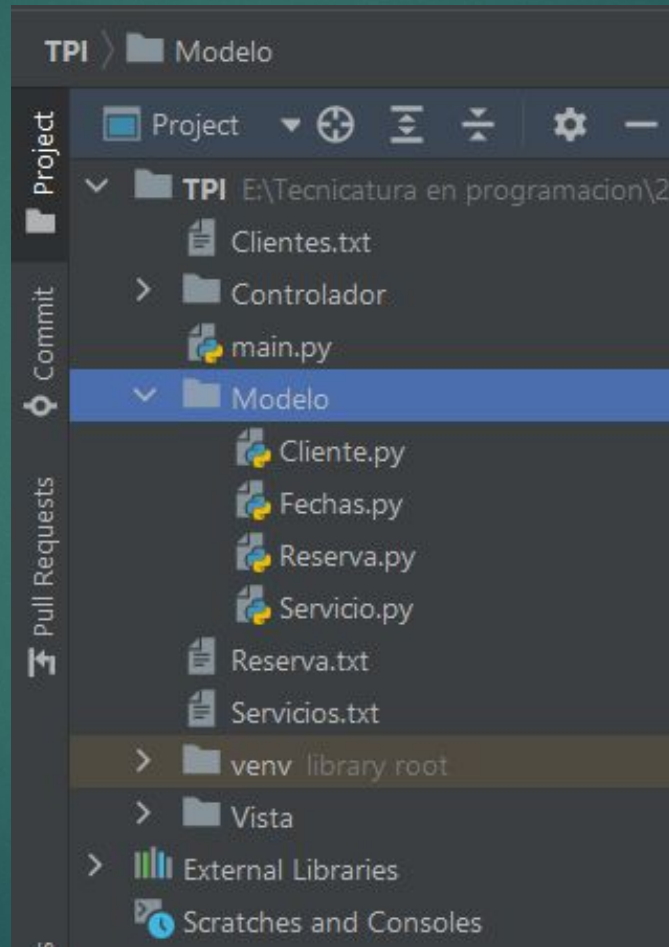
- ▶ Carpeta/directorio Modelo
- ▶ Carpeta/directorio Vista
- ▶ Carpeta/directorio Controlador
- ▶ 3 archivos .txt (Reserva, Servicio, Clientes)
- ▶ Archivo main.py



# MODELO

El modelo consta de las siguientes clases:

- ▶ Cliente.py
- ▶ Fechas.py
- ▶ Reserva.py
- ▶ Servicio.py





# MODELO

## Clase Cliente

La clase Cliente consta de los siguientes métodos y atributos:

- ▶ Método init: pide 3 parámetros (nombre, dni, teléfono y edad) y adentro de este método se declaran los atributos nombre, dni, teléfono y edad e inicializan con los valores de los parámetros.
- ▶ Métodos Get y Set: las variables creadas e inicializadas dentro del constructor van a tener sus respectivos Get y Set para poder obtener el valor actual o establecerle uno nuevo
- ▶ Método str: este método va a retornar lo que se quiera mostrar del objeto.

# MODELO

## Clase Cliente

```
TPI > Modelo > Cliente.py
Cliente.py x
Project
  Saul +1
  1 class Cliente:
  2     Saul +1
  3     def __init__(self, nombre="", dni=0, telefono=0, edad=0):
  4         self.__nombre=nombre
  5         self.__dni=dni
  6         self.__telefono=telefono
  7         self.__edad=edad
  8     Saul
  9     def GetNombre(self):
 10         return self.__nombre
 11     Saul
 12     def GetDni(self):
 13         return self.__dni
 14     Saul
 15     def GetTelefono(self):
 16         return self.__telefono
 17     Saul
 18     def GetEdad(self):
 19         return self.__edad
```

```
def SetNombre(self, nombre):
    self.__nombre=nombre

    Saul
    def SetDni(self, dni):
        self.__dni=dni

    Saul
    def SetTelefono(self, telefono):
        self.__telefono=telefono

    Saul
    def SetEdad(self, edad):
        self.__edad=edad

    Nico
    def __str__(self, id):
        return "|| ID: " + str(id) + " - Nombre: " + str(self.__nombre) + "
```

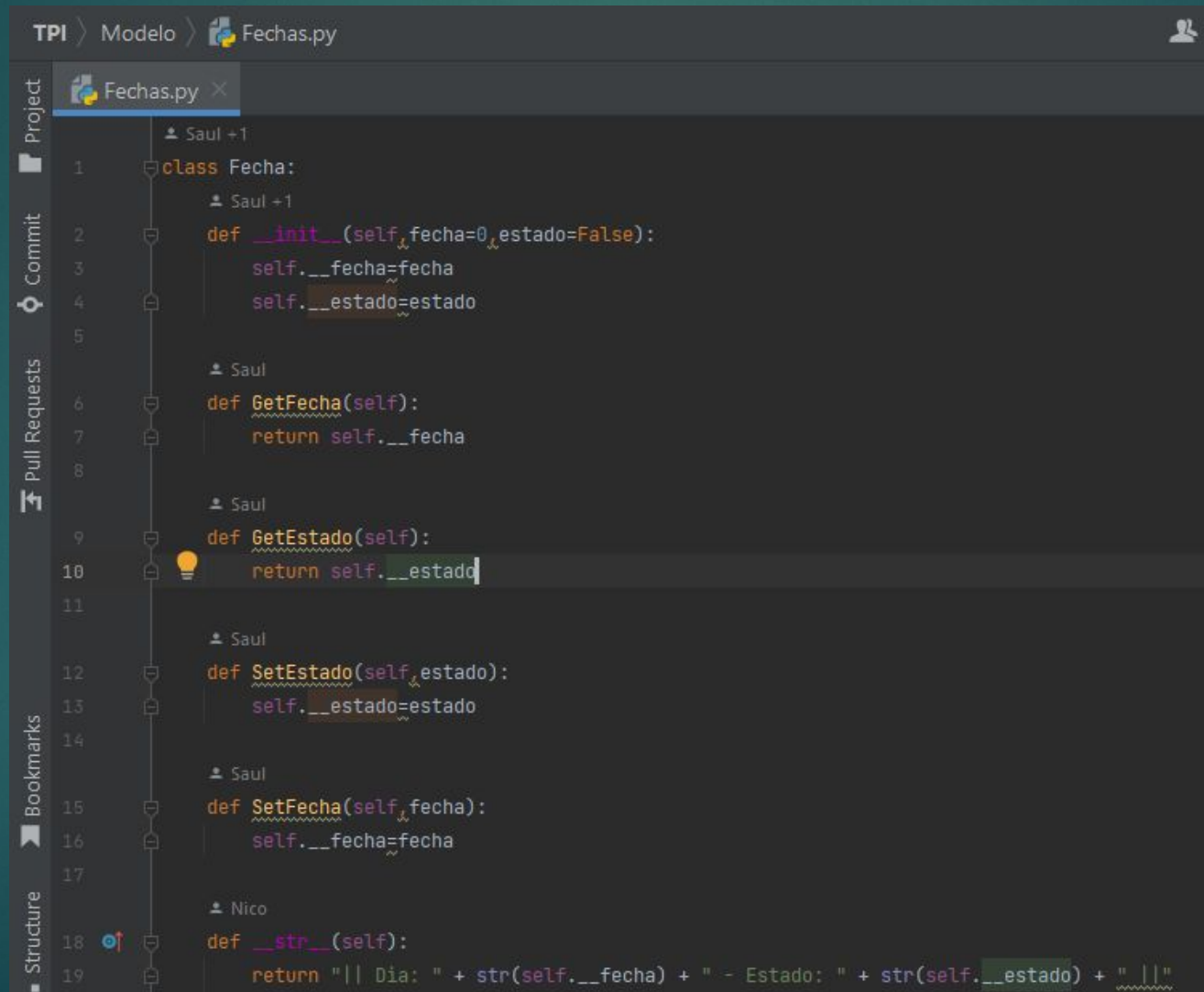
# MODELO

## Clase Fechas

- ▶ Método init: pide 2 parámetros (fecha y estado) y adentro de este método se declaran los atributos fecha y estado e inicializan con los valores de los parámetros.
- ▶ Métodos Get y Set: los atributos creados e inicializadas dentro del constructor van a tener sus respectivos Get y Set para poder obtener el valor actual o establecerle uno nuevo
- ▶ Método str: este método va a retornar lo que se quiera mostrar del objeto.

# MODELO

## Clase Fechas



```
TPI > Modelo > Fechas.py

Fechas.py x
├── Saul +1
1  class Fecha:
2      └── Saul +1
3      def __init__(self, fecha=0, estado=False):
4          self.__fecha=fecha
5          self.__estado=estado
6
7      └── Saul
8      def GetFecha(self):
9          return self.__fecha
10
11     └── Saul
12     def GetEstado(self):
13         return self.__estado
14
15     └── Saul
16     def SetEstado(self, estado):
17         self.__estado=estado
18
19     └── Saul
20     def SetFecha(self, fecha):
21         self.__fecha=fecha
22
23     └── Nico
24     def __str__(self):
25         return "|| Dia: " + str(self.__fecha) + " - Estado: " + str(self.__estado) + " ||"
```



