MARTA POTOCKA

PYTHON
50 PYTAŃ Z ROZMÓW KWALIFIKACYJNYCH

Przygotuj się do rozmowy i dostań pracę jako Python Developer

JAKZOSTACPROGRAMISTA.PL

Autor: Marta Potocka
marta.m.potocka@gmail.com

Tytuł: “Python – 50 pytań z rozmów kwalifikacyjnych.”

Fotografia na okładce: Canva (licencja CC).
Fotografia na stronie o autorce: własność prywatna autorki.

Redakcja: Dawid Wiktorski
dawidwiktorski@gmail.com

Spis treści

Kliknij w linki poniżej, aby przenieść się do odpowiedniej strony.

Spis treści

Spis treści

Co znajdziesz w tym ebooku?

Cześć!

Bardzo się cieszę, że postanowiłeś (albo postanowiłaś) pobrać mojego ebooka!

Napisałam go, aby pomóc Ci przygotować się do rozmowy kwalifikacyjnej z Pythona. Znajdziesz tu 50 pytań, które wybrałam na podstawie swojego doświadczenia z licznych rozmów kwalifikacyjnych, doświadczeń znajomych programistów, a także rozmów z rekruterami, którzy opowiadali mi o swoich zmaganiach o to, by znaleźć idealnego kandydata.

Jeśli przerobisz te pytania, to masz dużą szansę, że na rozmowie kwalifikacyjnej zostaniesz zapytany o któreś z nich. Szczególną popularnością cieszą się cztery zadania algorytmiczne: FizzBuzz, Ciąg Fibonacciego, rekurencyjne przeszukiwanie katalogów i wyszukiwanie binarne.

Poza pytaniami znajdziesz tu również podpowiedzi, jak podejść do ich rozwiązania, informacje, co chcą sprawdzić rekruterzy, zadając konkretne pytanie, oraz krótkie listy tematów do powtórzenia.

A jeśli któreś z pytań sprawiają Ci trudność albo chciałbyś zobaczyć, jak je rozwiązuję, tłumacząc wszystko krok po kroku, to zajrzyj na kolejną stronę.

Gdzie znajdziesz rozwiązania wszystkich zadań?

Wszystkie wymienione w tym ebooku pytania uzupełniłam o podpowiedzi, jak należy podejść do ich rozwiązania.
Nie znajdziesz tu jednak informacji, jak rozwiązać zadania krok po kroku. Ich zamieszczenie w formie tekstowej spowodowałoby, że ebook miałby kilka tysięcy stron.

Wszystkie wymienione tu pytania omówiłam i rozwiązałam krok po kroku w moim kursie na Udemy:

Python – 50 popularnych pytań z rozmów kwalifikacyjnych

W kursie znajdziesz:

- ➔ 5,5 h nagrań.
- ➔ Quizy.
- ➔ Ćwiczenia z kodowania online.
- ➔ Linki do materiałów dodatkowych, poszerzających wiedzę na temat zagadnień poruszonych w każdym z pytań.
- ➔ Linki do mojego githuba, gdzie znajdują się rozwiązania wszystkich zadań z bardzo szczegółowymi komentarzami na temat tego, co dzieje się w każdej linijce kodu.

Kurs cieszy się dużą popularnością i osiągnął na Udemy niezwykle wysoką średnią ocen – **4,9/5.0!**

Kilka opinii na temat kursu:

“Świetny i przystępny. czekam na następne” ~Anna

“Przerobiłem ten kurs. Oceniam bardzo pozytywnie. Duża dawka praktycznej wiedzy.
Widać dużą pracę, jaką włożyłaś w przygotowanie materiału. Nie mylisz się na nagraniach i używasz ładnego języka, co jak dla mnie – osoby, która dużo czyta – jest bardzo ważne. Fajnie się słucha kobiety na kursie. Wszędzie tylko faceci... Czekam na kolejny!” ~Marcin

“Mimo że znałem już podstawy, zdecydowałem się na ten kurs i muszę przyznać, że spełnił moje oczekiwania. Materiał przekazywany w bardzo przystępny sposób. Niektóre trudne wcześniej dla mnie tematy przekazywane są w jasny i zrozumiały sposób, dzięki czemu lepiej je zrozumiałem. Podobały mi się ćwiczenia z kodowania. Na duży plus również kwestia zamieszczenia informacji, gdzie możemy poszerzyć wiedzę z danego zagadnienia. Polecam :)” ~Łukasz

“Świetny kurs dla kogoś, kto chce przygotować się do rozmowy o pracę z Pythona. Pytania omówione bardzo szczegółowo i tak, by każdy mógł zrozumieć. Pokrywają się z tym co mówią Google o najczęściej pojawiających się pytaniach z Pythona. Dobre tempo kursu.” ~Krystyna

Specjalnie dla czytelników tego ebooka przygotowałam zniżkę na kurs. Po kliknięciu w poniższy link uzyskasz promocyjną cenę na dostęp do kursu.

