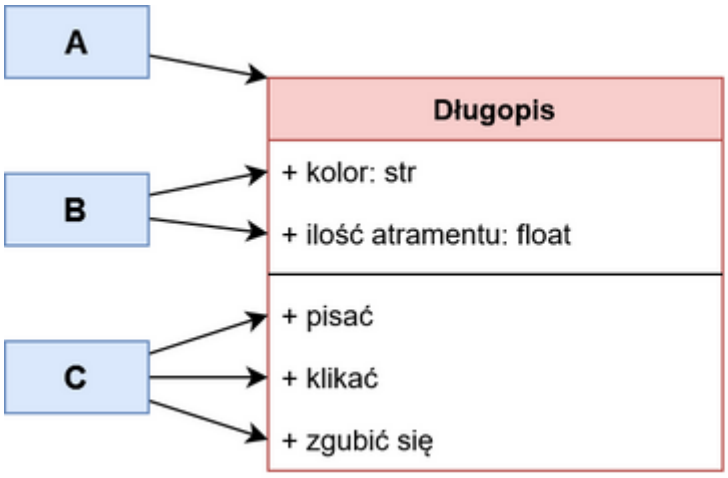
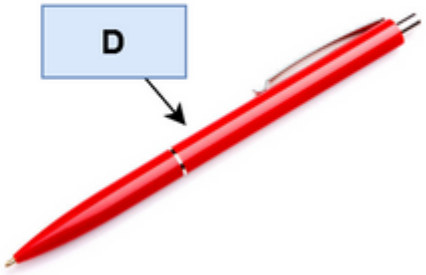
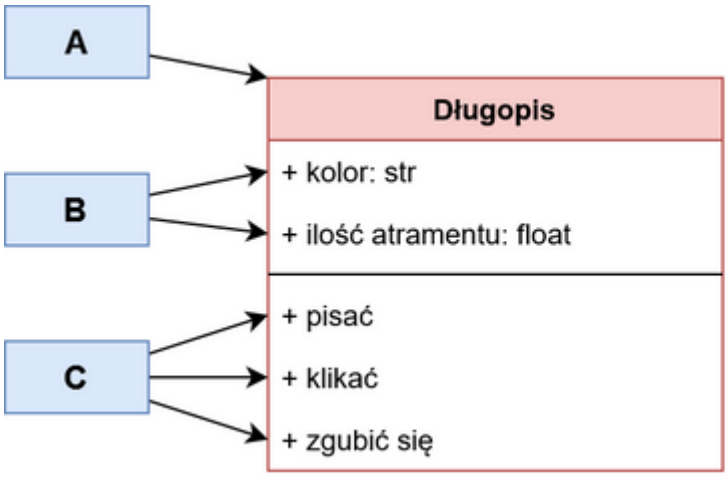
1) Sprawdzenie obecności😁

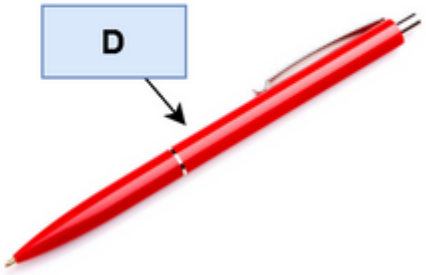
A. Jestem i będę wykonywał zadania ❤️
B. Jestem, ale tylko oglądam 😝



