**CENTRO UNIVERSITÁRIO BELAS ARTES**

**ANÁLISE E DESENVOLVIMENTO DE SISTEMAS EAD**

**PADRÕES DE DESIGN - MÓDULO 2**

**FELIPE SCHAITEL RA: 24112424**

**ATIVIDADE 02: JOGO DO BINGO (UNIDADE 4)**

**FLORIANÓPOLIS**

**2024**

**1 ENUNCIADO**

Faça:

1) Crie uma parte para realizar o sorteio das bolas e exibir o número sorteado

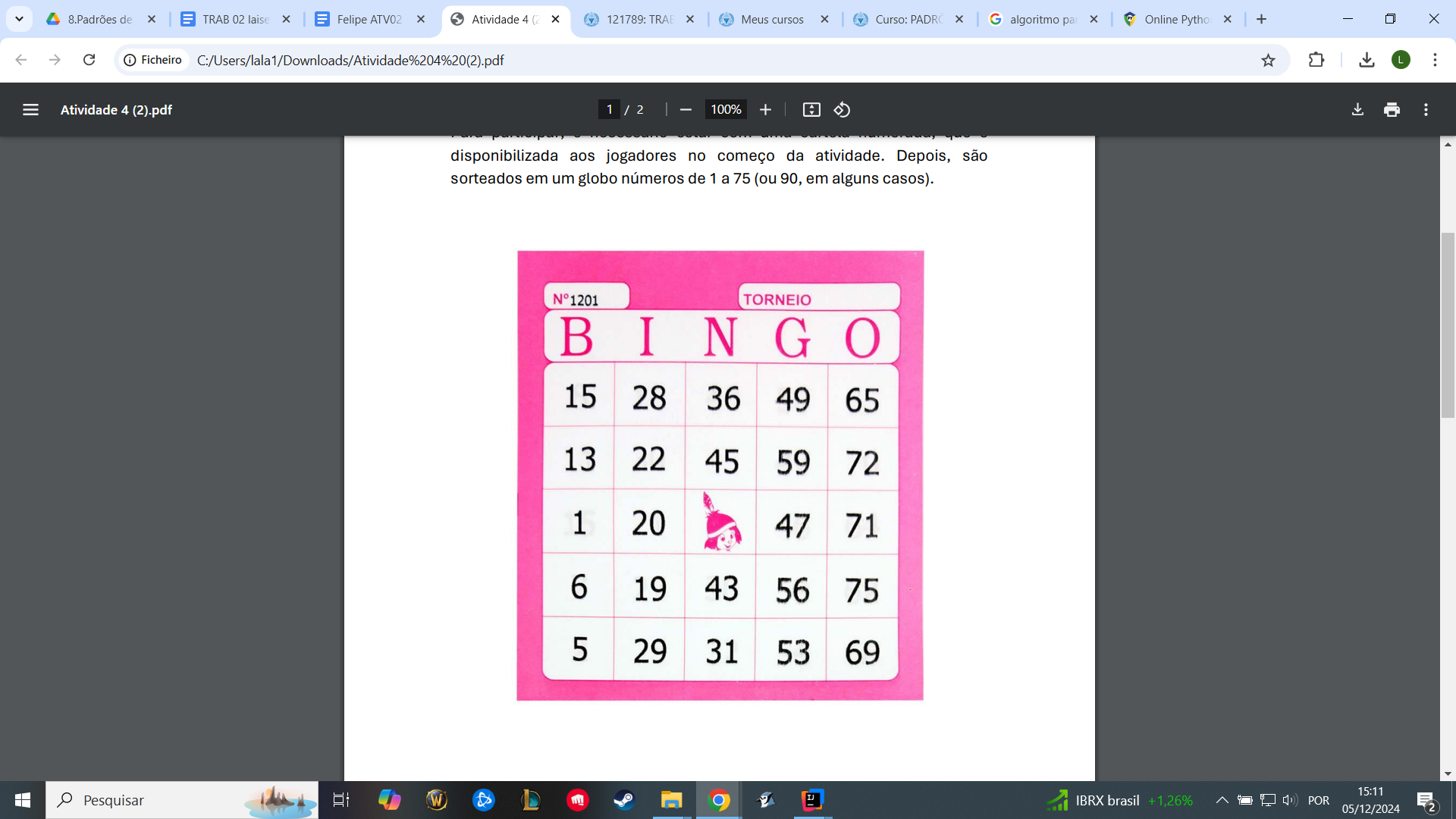
2) Crie uma cartela com 24 números respeitando as colunas e os números válidos