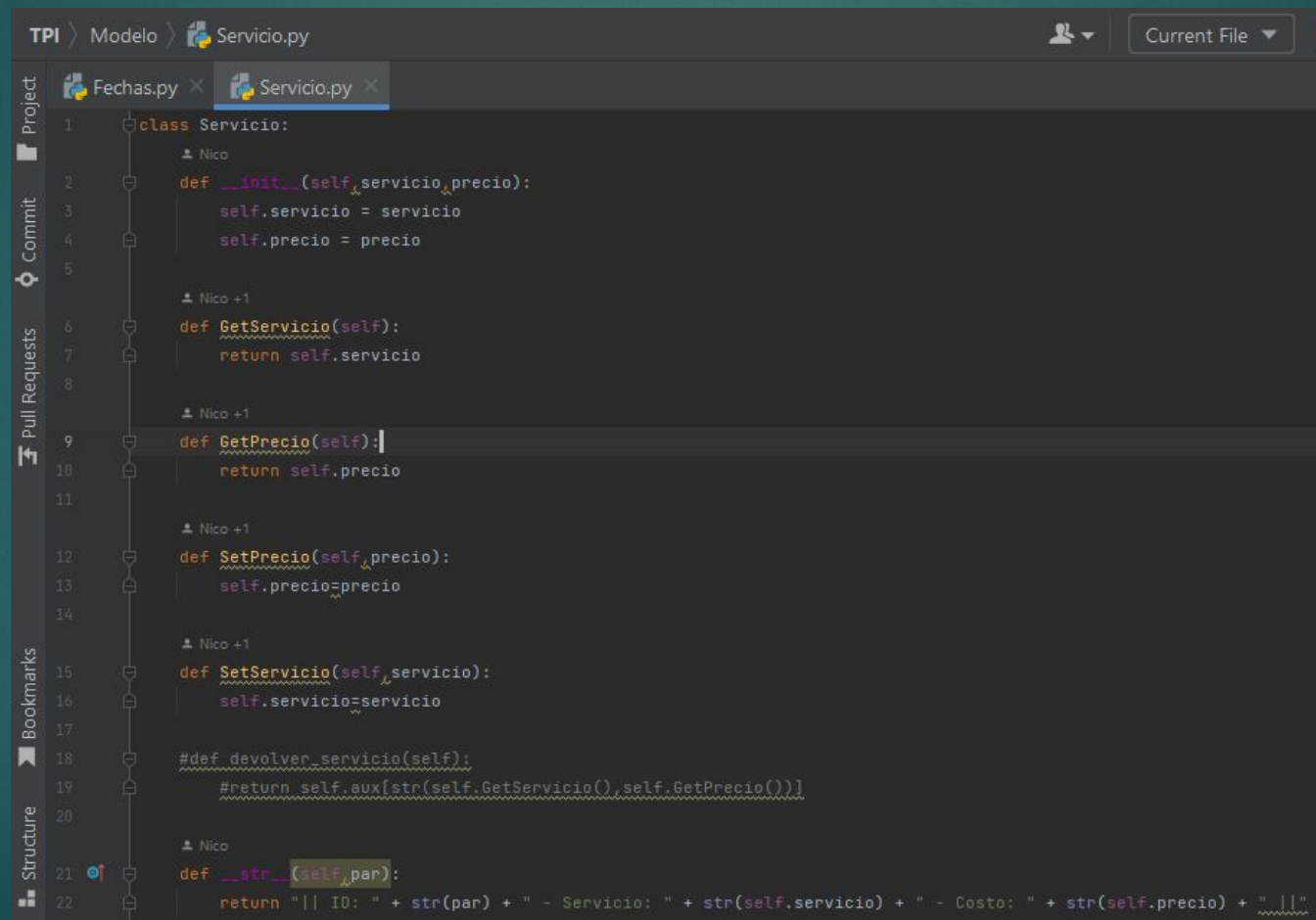
# MODELO

## Clase Servicio

- ▶ Método init: pide 2 parámetros (servicio y precio) y adentro de este método se declaran los parámetros servicio y precio e inicializan con los valores de los parámetros.
- ▶ Métodos Get y Set: los atributos creados dentro del constructor van a tener sus respectivos Get y Set para poder obtener el valor actual o establecerle uno nuevo.
- ▶ Método str: este método va a retornar lo que se quiera mostrar del objeto.

# MODELO

## Clase Servicio



```
1 class Servicio:
2     def __init__(self, servicio, precio):
3         self.servicio = servicio
4         self.precio = precio
5
6     def GetServicio(self):
7         return self.servicio
8
9     def GetPrecio(self):
10        return self.precio
11
12    def SetPrecio(self, precio):
13        self.precio = precio
14
15    def SetServicio(self, servicio):
16        self.servicio = servicio
17
18    #def devolver_servicio(self):
19    #    return self.aux[str(self.GetServicio()), self.GetPrecio()]
20
21    def __str__(self, par):
22        return "|| ID: " + str(par) + " - Servicio: " + str(self.servicio) + " - Costo: " + str(self.precio) + " ||"
```

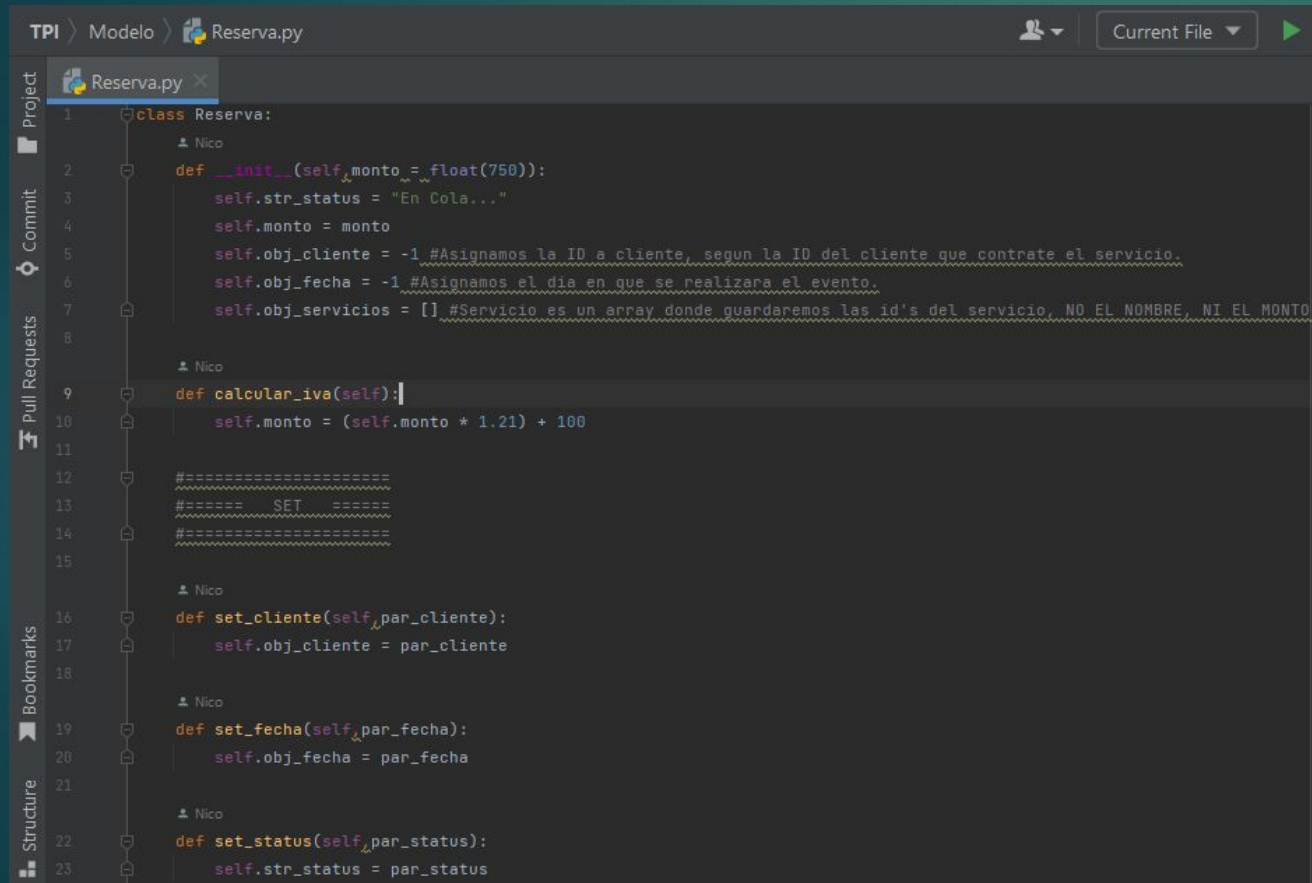
# MODELO

## Clase Reserva

- ▶ Método init: pide un parámetro (monto) y adentro de este método se declaran los atributos monto, str\_status, obj\_cliente, obj\_fecha y obj\_servicios.  
El atributo monto se va a inicializar con el valor del parámetro  
El atributo str\_status se va a inicializar con el valor "En cola..."  
Los atributos obj\_fecha y obj\_cliente ambos van a inicializarse con el valor -1  
El atributo obj\_servicios se va a inicializar como una lista vacía
- ▶ Métodos Get: los atributos creados dentro del constructor van a tener sus respectivos métodos Get para poder acceder a su valor.
- ▶ Métodos Set: los atributos str\_status, obj\_cliente, obj\_fecha van a tener métodos Set para poder actualizar/cambiar sus valores