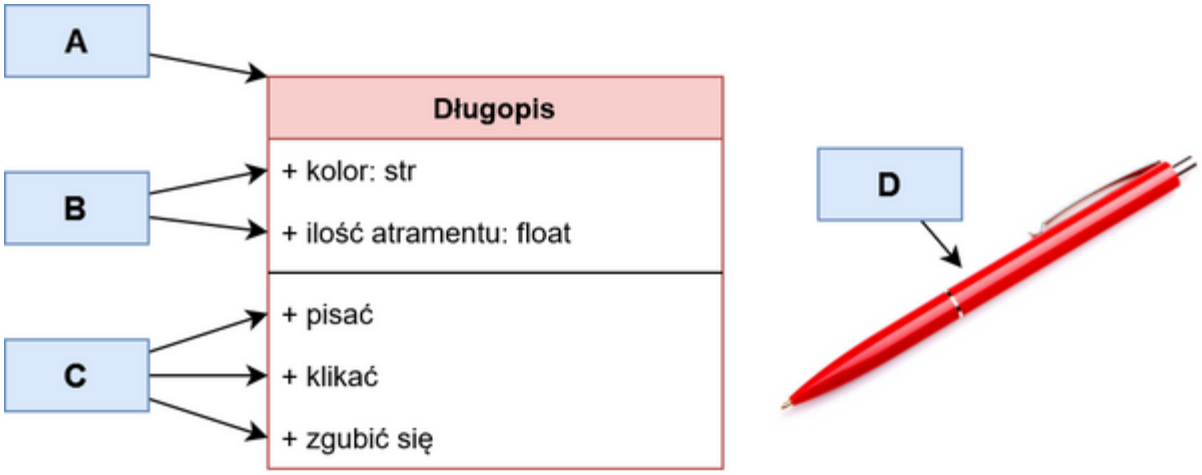


2)
Wpisz jedną literkę w poniższym polu:
_____ - klasa

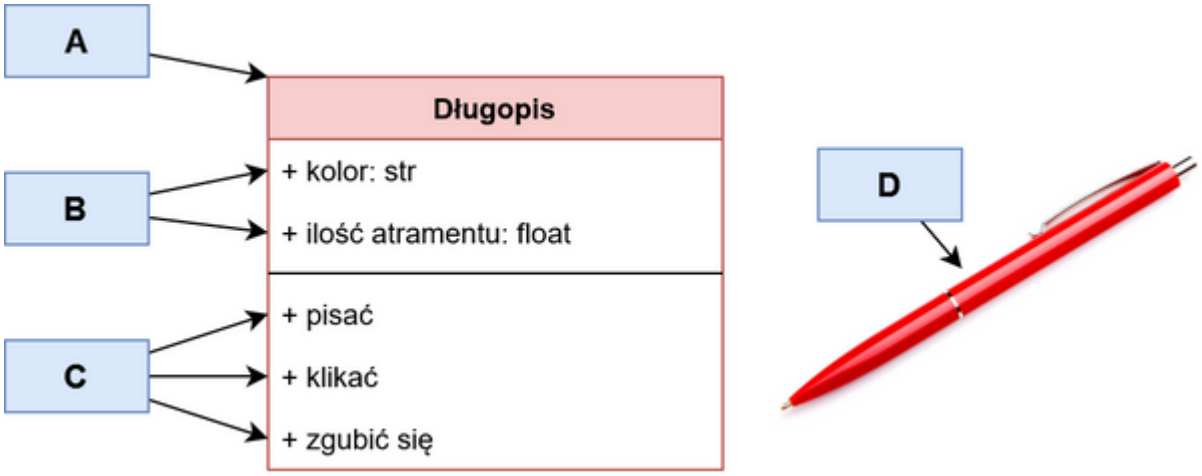




3)
Wpisz jedną literkę w poniższym polu:
_____ - obiekt



4)
Wpisz jedną literkę w poniższym polu:
_____ - metoda



5)
Wpisz jedną literkę w poniższym polu:
_____ - atrybut

6) Klasa `Długopis`:

```
class Długopis:
    kolor: str
    ilość_atramentu: float

    def __init__(self, kolor: str, ilość_atramentu: float) -> None:
        self.kolor = kolor
        self.ilość_atramentu = ilość_atramentu
        print(f"Długopis o kolorze {self.kolor}. Zostało
```

```
{self.ilość_atramentu:.0%} atramentu.")

    def pisz(self) -> str:
        return "Piszę!"

    def zgub_się(self) -> str:
        return "Zgubiłem się!"
```

A. Prawie zasnąłem, szybciej prowadź te zajęcia!
B. Czemu nie `Ołówek`?
C. Gdzie się tak śpieszysz? Zwolnij trochę!

7) Klasa `DługopisZeSprężyną`:

```
class DługopisZeSprężyną(Długopis):
    włączony: bool = False

    def kliknij(self) -> None:
        self.włączony ^= True
        print(f"Klik-klik! Długopis jest teraz {'włączony' if
self.włączony else 'wyłączony'}.")
```

Klasa `DługopisZeŚciągą`:

```
class DługopisZeŚciągą(Długopis):
    tekst_ściągi: str = ""

    def dodaj_tekst_do_ściągi(self, tekst: str) -> None:
        self.tekst_ściągi += tekst
        print("Dodano tekst do ściągi.")
```

Klasa `SuperDługopis`:

```
class SuperDługopis(DługopisZeSprężyną, DługopisZeŚciągą):
    pass
```

Jak oceniasz trudność tego materiału?