3) Simule uma jogada com a cartela criada.

Entregue:

1) Crie uma implementação do padrão observer para identificar se a cartela está cheia.

Imagem 01: cartela de bingo



Fonte: Material Didático

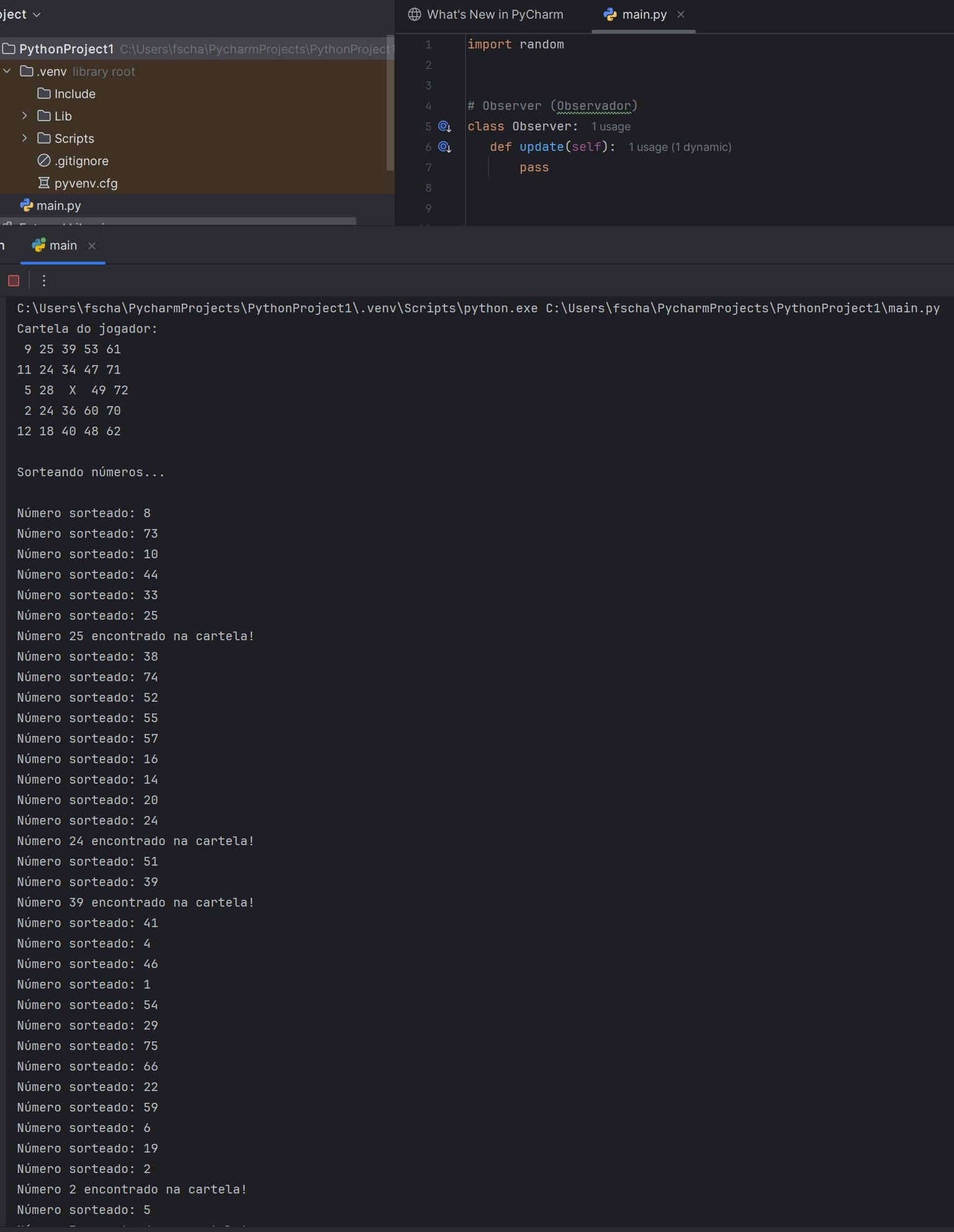
**INTRODUÇÃO**

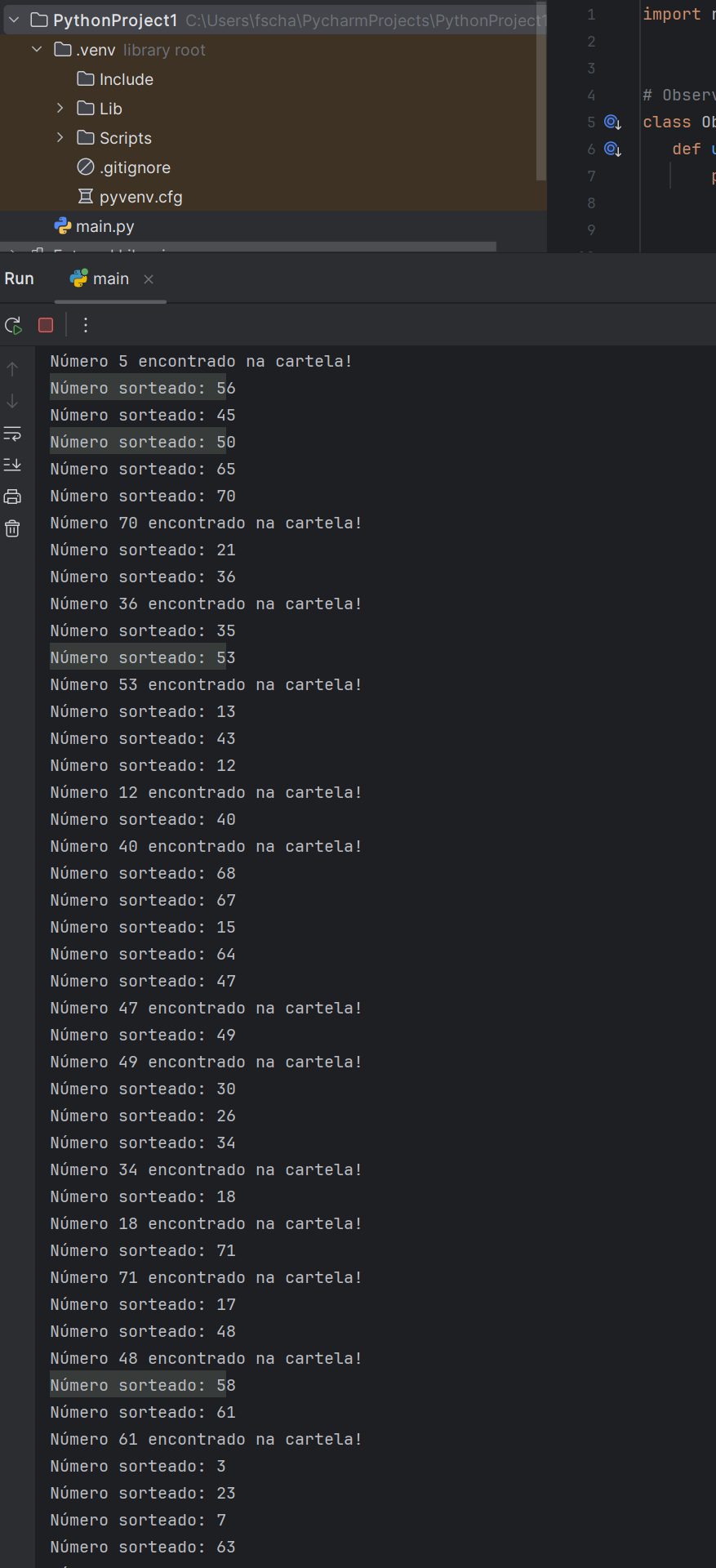
Na aula do dia 19 de novembro de 2924 o professor abordou sobre os padrões comportamentais. O padrão Observer é considerado padrão comportamental, vimos os criacionais e estruturais, que refere-se ao comportamento dos objetos. Outros padrões comportamentais além do Observer são: Responsabilidade encadeada, *comands*, *iterator*, *mediator*, *memento*, *state, strategy, visitor e template method.* São muitas opções para utilizar.