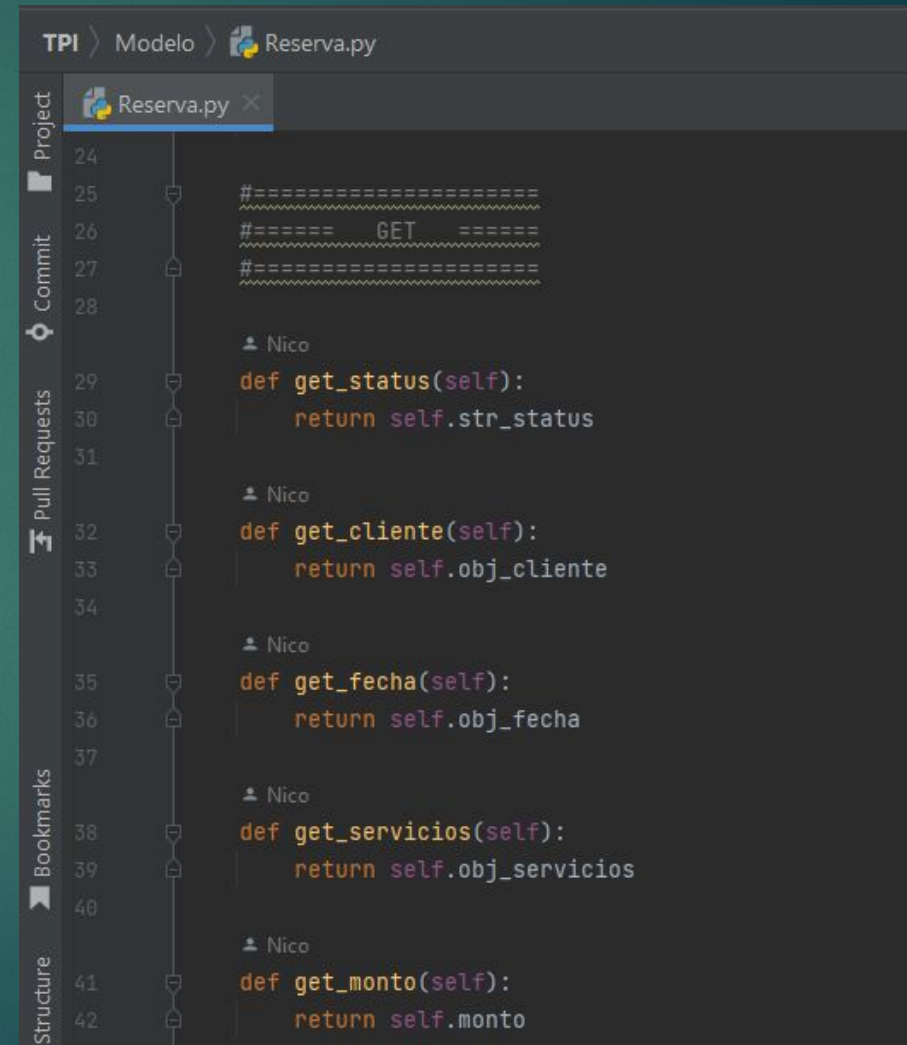
# MODELO

## Clase Reserva



The screenshot shows a code editor with the file 'Reserva.py' open. The code defines a class 'Reserva' with the following methods:

```
1 class Reserva:
2     def __init__(self, monto = float(750)):
3         self.str_status = "En Cola..."
4         self.monto = monto
5         self.obj_cliente = -1 #Asignamos la ID a cliente, segun la ID del cliente que contrate el servicio.
6         self.obj_fecha = -1 #Asignamos el dia en que se realizara el evento.
7         self.obj_servicios = [] #Servicio es un array donde guardaremos las id's del servicio, NO EL NOMBRE, NI EL MONTO.
8
9     def calcular_iva(self):
10         self.monto = (self.monto * 1.21) + 100
11
12     #=====
13     #===== SET =====
14     #=====
15
16     def set_cliente(self, par_cliente):
17         self.obj_cliente = par_cliente
18
19     def set_fecha(self, par_fecha):
20         self.obj_fecha = par_fecha
21
22     def set_status(self, par_status):
23         self.str_status = par_status
```



The screenshot shows a code editor with the file 'Reserva.py' open, displaying the GET methods of the 'Reserva' class. The code is as follows:

```
24
25 #=====
26 #===== GET =====
27 #=====
28
29 def get_status(self):
30     return self.str_status
31
32 def get_cliente(self):
33     return self.obj_cliente
34
35 def get_fecha(self):
36     return self.obj_fecha
37
38 def get_servicios(self):
39     return self.obj_servicios
40
41 def get_monto(self):
42     return self.monto
```



# MODELO

## Clase Reserva

- Métodos add: estos métodos reemplazan a los tradicionales métodos Set para los atributos, pero cumplen una tarea muy similar a ellos

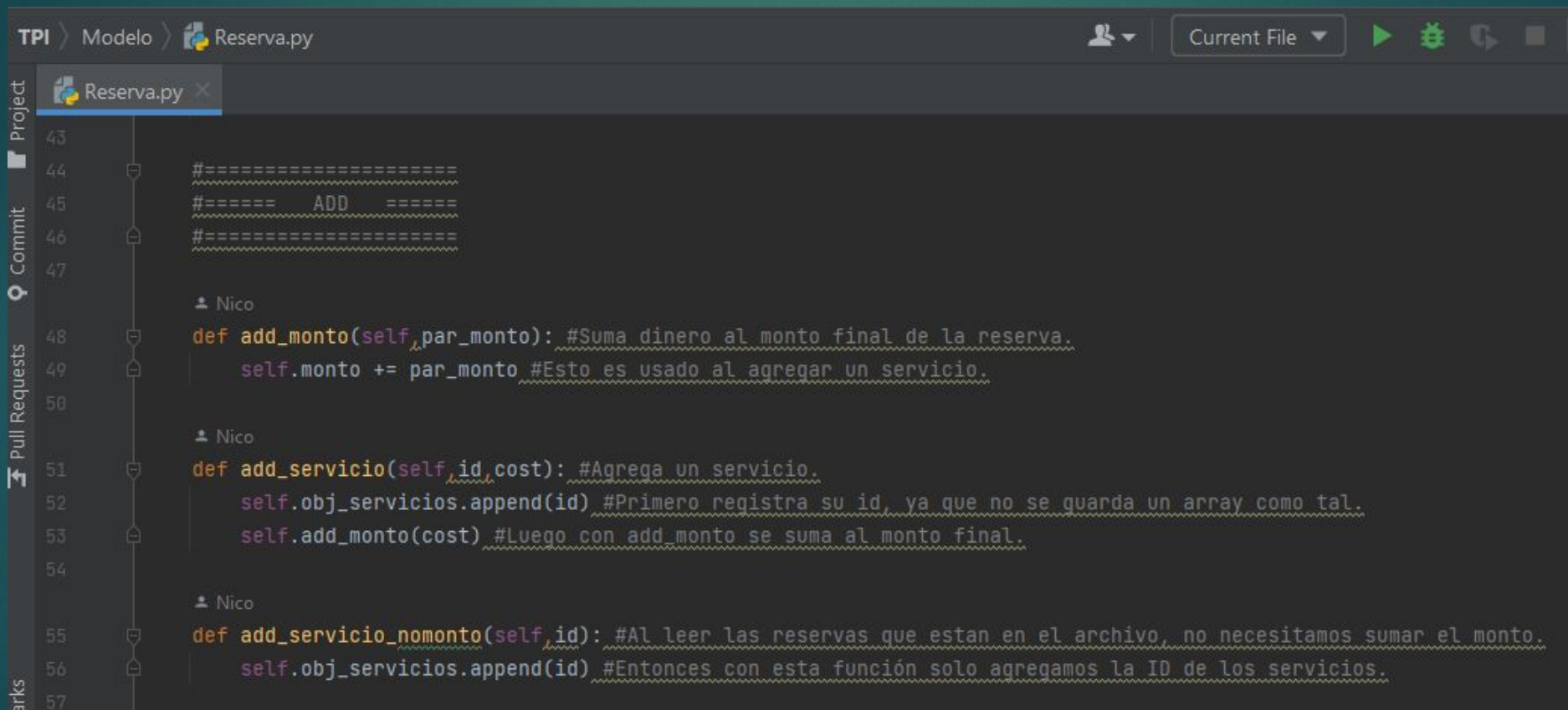
El método `add_servicio` pide 2 parámetros (`id` y `cost`). Este método agrega un servicio. Adentro de este método al atributo `obj_servicios` se le adjunta/añade un valor, el cual va a ser el valor del parámetro `id`, y se invoca al método `add_monto` y se le pasa el valor del parámetro `cost`.

El método `add_monto` pide un parámetro (`par_monto`). Adentro de este método se adiciona un nuevo valor para el atributo `monto`. El nuevo valor va a ser el el valor del parámetro `par_monto`

El método `add_servicio_nomonto` pide un parámetro (`id`). Este método lee las reservas dentro del archivo `Reservas.txt`, por lo que no hace falta asignarle un monto por que ya esta establecido en el archivo de texto

# MODELO

## Clase Reserva



```
TPI > Modelo > Reserva.py
Reserva.py x
43
44 #=====
45 #=====  ADD  =====
46 #=====
47
48 Nico
49 def add_monto(self, par_monto): #Suma dinero al monto final de la reserva.
50     self.monto += par_monto #Esto es usado al agregar un servicio.
51
52 Nico
53 def add_servicio(self, id, cost): #Agrega un servicio.
54     self.obj_servicios.append(id) #Primero registra su id, ya que no se guarda un array como tal.
55     self.add_monto(cost) #Luego con add_monto se suma al monto final.
56
57 Nico
58 def add_servicio_nomonto(self, id): #Al leer las reservas que estan en el archivo, no necesitamos sumar el monto.
59     self.obj_servicios.append(id) #Entonces con esta función solo agregamos la ID de los servicios.
```

# MODELO

## Clase Reserva

- Métodos str: a diferencia de las otras clases del modelo, esta clase posee varios métodos str, pero todos terminan retornando lo que se quiere mostrar del objeto

```
Nico
def __str__(self, type):
    match type:
        case 0: return "|| Cliente: " + str(self.get_cliente()) + " ||" #Mostramos el nombre del cliente.
        case 1: return "|| Dia: " + str(self.get_fecha()) + " ||" #Mostramos el día del evento.
        case 2: return "|| ID Servicios: " + str(self.get_servicios()) + " ||" #Mostramos el ID de los servicios.
        case 3: return "|| $ " + str(self.get_monto()) + " ||" #Mostramos el monto total.
        case 4: return "|| Estado: " + str(self.get_status()) + " ||" #Mostramos el estado.

Nico
def __str_price__(self, type):
    match type:
        case 0: return "Precio de la reserva: $" + str(self.get_monto()) #Mostramos el precio de la reserva + servicios.
        case 1: return "Gastos administrativos: $100" #Mostramos el gasto administrativo.
        case 2: return "IVA: " + str((self.get_monto() * 1.21) - self.get_monto()) + "%" #Mostramos lo que se suma debido al IVA.
        case 3: return "En total, el evento costara: $" + str(self.get_monto()) #Mostramos el precio final.
        case 4: return "La seña a realizar es de: $" + str(self.get_monto() * 0.3) #Mostramos el precio a señar.

Nico
def __str_id__(self, id):
    return "|| ID: " + str(id) + " ||"

Nico
def __str_senia__(self):
    return "La seña a realizar es de: $" + str(self.get_monto() * 0.3)

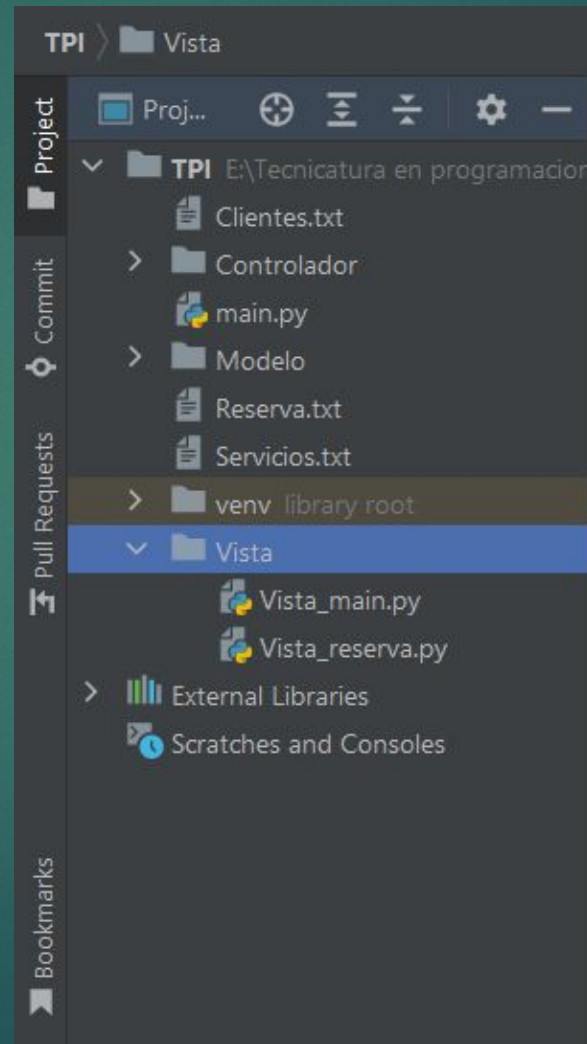
Nico
def __str_cancelar__(self):
    if self.get_status() == "Señado": #Si la reserva esta señada, se devolvera parte de la seña, en caso contrario, no.
        return "El dinero a devolver es: $" + str((self.get_monto() * 0.3) * 0.2)
    else:
        return "No hay que devolver dinero."
```