A. Trudny
B. Średni
C. Łatwy

8) Zaznacz wszystkie poprawne odpowiedzi:

Choose all that apply:

A. Klasa `Długopis` jest wyłącznie klasą bazową.
B. Klasa `DługopisZeSprężyną` jest wyłącznie klasą potomną.
C. Klasa `DługopisZeŚciągą` jest zarówno klasą bazową, jak i klasą potomną.
D. Klasa `DługopisZeSprężyną` jest klasą bazową.
E. Klasa `SuperDługopis` jest klasą bazową.

9) Tworzenie własnej klasy

Poniżej został podany działający szablon tworzenia dowolnej klasy.

```
class NazwaKlasy:
    atrybut_1: int
    atrybut_2: str

    def __init__(self, param_1, param_2):
        # Przypisanie atrybutom wartości parametrów:
        self.atrybut_1 = param_1
        self.atrybut_2 = param_2

    def metoda_1(self, parametr_1, parametr_2):
        # Jakieś działania
        pass

objekt = NazwaKlasy(param_1=12, param_2="Wooow!")
print(objekt.atrybut_1)
print(objekt.atrybut_2)
```

Na podstawie powyższego szablonu stwórz klasę `Vector`, którego atrybutami będą wartości `x` i `y`. Do konstruktora klasy dodaj kod, który przypisze atrybutom klasy podane wartości. Wykonaj poniższy kod, a w polu wpisz wynik poniższego kodu:

```
vec = Vector(2, 3)
print(vec.x + vec.y ** vec.x)
```

---

10) Kod do poprzedniego zadania:

```
class Vector:
    x: float
    y: float

    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def __repr__(self):
        return f"Vector({self.x}, {self.y})"
```

Stwórz "magiczną" metodę `__len__` dla klasy `Vector`, która zwróci ilość wymiarów wektora. Sprawdź poprawność napisania metody wykonując poniższe polecenie. Jego wynik podaj w odpowiednim polu.
`len(Vector(2, 4))`

---

11) Kod do poprzedniego zadania:

```
class Vector:
    # ...
    # Ten sam kod, co poprzednio
    # ...

    def __len__(self):
        return 2
```

Stwórz własność (property) `length` klasy `Vector`, która zwróci długość wektora. Sprawdź poprawność napisania metody wykonując poniższe polecenie. Jego wynik podaj w odpowiednim polu.
`round(Vector(2, 4).length, 2)`

---

12) Kod do poprzedniego zadania:

```
class Vector:
    # ...
    # Ten sam kod, co poprzednio
    # ...

    @property
    def length(self):
        return math.sqrt(self.x ** 2 + self.y ** 2)
```

Stwórz "magiczne" metody `__add__`, i `__mul__` dla klasy `Vector`. Metoda mnożenia powinna umieć pracować zarówno z innymi wektorami, jak i z liczbami. W celu sprawdzania tupy zmiennej możemy użyć funkcji `isinstance(object, class)`. Sprawdź poprawność napisania metody wykonując poniższe polecenie. Jego wynik podaj w odpowiednim polu.
`Vector(2, 5) * (Vector(10, 14) + Vector(3, 2) * (-2))`

---

13) Kod do poprzedniego zadania:

```
class Vector:
    # ...
    # Ten sam kod, co poprzednio
    # ...

    def __add__(self, obj):
        if isinstance(obj, Vector):
            return Vector(self.x + obj.x, self.y + obj.y)
        else:
            raise TypeError(f"Nie wiem jak dodać {type(self)} do
{type(obj)}.")

    def __mul__(self, obj):
        if isinstance(obj, Vector):
            return self.x * obj.x + self.y * obj.y
        elif isinstance(obj, int) or isinstance(obj, float):
            return Vector(self.x * obj, self.y * obj)
        else:
            raise TypeError(f"Nie wiem jak przemnożyć {type(self)}
przez {type(obj)}.")
```

Kiedy ostatni raz grałeś w gry komputerowe?