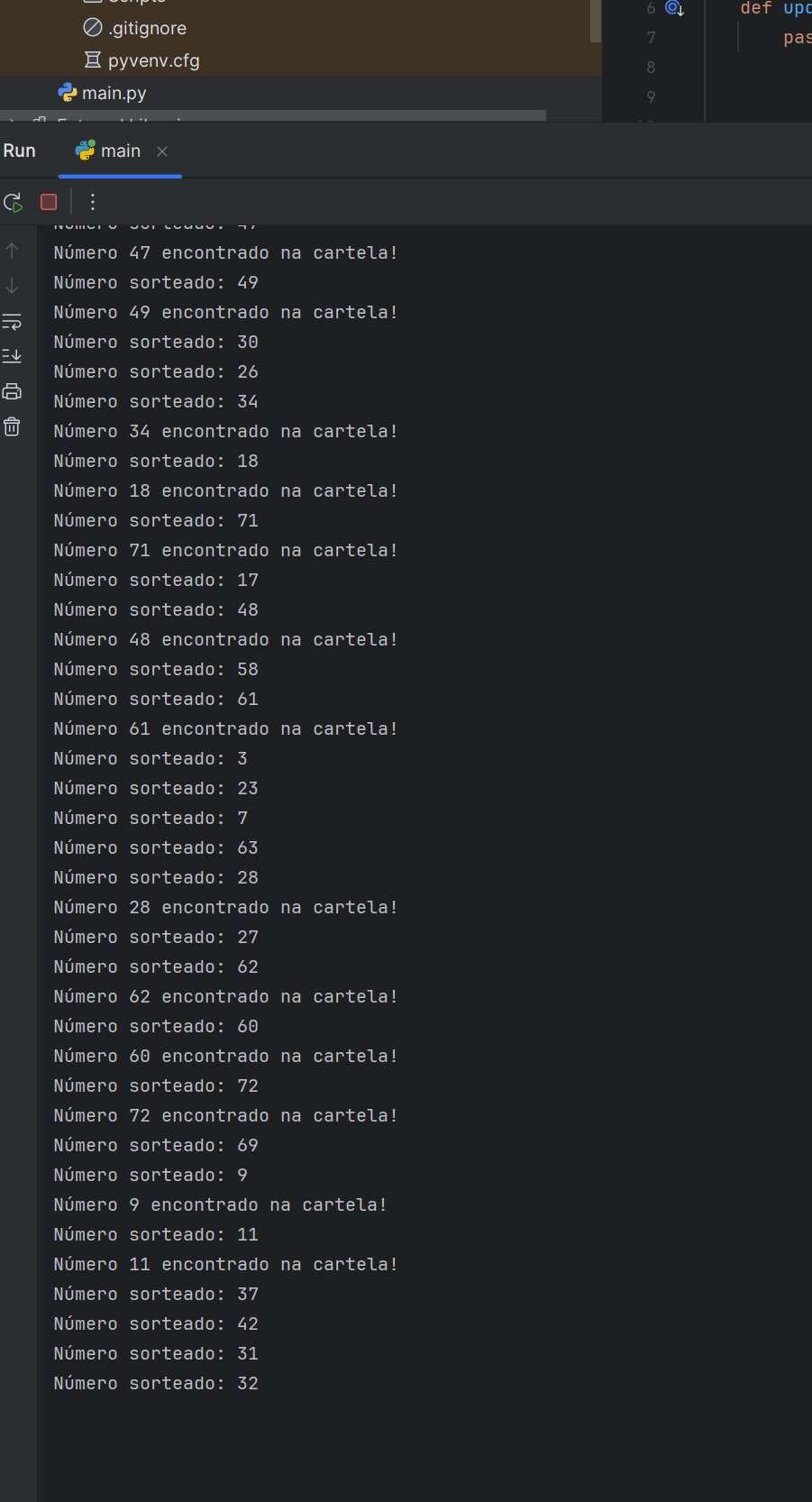
De acordo com *Refactoring.guru* o *Observer* permite definir um mecanismo de assinatura para notificar multiplos objetos sobre os eventos que acontecem com os objetos que eles estão observando. De acordo com o professor, Observer é o que usamos para gerenciar uma tela, para controlar os eventos que acontecem em uma tela e esse tipo de coisa.