# VISTA

La vista consta de las siguientes clases:

- ▶ Vista\_main.py
- ▶ Vista\_Reserva.py

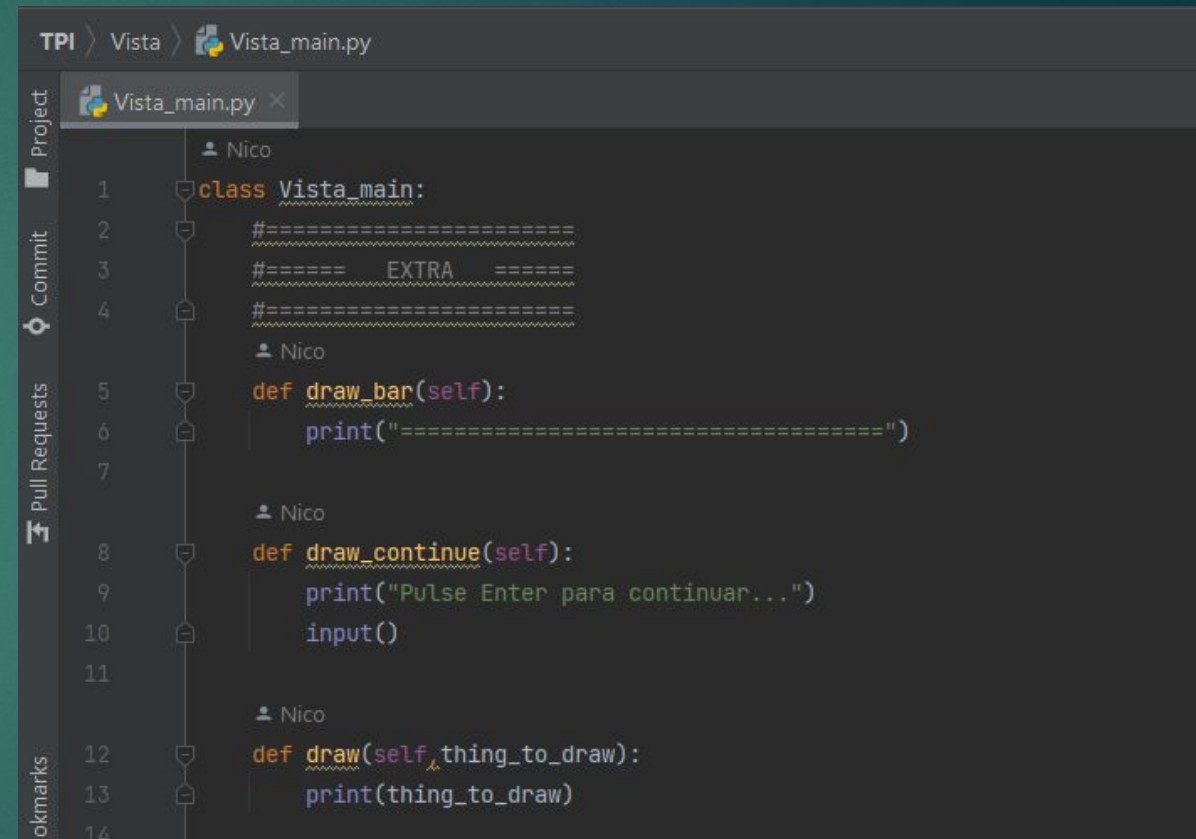




# VISTA

## Clase Vista\_main

- ▶ La clase Vista\_main va a ser donde se van a mostrar los datos de salida (print) y se van a pedir los datos de entrada(input) para interactuar con el programa.
- ▶ Esta clase no tiene codificado un constructor
- ▶ Los primeros 3 métodos de esta clase son 3 métodos para mostrar información por pantalla: draw\_bar, draw\_continue y draw
- ▶ A diferencia de draw\_bar y draw\_continue, draw va a mostrar/escribir lo que se le paso por el parámetro que posee.

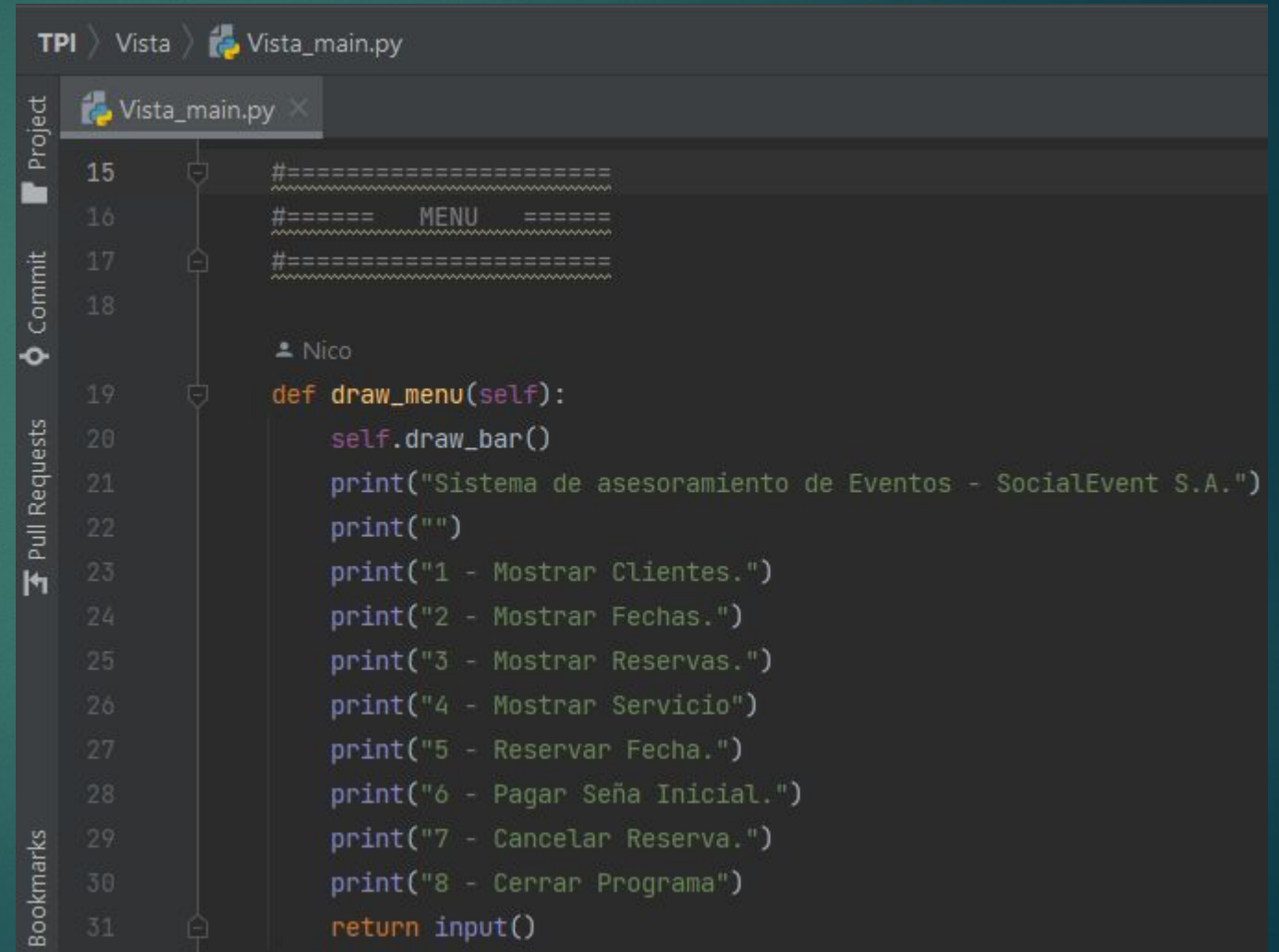


```
TPI > Vista > Vista_main.py
Vista_main.py x
Nico
1 class Vista_main:
2     #=====
3     #===== EXTRA =====
4     #=====
5     def draw_bar(self):
6         print("=====")
7
8     def draw_continue(self):
9         print("Pulse Enter para continuar...")
10        input()
11
12    def draw(self, thing_to_draw):
13        print(thing_to_draw)
14
```

# VISTA

## Clase Vista\_main

- ▶ El método `draw_menu` invoca al método `draw_bar`, escribe/muestra las opciones del menú y espera un dato de entrada, el cual va a ser retornado para ser usado por quien invoque este método