A. Dzisiaj
B. Wczoraj
C. W ciągu tygodnia
D. Gdzieś w tym semestrze
E. Przepraszam, ale robiłem CO?

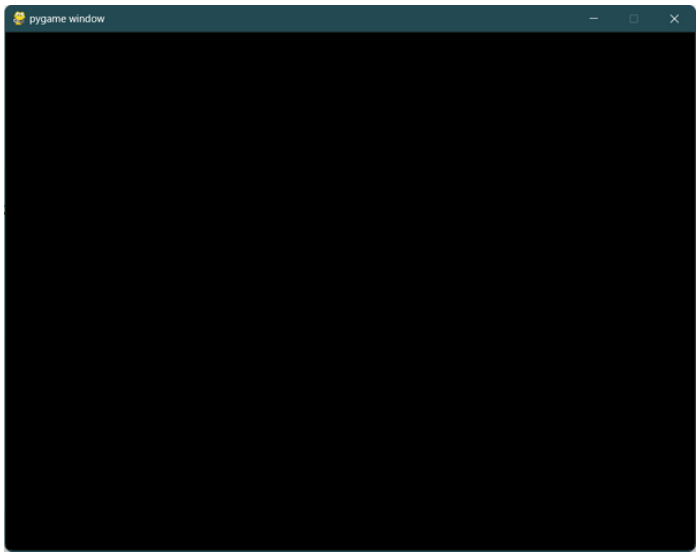
14) Wpisz poniższe polecenie w terminalu (otworzyć można skrótem `Ctrl+`):

```
pip install pygame
```

Za pomocą poniższego linku pobrać plik `Space-Invaders-Pygame.zip`, rozpakować go i zamieścić folder `Space-Invaders-Pygame` w folderze roboczym dzisiejszych zajęć.
https://github.com/attreyabhatt/Space-Invaders-Pygame/archive/refs/heads/master.zip
Po wykonaniu wyżej podanych czynności można uruchomić plik `main.py` (ale poprzednio **wyłączcie dźwięk na komputerze**).
Ile statków kosmitów zniszczyłeś?

_____

15) Pod poniższym linkiem znajduje się szablon dla gry Snake. Pobierz plik `snake_template.py` i zamieść go w folderze roboczym dzisiejszych zajęć.
https://prz1-my.sharepoint.com/:u:/g/personal/166731_o365_prz_edu_pl1/Eac7cdCVTEpBkStOmrrqd_sBCCxC9nrYtBObgph_RoxJog?e=RTOvPV
Dla sprawdzenia poprawności pobrania pliku, skrypt `snake_template.py` powinien uruchomić się bez najmniejszego problemu. Powinno wyskoczyć po prostu czarne okienko:



Czy kiedykolwiek pisałeś/napisałeś własną grę komputerową?

A. Tak, napisałem.
B. Tak, pisałem. Ale...
C. Nie, piszę po raz pierwszy.

16) Tworzymy klasę `Movements`, która będzie odpowiadała za wszystkie przemieszczenia naszego węża. Dla sprawdzenia poprawności rozwiązania podaj wynik poniższego polecenia w odpowiednim polu:
`print(Movements().opposite_to("left"))`

---

17) Kod do poprzedniego zadania (dodajemy go w sekcji `# (sekcja 1) Przygotowanie wszystkich klas`):

```
class Movements:
    directions = {
        "up": (0, -SEGMENT_SIZE),
        "down": (0, SEGMENT_SIZE),
        "left": (-SEGMENT_SIZE, 0),
        "right": (SEGMENT_SIZE, 0),
    }

    def to(self, direction: str):
        return self.directions.get(direction)

    def opposite_to(self, direction: str):
        direction_id = DIRECTIONS.index(direction) + 1
        opposite_direction = DIRECTIONS[-direction_id]
        return self.directions.get(opposite_direction)
```

Stworzymy teraz klasę bazową pojedynczego elementu węża `SnakeBodySection` i klasę głowy węża `SnakeHead`. Wynikiem poprawnego wykonania zadania pokazanego przez prowadzącego jest otrzymanie pojedynczego segmentu węża, którym można sterować z klawiatury.
Udało się?