O Observer vai ficar observando a cartela do bingo para quando ela ficar cheia avisar.

**2 SAÍDA DOS COMANDOS**

****

****

**3 CÓDIGO NA LINGUAGEM *PYTHON (IDE Pycharm)***

import random

class Observer:

def update(self):

pass

class Subject:

def \_\_init\_\_(self):

self.\_observers = []

def add\_observer(self, observer):

self.\_observers.append(observer)

def notify\_observers(self):

for observer in self.\_observers:

observer.update()

class Cartela(Subject):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.numeros = [[0]\*5 for \_ in range(5)]

self.marcados = [[False]\*5 for \_ in range(5)]

self.gerar\_cartela()

def gerar\_cartela(self):

for i in range(5):

for j in range(5):

if i == 2 and j == 2:

self.numeros[i][j] = -1

self.marcados[i][j] = True

else:

self.numeros[i][j] = random.randint(1, 15) + j \* 15

self.marcados[i][j] = False

def marcar\_numero(self, numero):

for i in range(5):

for j in range(5):

if self.numeros[i][j] == numero:

self.marcados[i][j] = True

if self.cartela\_cheia():

self.notify\_observers()

return True

return False

def cartela\_cheia(self):

for i in range(5):

for j in range(5):

if not self.marcados[i][j]:

return False

return True

def imprimir\_cartela(self):

for i in range(5):

for j in range(5):

print(f"{' X ' if self.numeros[i][j] == -1 else f'{self.numeros[i][j]:2}'}", end=" ")

print()

class Globo:

def \_\_init\_\_(self):

self.numeros\_sorteados = set()

def sortear\_numero(self):

while True:

numero = random.randint(1, 75)

if numero not in self.numeros\_sorteados:

self.numeros\_sorteados.add(numero)

return numero

class Jogador(Observer):

def \_\_init\_\_(self, nome):

self.nome = nome

def update(self):

print(f"Bingo! O jogador {self.nome} completou a cartela!")

def main():

cartela = Cartela()

jogador = Jogador("João")

cartela.add\_observer(jogador)

globo = Globo()

print("Cartela do jogador:")

cartela.imprimir\_cartela()

print("\nSorteando números...\n")

while True:

numero = globo.sortear\_numero()

print(f"Número sorteado: {numero}")

if cartela.marcar\_numero(numero):

print(f"Número {numero} encontrado na cartela!")

if cartela.cartela\_cheia():

break

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Referências bibliográficas:**

1. Disponivel em: <https://refactoring.guru/pt-br/design-patterns/observer> <05/12/2024>
2. Documentação Python Disponível em: <https://docs.python.org/3/> <05/12/2024>
3. Material didático da disciplina;
4. GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1. ed. Boston: Addison-Wesley, 1995.
5. O'REILLY, Eric Freeman; RUSH, Elisabeth. *Head First Design Patterns*. 1. ed. São Paulo: O'Reilly Media, 2004.