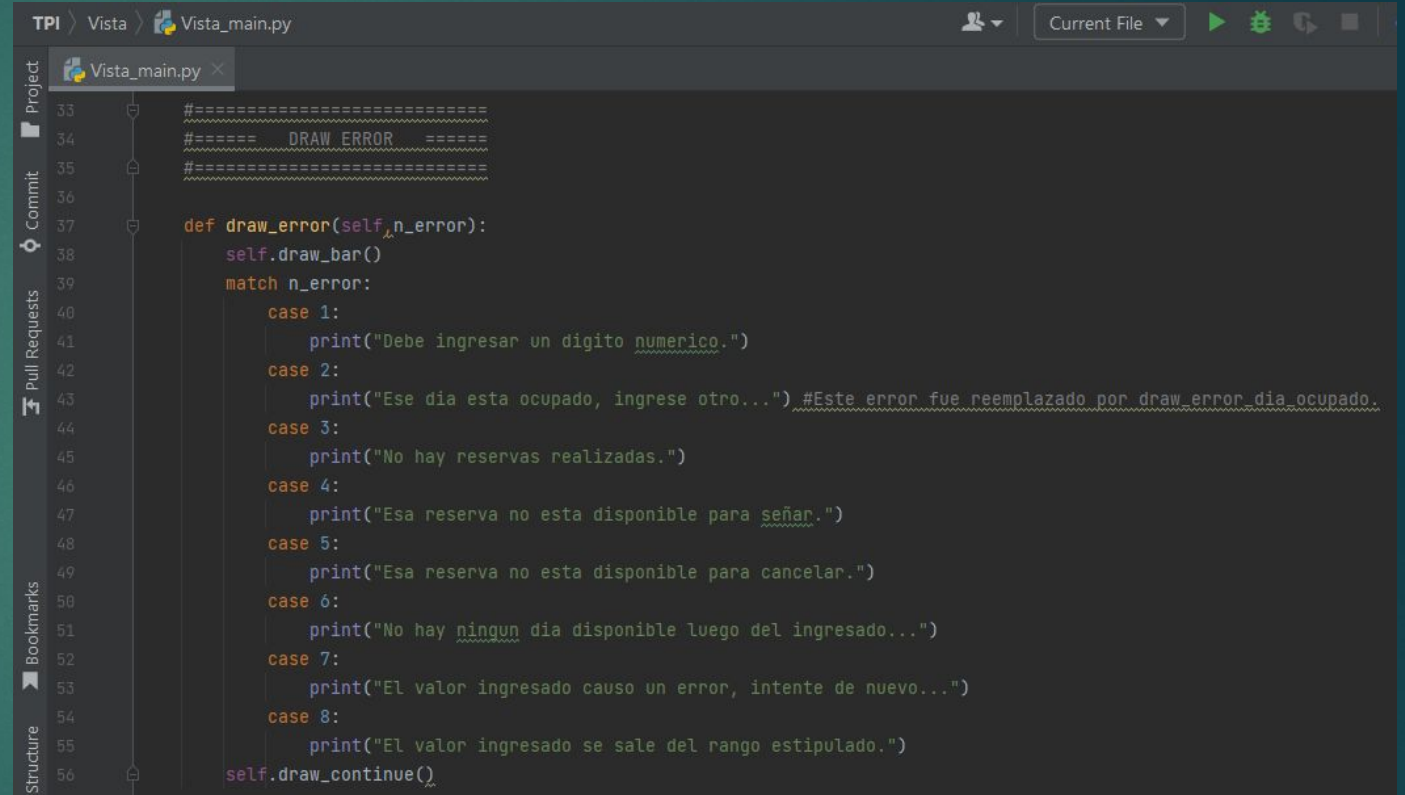
```
TPI > Vista > Vista_main.py

Vista_main.py x
15  #####
16  ##### MENU #####
17  #####
18
19  Nico
20  def draw_menu(self):
21      self.draw_bar()
22      print("Sistema de asesoramiento de Eventos - SocialEvent S.A.")
23      print("")
24      print("1 - Mostrar Clientes.")
25      print("2 - Mostrar Fechas.")
26      print("3 - Mostrar Reservas.")
27      print("4 - Mostrar Servicio")
28      print("5 - Reservar Fecha.")
29      print("6 - Pagar Señal Inicial.")
30      print("7 - Cancelar Reserva.")
31      print("8 - Cerrar Programa")
32      return input()
```

# VISTA

## Clase Vista\_main

El método `draw_error` invoca al método `draw_bar`, pide un parámetro (`n_error`) el cual va a ser usado en la sentencia `switch` para escribir/mostrar las opciones en caso de error en donde el mensaje de error a mostrarse va a ser el del valor pasado por el parámetro `n_error` y al final invoca al método `draw_continue`



```
TPI > Vista > Vista_main.py
Vista_main.py x
33  #####
34  #####  DRAW ERROR  #####
35  #####
36
37  def draw_error(self, n_error):
38      self.draw_bar()
39      match n_error:
40          case 1:
41              print("Debe ingresar un digito numerico.")
42          case 2:
43              print("Ese dia esta ocupado, ingrese otro...") #Este error fue reemplazado por draw_error_dia_ocupado.
44          case 3:
45              print("No hay reservas realizadas.")
46          case 4:
47              print("Esa reserva no esta disponible para se\u00f1ar.")
48          case 5:
49              print("Esa reserva no esta disponible para cancelar.")
50          case 6:
51              print("No hay ningun dia disponible luego del ingresado...")
52          case 7:
53              print("El valor ingresado causo un error, intente de nuevo...")
54          case 8:
55              print("El valor ingresado se sale del rango estipulado.")
56      self.draw_continue()
```



# VISTA

## Clase Vista reserva

- ▶ Al igual que la otra clase de la vista, esta clase también va a ser donde se van a mostrar los datos de salida (print) y se van a pedir los datos de entrada(input) para interactuar con el programa
- ▶ Esta clase no tiene codificado un constructor
- ▶ Los métodos de esta clase se pueden agrupar en 3 grupos: Métodos Enter, Métodos Pregunta y Métodos Complete



# VISTA

## Clase Vista reserva

### Métodos Enter

Los métodos Entes son los siguientes:

- draw\_enter\_fecha
- draw\_enter\_servicio
- draw\_enter\_cliente
- draw\_enter\_seniar

```
TPI > Vista > Vista_reserva.py

Vista_reserva.py x
===== ENTER =====
#=====

def draw_enter_fecha(self):
    try:
        return int(input("Ingrese el día en el que reservar: "))
    except Exception:
        self.draw_error(2)

def draw_enter_servicios(self):
    try:
        return int(input("Ingrese el ID del servicio a reservar: "))
    except Exception:
        self.draw_error(2)

def draw_enter_cliente(self):
    try:
        return int(input("Ingrese el ID del cliente a cargo de la reserva: "))
    except Exception:
        self.draw_error(2)

def draw_enter_seniar(self):
    try:
        return int(input("Ingrese el ID de la reserva a seniar: "))
    except Exception:
        self.draw_error(2)

def draw_enter_cancelar(self):
    try:
        return int(input("Ingrese el ID de la reserva a cancelar: "))
    except Exception:
        self.draw_error(2)
```

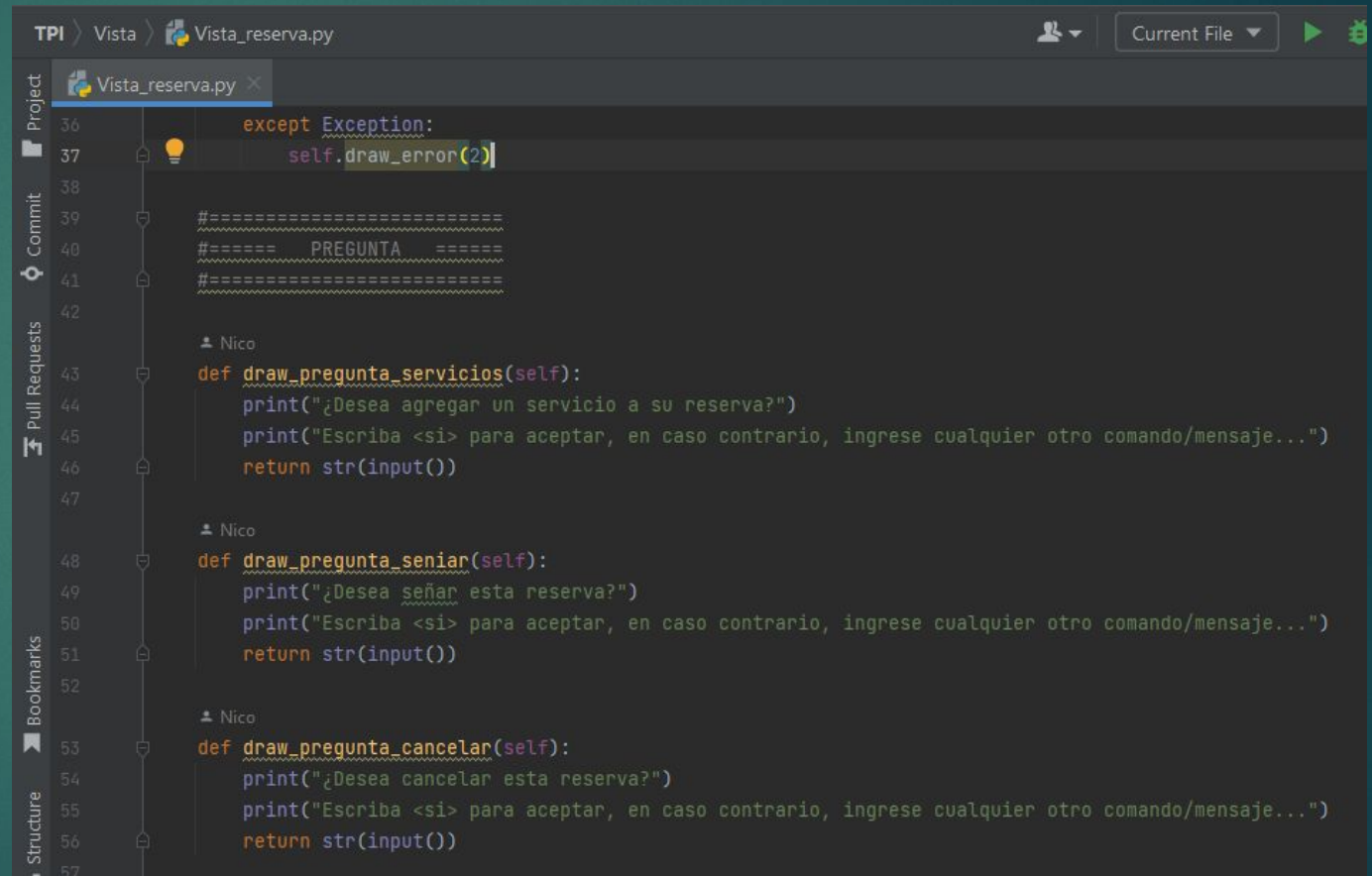
# VISTA

## Clase Vista reserva

### Métodos Pregunta

Los métodos Pregunta son los siguientes:

- draw\_pregunta\_servicios
- draw\_pregunta\_seniar
- draw\_pregunta\_cancelar



```
TPI > Vista > Vista_reserva.py
Vista_reserva.py x
36 except Exception:
37     self.draw_error(2)
38
39     =====
40     ===== PREGUNTA =====
41     =====
42
43     Nico
44     def draw_pregunta_servicios(self):
45         print("¿Desea agregar un servicio a su reserva?")
46         print("Escriba <si> para aceptar, en caso contrario, ingrese cualquier otro comando/mensaje...")
47         return str(input())
48
49     Nico
50     def draw_pregunta_seniar(self):
51         print("¿Desea señalar esta reserva?")
52         print("Escriba <si> para aceptar, en caso contrario, ingrese cualquier otro comando/mensaje...")
53         return str(input())
54
55     Nico
56     def draw_pregunta_cancelar(self):
57         print("¿Desea cancelar esta reserva?")
58         print("Escriba <si> para aceptar, en caso contrario, ingrese cualquier otro comando/mensaje...")
59         return str(input())
59
```

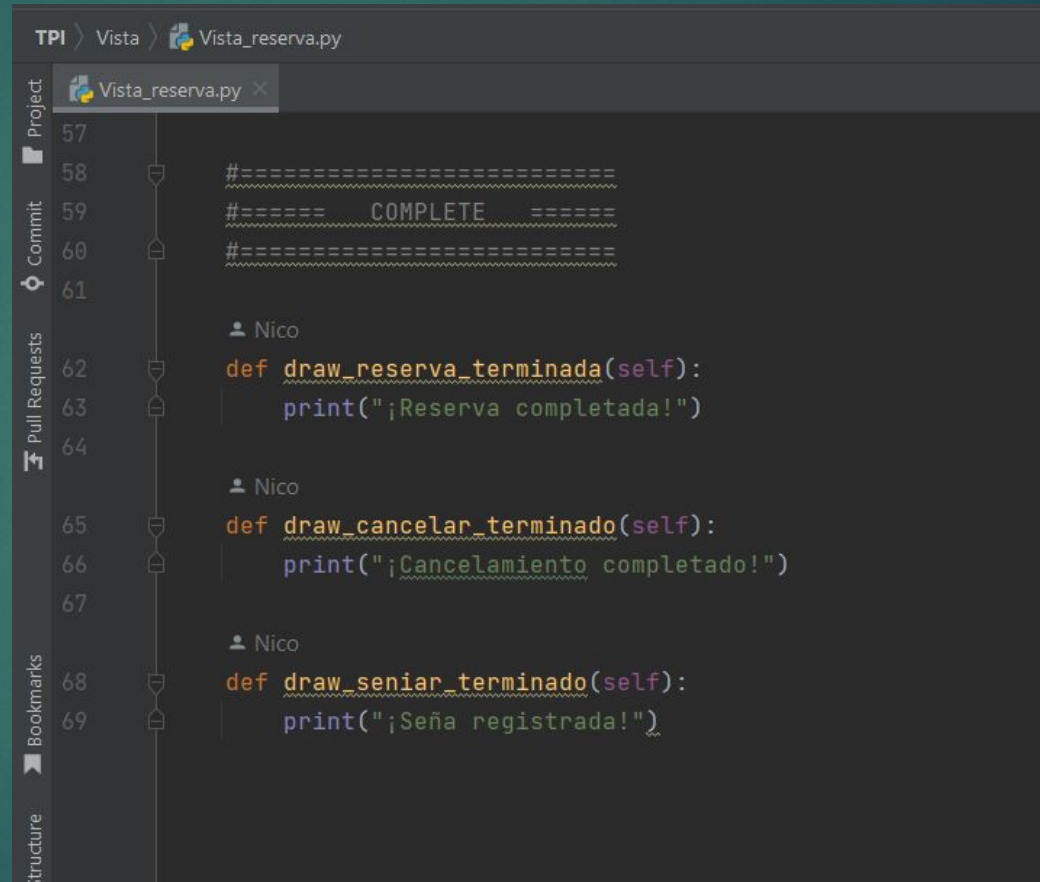
# VISTA

## Clase Vista reserva

### Métodos Complete

Los métodos Complete son los siguientes:

- draw\_reserva\_terminada
- draw\_cancelar\_terminado
- draw\_seniar\_terminado



```
TPI > Vista > Vista_reserva.py
Vista_reserva.py x
57
58 #=====
59 #===== COMPLETE =====
60 #=====
61
62 Nico
63 def draw_reserva_terminada(self):
64     print("¡Reserva completada!")
65
66 Nico
67 def draw_cancelar_terminado(self):
68     print("¡Cancelamiento completado!")
69
70 Nico
71 def draw_seniar_terminado(self):
72     print("¡Seña registrada!")
```



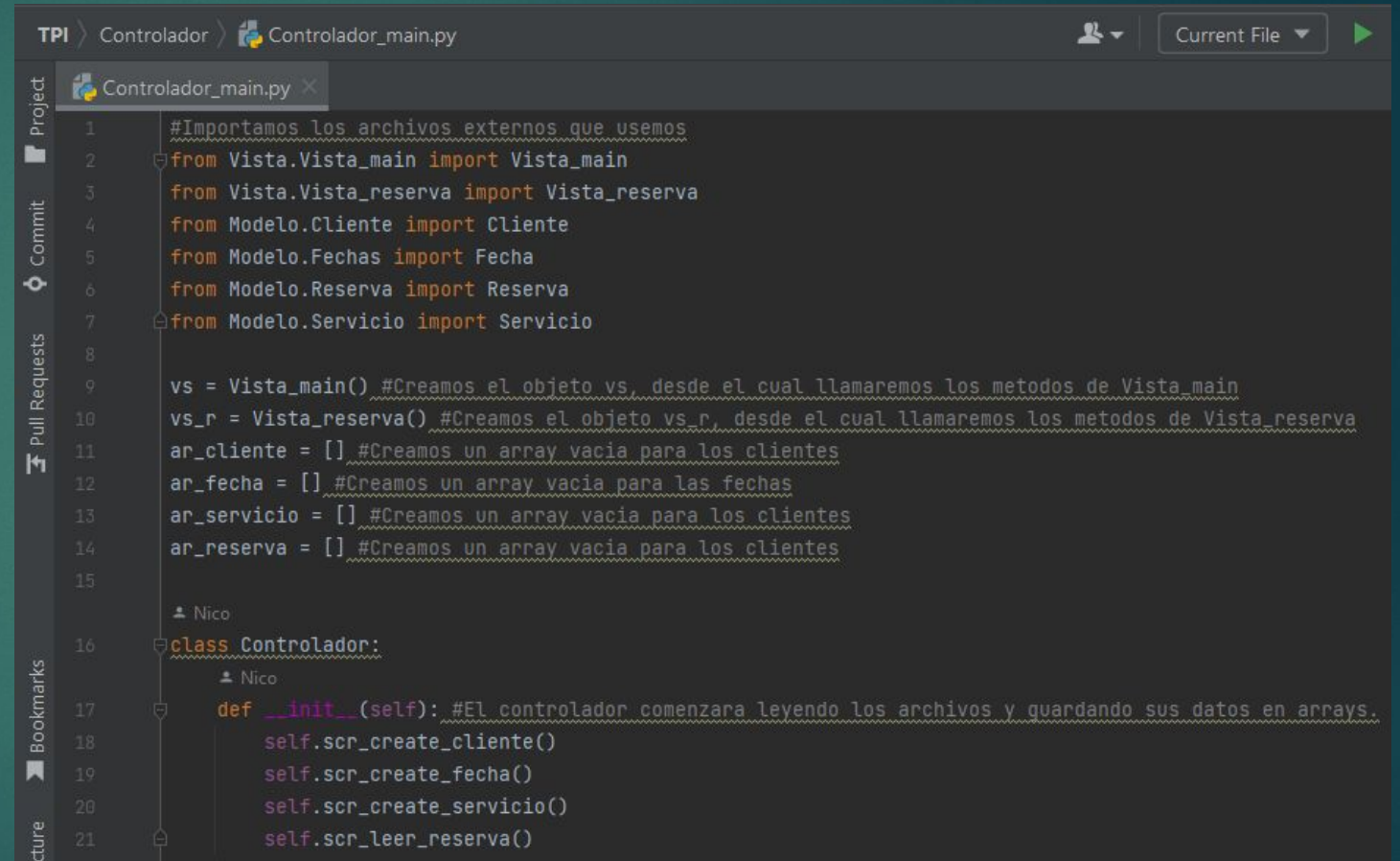
# CONTROLADOR

El controlador va a tener una sola clase, la clase Controlador

Esta clase es la espina dorsal de todo el proyecto, ya que conecta el main, los modelos y las vistas.

Al principio del código, antes de codificar la clase, se importan los archivos externos y se crean los objetos para la clase Vista\_menu y clase Vista\_menu. Después de eso se crean 4 vectores vacíos

El controlador va a leer los archivos y guardar sus datos en arrays. Para hacerlo invoca a 4 métodos locales



```
1  #Importamos los archivos externos que usamos
2  from Vista.Vista_main import Vista_main
3  from Vista.Vista_reserva import Vista_reserva
4  from Modelo.Cliente import Cliente
5  from Modelo.Fechas import Fecha
6  from Modelo.Reserva import Reserva
7  from Modelo.Servicio import Servicio
8
9  vs = Vista_main() #Creamos el objeto vs, desde el cual llamaremos los metodos de Vista_main
10 vs_r = Vista_reserva() #Creamos el objeto vs_r, desde el cual llamaremos los metodos de Vista_reserva
11 ar_cliente = [] #Creamos un array vacia para los clientes
12 ar_fecha = [] #Creamos un array vacia para las fechas
13 ar_servicio = [] #Creamos un array vacia para los clientes
14 ar_reserva = [] #Creamos un array vacia para los clientes
15
16 class Controlador:
17     def __init__(self): #El controlador comenzara leyendo los archivos y guardando sus datos en arrays.
18         self.scr_create_cliente()
19         self.scr_create_fecha()
20         self.scr_create_servicio()
21         self.scr_leer_reserva()
```