A. Tak
B. Prawie
C. Nie

18) Kod do poprzedniego zadania:
1. W sekcji `# (sekcja 1) Przygotowanie wszystkich klas`:

```
class SnakeBodySection(pygame.sprite.Sprite):
    direction: str
    previous_direction: str

    def __init__(self, *groups: pygame.sprite.AbstractGroup):
        super().__init__(*groups)
        self.surface = pygame.Surface(size=(SEGMENT_SIZE,
SEGMENT_SIZE), flags=pygame.SRCALPHA)
        self.surface.fill(SNAKE_BODY_COLOR)


class SnakeHead(SnakeBodySection):

    def __init__(self, *groups: pygame.sprite.AbstractGroup):
        super().__init__(*groups)
        self.direction = self.previous_direction =
self.next_direction = DIRECTIONS[random.randint(0, 3)]
        self.surface.fill(SNAKE_HEAD_COLOR)

        width_positions = SCREEN_WIDTH // SEGMENT_SIZE
        height_positions = SCREEN_HEIGHT // SEGMENT_SIZE
        self.rect = self.surface.get_rect()
        self.rect.move_ip(
            random.randint(width_positions // 4, width_positions -
width_positions // 4) * SEGMENT_SIZE,
            random.randint(height_positions // 4, height_positions -
height_positions // 4) * SEGMENT_SIZE
        )

    def update_direction(self, pressed_keys):
        if pressed_keys[K_UP] and not self.direction == "down":
            self.next_direction = "up"
        elif pressed_keys[K_DOWN] and not self.direction == "up":
            self.next_direction = "down"
        elif pressed_keys[K_LEFT] and not self.direction == "right":
            self.next_direction = "left"
        elif pressed_keys[K_RIGHT] and not self.direction == "left":
            self.next_direction = "right"

    def update(self, *args, **kwargs):
        self.previous_direction = self.direction
        self.direction = self.next_direction
        self.rect.move_ip(Movements().to(self.direction))
```

2. W sekcji `# (sekcja 2) Przygotowanie wszystkich obiektów`:

```
all_sprites = pygame.sprite.Group()
head = SnakeHead(all_sprites)
```

3. W sekcji `# (sekcja 3) Opracowanie wszystkich wydarzeń`:

```
head.update_direction(pygame.key.get_pressed())
```

4. W sekcji `# (sekcja 4) Opracowanie nowej klatki`:

```
head.update()
```

5. W sekcji `# (sekcja 5) Aktualizacja obrazu`:

```
for entity in all_sprites:
    screen.blit(entity.surface, entity.rect)
```

Teraz chcemy stworzyć dodatkowe segmenty węża. W tym celu tworzymy klasę `SnakeTail`, której obiektami będą wszystkie segmenty węża oprócz głowy. Po skutecznym wykonaniu zadań podanych przez prowadzącego, w grze powinien tworzyć się wąż złożony z pięciu segmentów.
Udało się?

A. Tak
B. Prawie
C. Nie

19) Kod do poprzedniego zadania:
1. W sekcji `# (sekcja 1) Przygotowanie wszystkich klas`:

```
class SnakeTail(SnakeBodySection):

    def __init__(self, previous_segment: SnakeBodySection, *groups:
pygame.sprite.AbstractGroup):
        super().__init__(*groups)
        self.previous_segment = previous_segment
        self.direction = self.previous_direction =
self.previous_segment.direction
        self.rect =
self.previous_segment.rect.move(Movements().opposite_to(self.directio
n))

    def update(self, *args, **kwargs):
        self.previous_direction = self.direction
        self.direction = self.previous_segment.previous_direction
        self.rect.move_ip(Movements().to(self.direction))
```

