Zadanie 1. Utwórz plik o zawartości

```
nagłówek
linia 1
linia 2
linia 3
```

i nazwij go `plik.txt`. Otwórz plik funkcją `open` (`https://docs.python.org/3/library/functions.html#open`) i przeczytaj go linia po linii wypisując zawartość na ekran. Zapisz w pliku `wyjscie.txt` liczbę przeczytanych linii (policz linie odpowiednimi instrukcjami w Pythonie). Zapoznaj się z funkcjami `read`, `readline`, `readlines`, `write` i `seek`.

`https://docs.python.org/3.5/tutorial/inputoutput.html#reading-and-writing-files`.

Zadanie 2. Dodaj obsługę błędów. W momencie gdy otwierany plik nie istnieje, rzucany jest wyjątek `OSError`. Łap go za pomocą konstrukcji `try/except` i wypisuj treść komunikatu wyjątku w przypadku jego złapania. Zmień nazwę otwieranego pliku tak, aby przetestować łapanie wyjątku.

`https://docs.python.org/3/tutorial/errors.html#handling-exceptions`

Zadanie 3. Zmodyfikuj kod tak, aby plik był poprawnie zamykany. Wołaj metodę `close` tylko w przypadku poprawnego otwarcia pliku. Jeśli wyjątek nie został rzucony (i tylko wtedy) wypisuj na ekran komunikat o braku błędów.

Zadanie 4. Zdefiniuj własny wyjątek o nazwie `OutOfLuck`. Rzucaj go tuż przed otwarciem pliku jeśli losowa liczba z zakresu of 0 do 3 jest równa 0. `https://docs.python.org/3/library/random.html#random.randint` Łap ten wyjątek w tym samym bloku try co `OSError`. W przypadku jego złapania wypisuj na ekran „A to pech!” Dodaj również blok `except` łapiący wszystkie pozostałe wyjątki i wypisujący przy tym „Nieznany wyjątek”. Spróbuj rzucić nieobsługiwany wyjątek. Jeśli nie pojawił się żaden wyjątek wypisz „Bez wyjątków” (wykorzystaj blok `else` dla `try`).

`https://docs.python.org/3/tutorial/errors.html#user-defined-exceptions`

Zadanie 5. Zmień obsługę pliku z poprzedniego zadania na konstrukcję `with/as`.

`https://docs.python.org/3/tutorial/errors.html#predefined-clean-up-actions`

Zadanie 6. Przyjrzyj się poniższemu fragmentowi kodu i sprawdź jak działa.

```
class Kwadraty:
    def __init__(self):
        self.num = 0

    def __iter__(self):
        return self
```

```
    def __next__(self):
        self.num += 1
        return (self.num-1)**2

for i in Kwadraty():
    print(i)
    if i > 10000:
        break
```

Zmodyfikuj iterator `Kwadraty` tak, aby zaczynał liczyć nie od zera, lecz od liczby przekazanej do konstruktora klasy. Wydziel wyrażenie zwracane w metodzie `__next__` do osobnej metody o nazwie `f`. Zmodyfikuj metodę `f` tak, aby zwracała nie kwadrat liczby, ale `(n%4)`-ty element listy `['a', 'b', 'c', 'd']` (gdzie `n` jest argumentem metody `f`).

`https://docs.python.org/3/tutorial/classes.html#iterators`

Zadanie 7. Zmodyfikuj kod z zadania 5 tak, aby iteracja po kwadratach była przerywana po zwróceniu `n` wartości, gdzie `n` jest podawane do konstruktora klasy.

Zadanie 8. Zamień kod z poprzedniego zadania na iterator zaimplementowany za pomocą generatora. Szkielet implementacji:

```
def f(n):
    return ???

def gen(k):
    for i in ???:
        yield ???

for z in gen(10):
    print(z)
```