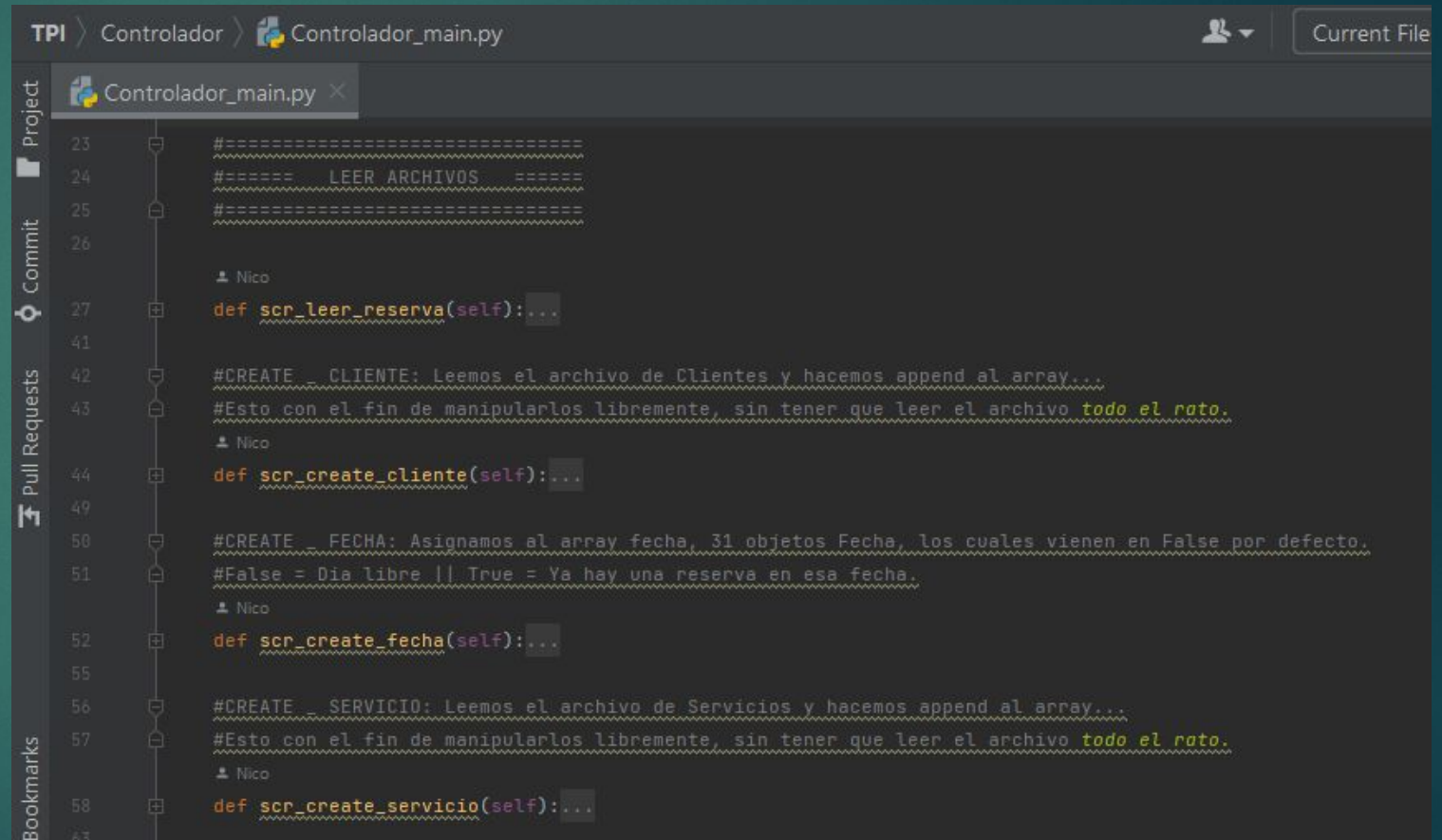
# CONTROLADOR

## Sección LEER ARCHIVOS

Dentro de esta sección se encuentran los métodos que van a leer los archivos txt del proyecto y crear objetos con los datos de estos mismos.

El código dentro de cada método se base en leer cada renglón del archivo txt, separar los atributos del renglón que se esta leyendo y crear el objeto.

Los objetos creados van a ser guardados en los vectores que se encuentran arriba de todo, después de los archivos importados

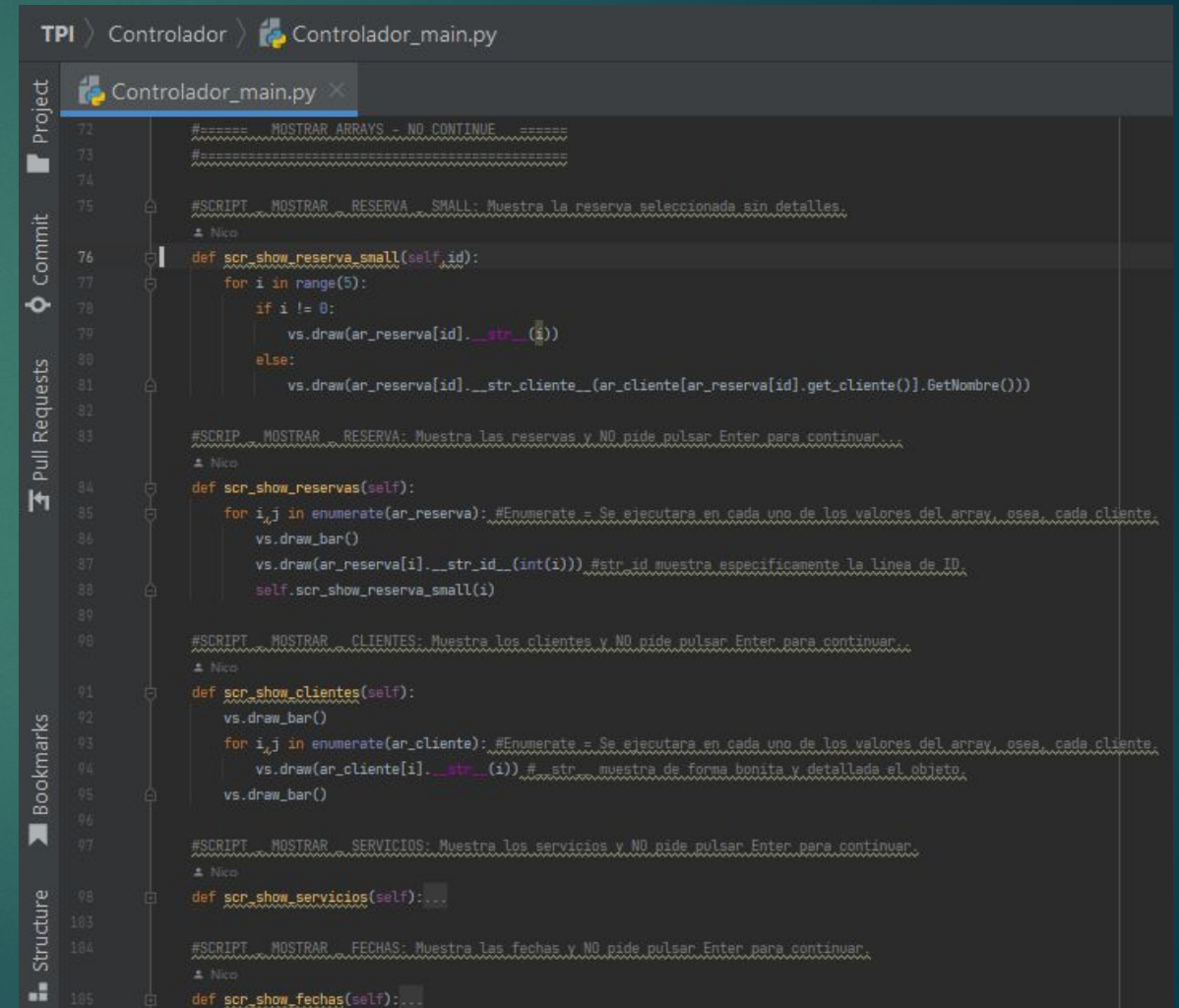


```
TPI > Controlador > Controlador_main.py
Controlador_main.py x
23  #=====
24  #===== LEER ARCHIVOS =====
25  #=====
26
27  def scr_leer_reserva(self):...
41
42  #CREATE _ CLIENTE: Leemos el archivo de Clientes y hacemos append al array...
43  #Esto con el fin de manipularlos libremente, sin tener que leer el archivo todo el rato.
44  def scr_create_cliente(self):...
49
50  #CREATE _ FECHA: Asignamos al array fecha, 31 objetos Fecha, los cuales vienen en False por defecto.
51  #False = Dia libre || True = Ya hay una reserva en esa fecha.
52  def scr_create_fecha(self):...
55
56  #CREATE _ SERVICIO: Leemos el archivo de Servicios y hacemos append al array...
57  #Esto con el fin de manipularlos libremente, sin tener que leer el archivo todo el rato.
58  def scr_create_servicio(self):...
63
```

# CONTROLADOR

## Sección MOSTRAR ARRAYS – NO CONTINUE

En muy resumida explicación, dentro de esta sección se encuentran los métodos que sirven para mostrar un array sin pedirle al usuario que pulse enter para continuar.