2. W sekcji `# (sekcja 2) Przygotowanie wszystkich obiektów` usuwamy poprzednie napisaną linijkę `head = SnakeHead(all_sprites)`, natomiast dodajemy poniższy kod:

```
snake_tail = pygame.sprite.Group()
snake_body = pygame.sprite.Group()
new_snake_segment = head = SnakeHead(snake_body, all_sprites)
for i in range(INITIAL_SNAKE_LENGTH - 1):
    new_snake_segment = SnakeTail(new_snake_segment, snake_tail,
snake_body, all_sprites)
```

3. W sekcji `# (sekcja 4) Opracowanie nowej klatki`:

```
snake_tail.update()
```

Prawie mamy gotową grę. Zostaje tylko dodać żywność i warunki przegrania. Zacznijmy od żywności. Dlatego tworzymy klasę `Food`. Poprawne wykonanie zadań powinno tworzyć nowe jabłko za każdym razem, gdy poprzednie było zjedzone.
Udało się?

A. Tak
B. Prawie
C. Nie

20) Kod do poprzedniego zadania:
1. W sekcji `# (sekcja 1) Przygotowanie wszystkich klas`:

```
class Food(pygame.sprite.Sprite):
    def __init__(self, snake: pygame.sprite.AbstractGroup, *groups:
pygame.sprite.AbstractGroup):
        super().__init__(*groups)
        self.snake = snake
        self.surface = pygame.Surface(size=(SEGMENT_SIZE,
SEGMENT_SIZE), flags=pygame.SRCALPHA)
        self.surface.fill(FOOD_COLOR)
        self.update()

    def update(self, *args, **kwargs):
        while True:
            self.rect = self.surface.get_rect(
                left=random.randint(0, SCREEN_WIDTH // SEGMENT_SIZE -
1) * SEGMENT_SIZE,
                top=random.randint(0, SCREEN_HEIGHT // SEGMENT_SIZE -
1) * SEGMENT_SIZE
            )
            if not pygame.sprite.spritecollideany(self, self.snake):
                break
```

2. W sekcji `# (sekcja 2) Przygotowanie wszystkich obiektów`:

```
food_group = pygame.sprite.Group()
food = Food(snake_body, food_group, all_sprites)
```

3. W sekcji `# (sekcja 4) Opracowanie nowej klatki` pomiędzy linijkami `head.update()` i `snake_tail.update()`:

```
if pygame.sprite.spritecollideany(head, food_group):
            new_snake_segment = SnakeTail(new_snake_segment,
snake_tail, snake_body, all_sprites)
            food.update()
            frames_per_move -= ACCELERATION
```

Zostało teraz tylko dodać warunki zakończenia gry. Będzie ich u nas dwa: wąż najeżdża na siebie bądź wąż wjeżdża w ścianę. Po wykonaniu zadań prowadzącego powinniśmy otrzymać już całkiem działającą grę.
Udało się?

A. Tak
B. Prawie
C. Nie

21) Kod do poprzedniego zadania:
W sekcji `# (sekcja 4) Opracowanie nowej klatki` po linijce `snake_tail.update()`:

```
if pygame.sprite.spritecollideany(head, snake_tail):
            game_is_running = False
        if any((head.rect.top < 0,
                head.rect.left < 0,
                head.rect.bottom > SCREEN_HEIGHT,
                head.rect.right > SCREEN_WIDTH,
                )):
            game_is_running = False
```

Za pomocą poniższego linku możecie pobrać finalną wersję gry z dodanymi teksturami:
https://github.com/knmlprz/python-tutorial/archive/refs/heads/4-snake-game.zip

Jakie macie wrażenia od dzisiejszych zajęć? (opcjonalnie)

22) Czy pójdziecie na imprezę integracyjną koła?

A. Tak, chętnie przyjdę
B. Chciałbym, ale niestety nie dam rady
C. Nie, imprezy to nie moja pasja

Key:
1.
2. A
3. D
4. C
5. B
6.
7.
8. A, C, D
9. 11
10. 2
11. 4.47
12. 58
13.
14.
15.
16. (25, 0)
17.
18.
19.
20.
21.
22. A