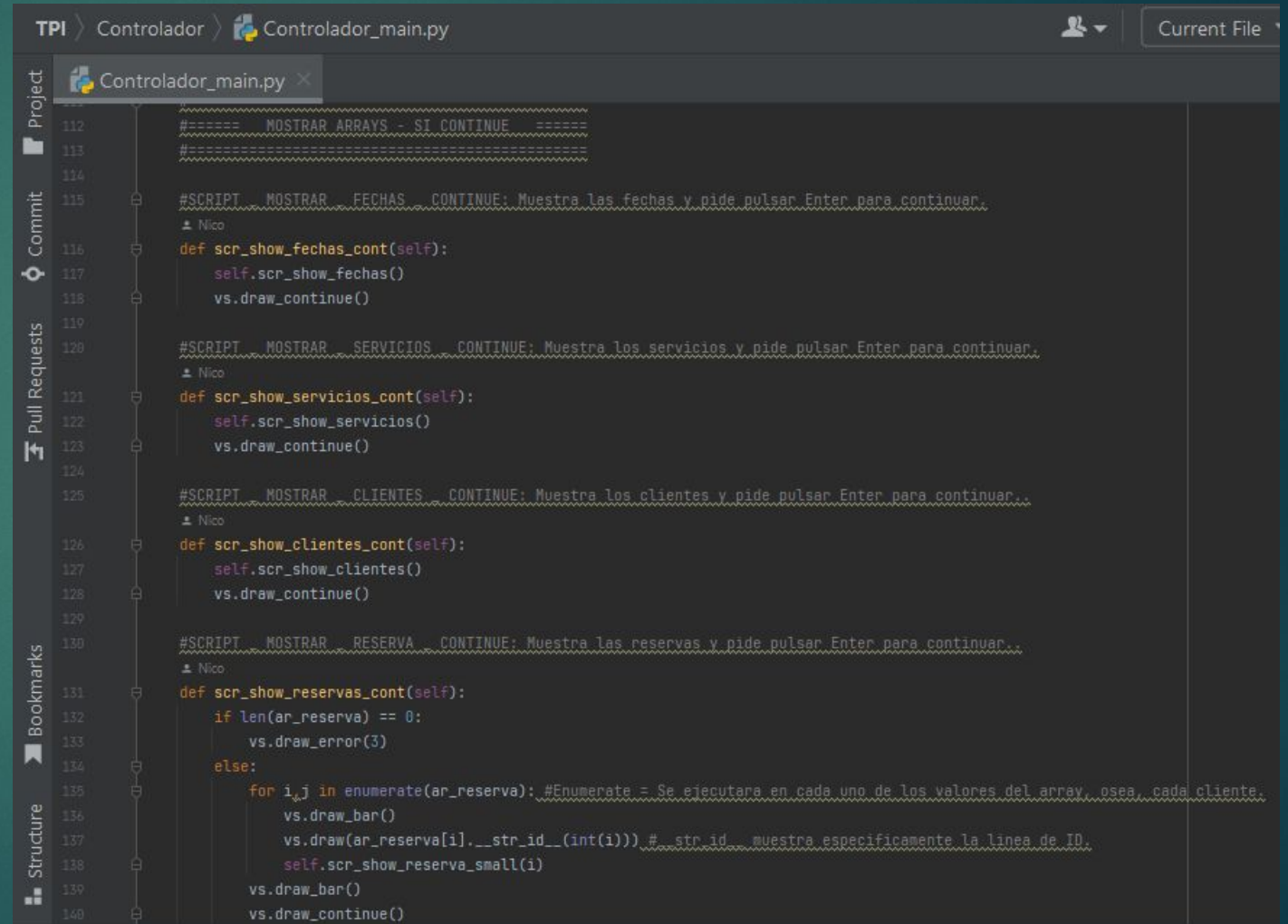
```
TPI > Controlador > Controlador_main.py

Controlador_main.py x
72  #===== MOSTRAR ARRAYS - NO CONTINUE =====
73  #=====
74
75  #SCRIPT MOSTRAR RESERVA SMALL: Muestra la reserva seleccionada sin detalles.
76  # Nico
77  def scr_show_reserva_small(self, id):
78      for i in range(5):
79          if i != 0:
80              vs.draw(ar_reserva[id].__str__(i))
81          else:
82              vs.draw(ar_reserva[id].__str_cliente__(ar_cliente[ar_reserva[id].get_cliente()].getNombre()))
83
84  #SCRIPT MOSTRAR RESERVA: Muestra las reservas y NO pide pulsar Enter para continuar...
85  # Nico
86  def scr_show_reservas(self):
87      for i, j in enumerate(ar_reserva): #Enumerate = Se ejecutara en cada uno de los valores del array, osea, cada cliente
88          vs.draw_bar()
89          vs.draw(ar_reserva[i].__str_id__(int(i))) #str_id muestra especificamente la linea de ID
90          self.scr_show_reserva_small(i)
91
92  #SCRIPT MOSTRAR CLIENTES: Muestra los clientes y NO pide pulsar Enter para continuar...
93  # Nico
94  def scr_show_clientes(self):
95      vs.draw_bar()
96      for i, j in enumerate(ar_cliente): #Enumerate = Se ejecutara en cada uno de los valores del array, osea, cada cliente
97          vs.draw(ar_cliente[i].__str__(i)) #str muestra de forma bonita y detallada el objeto
98          vs.draw_bar()
99
100  #SCRIPT MOSTRAR SERVICIOS: Muestra los servicios y NO pide pulsar Enter para continuar...
101  # Nico
102  def scr_show_servicios(self):...
103
104  #SCRIPT MOSTRAR FECHAS: Muestra las fechas y NO pide pulsar Enter para continuar...
105  # Nico
106  def scr_show_fechas(self):...
```

# CONTROLADOR

## Sección MOSTRAR ARRAYS – CONTINUE

En muy resumida explicación, dentro de esta sección se encuentran los métodos que sirven para mostrar un array y le piden al usuario que pulse enter para continuar.



```
112 #===== MOSTRAR ARRAYS - SI CONTINUE =====
113 #=====
114
115 #SCRIPT MOSTRAR FECHAS CONTINUE: Muestra las fechas y pide pulsar Enter para continuar.
116 # Nico
117 def scr_show_fechas_cont(self):
118     self.scr_show_fechas()
119     vs.draw_continue()
120
121 #SCRIPT MOSTRAR SERVICIOS CONTINUE: Muestra los servicios y pide pulsar Enter para continuar.
122 # Nico
123 def scr_show_servicios_cont(self):
124     self.scr_show_servicios()
125     vs.draw_continue()
126
127 #SCRIPT MOSTRAR CLIENTES CONTINUE: Muestra los clientes y pide pulsar Enter para continuar..
128 # Nico
129 def scr_show_clientes_cont(self):
130     self.scr_show_clientes()
131     vs.draw_continue()
132
133 #SCRIPT MOSTRAR RESERVA CONTINUE: Muestra las reservas y pide pulsar Enter para continuar..
134 # Nico
135 def scr_show_reservas_cont(self):
136     if len(ar_reserva) == 0:
137         vs.draw_error(3)
138     else:
139         for i,j in enumerate(ar_reserva): #Enumerate = Se ejecutara en cada uno de los valores del array, osea, cada cliente.
140             vs.draw_bar()
141             vs.draw(ar_reserva[i].__str_id__(int(i))) # str_id muestra especificamente la linea de ID.
142             self.scr_show_reserva_small(i)
143             vs.draw_bar()
144             vs.draw_continue()
```



# Controlador

## Sección CREAR RESERVA

Se le pide que:

1. Seleccione al cliente: busca la id del cliente
2. Seleccionar un dia: se le muestra los días y si están disponibles
3. Seleccionar los servicios: se le muestra los servicios que puede elegir
4. Finaliza la reserva: guarda los datos en un array

```
#=====
#=====  CREAR RESERVA  =====
#=====
def scr_create_reserva(self):
    try:
        ar_reserva.append(Reserva()) #Creamos la reserva
        #=====Seleccionar Cliente=====
        self.scr_show_clientes() #Mostramos al usuario los clientes.
        while True:
            self.int_idcliente = vs_r.draw_enter_cliente() #Pedimos al usuario ingrese la ID del cliente.
            if self.int_idcliente > -1 and self.int_idcliente < len(ar_cliente): #Se asegura que la ID in
                break
            else:
                vs.draw_error(8)
        ar_reserva[(len(ar_reserva) - 1)].set_cliente(self.int_idcliente)
        #=====Seleccionar Dia=====
        self.scr_show_fechas() #Mostramos al usuario los dias y si estan disponibles o no.
        while True:
            self.int_fecha = vs_r.draw_enter_fecha() #Pedimos al usuario ingresar el dia.
            if self.int_fecha > 0 and self.int_fecha < 31:
                break
            else:
                vs.draw_error(8)
        try:
            while ar_fecha[self.int_fecha].GetEstado() == True: #Aca leemos el array fecha segun el dia i
                self.int_fecha = self.scr_dia_cercano(self.int_fecha) #En caso de que el dia este ocupado
                if self.int_fecha != 31: #En caso de que no hayan dias disponibles luego de lo pedido.
                    vs_n.draw_error_dia_ocupado(self.int_fecha) #En caso de que el dia este ocupado
```



# Controlador

## Selección CAMBIAR STATUS

### Señar reserva:

1. Se le pide el ID de la reserva
2. Muestra que la reserva no esté cancelada y si está en la cola

### Cancelar reserva:

1. Se le pide el ID de la reserva a cancelar
2. Se mostrará el dinero que se le devolverá y pedirá una confirmación

```
#####
##### CAMBIAR STATUS #####
#####
def scr_seniar(self):
    try:
        self.scr_show_reservas() #Mostramos las reservas.
        vs.draw_bar()
        int_seniar = vs_r.draw_enter_seniar() #Pedimos al usuario que escriba la ID de la reserva a seniar.
        if ar_reserva[int_seniar].get_status() == "En Cola...": #Preguntamos si la reserva esta En Cola, para no
            vs.draw(ar_reserva[int_seniar].__str_senia__()) #En caso de que este En Cola, le mostramos al usuario
            if vs_r.draw_pregunta_seniar() == "si": #Pedimos al usuario que escriba si para confirmar.
                ar_reserva[int_seniar].set_status("Señado") #Cambiamos el estado de la reserva a Señado.
                vs_r.draw_seniar_terminado() #Printeamos que el señado fue terminado.
                vs.draw_continue()
        else:
            vs.draw_error(4) #En caso de que no este En Cola, se le avisara al usuario.
    except Exception:
        vs.draw_error(7)

def scr_cancelar(self):
    try:
        self.scr_show_reservas() #Mostramos las reservas.
        vs.draw_bar()
        int_cancel = vs_r.draw_enter_cancelar() #Pedimos al usuario que escriba la ID de la reserva a cancelar.
        if ar_reserva[int_cancel].get_status() == "En Cola..." or ar_reserva[int_cancel].get_status() == "Señado":
            vs.draw(ar_reserva[int_cancel].__str_cancelar__()) #En caso de que no este cancelado, le mostramos al usuario
            if vs_r.draw_pregunta_cancelar() == "si": #Pedimos al usuario que escriba si para confirmar.
                ar_reserva[int_cancel].set_status("Cancelado") #Cambiamos el estado de la reserva a Cancelado.
                vs_r.draw_cancelar_terminado() #Printeamos que el cancelamiento fue terminado.
```

# Controlador

## DIA CERCANO

Cuando se ingresa el día para reservar y ya se encuentre ocupado, el programa buscará el día más cercano que se detecte sin reservas, siempre y cuando se encuentre posteriormente al ingresado.

```
#=====
#===== DIA CERCANO =====
#=====
def scr_dia_cercano(self, day):
    while ar_fecha[day].GetEstado() == True:
        day += 1
        if day == 31:
            break
    return day
```