Kurs – zniżka dla czytelników

O autorce

Nazywam się Marta Potocka i od pięciu lat jestem programistką.

W swojej karierze analizowałam logi telekomunikacyjne, pisałam oprogramowanie dla loterii i optymalizowałam procesory sygnałowe. Pracowałam w wielu językach, jednak to Python skradł moje serce.

Poza programowaniem od wielu lat zajmuję się też uczeniem. Prowadziłam szkolenia, bootcampy programistyczne i byłam prelegentką na meetupach. Jakiś czas temu postanowiłam przenieść swoją działalność do sieci, aby jak najwięcej ludzi mogło mieć dostęp do moich szkoleń. Tak powstał mój pierwszy kurs na Udemy.

Od swoich kursantów otrzymuję dziesiątki pytań na temat tego, jak zostać programistą. Zainspirowało mnie to do stworzenia strony jakzostacprogramista.pl, na której zamierzam dzielić się swoją wiedzą i doświadczeniem związanymi z całym procesem zostawania programistą. Aby się to udało, trzeba – wbrew obiegowej opinii – zrobić znacznie więcej, niż tylko nauczyć się języka. Mam nadzieję, że dzięki swojej działalności pomogę ludziom takim jak Ty spełnić marzenia i rozpocząć pracę w tym fascynującym zawodzie.

Jak korzystać z ebooka by przygotować się do rozmowy?

Poniżej znajdziesz 50 pytań, jakie często pojawiają się na rozmowach kwalifikacyjnych z Pythona.

Pytania zostały ułożone w możliwie logicznym porządku – to znaczy, że pewne pojęcia, które pojawiają się na początku (mutowalność, slice'y i inne) – przydają się podczas rozwiązywania zadań z dalszej części listy.

Jeśli chcesz zrobić sobie **solidną powtórkę** wiedzy, polecam przerabiać pytania w takiej kolejności, w jakiej występują w ebooku.

Jeśli jednak czujesz się w Pythonie dość pewnie i chcesz tylko rzucić okiem na kilka tematów, by je sobie odświeżyć, po prostu przejrzyj spis treści i, korzystając z linków, skocz do tych pytań, które są dla ciebie interesujące. **Nic się nie stanie, jeśli nie przeczytasz ebooka od deski do deski!**

Każda pytanie omówione jest na osobnej stronie, na której znajdziesz następujące elementy:

- ➔ Treść pytania, w niektórych przypadkach razem z kodem.
- ➔ Krótką listę tematów do powtórzenia, które związane są z danym pytaniem.
- ➔ Sekcję "**Co sprawdza**", w której spojrzysz na pytanie z punktu widzenia rekrutera i zobaczysz, na co będzie zwracał uwagę, słuchając twojej odpowiedzi.
- ➔ Sekcję "**Jak podejść**", w której zamieściłam porady na temat tego, jak "ugryźć" pytanie, aby zrobić jak najlepsze wrażenie w trakcie rozmowy.

Pytanie 1
Lista niepowtarzalnych elementów

Korzystając z podanej listy A, stwórz listę B, która będzie zawierać tylko unikalne elementy z listy A.

```
A = [1,2,3,3,2,1,2,3]
```

Co sprawdza?

- Czy potrafimy rozwiązać proste zadanie algorytmiczne.
- Czy potrafimy używać pętli for.
- Czy znamy podstawowe metody list.
- Czy potrafimy skorzystać z właściwości set.

Do powtórzenia:

- *pętla for*
- *operacje na listach*
- *właściwości set*

Jak podejść:

- Warto zacząć od rozwiązania wykorzystującego set: B = list(set(A)). Dzięki temu pokażemy, że znamy właściwości set i potrafimy wykorzystać je, by sprytnie i zwięźle rozwiązać zadanie.
- Jeśli rekruter poprosi o alternatywne rozwiązanie, to piszemy pętlę for, w której przeglądamy elementy listy A i jeśli nie znajdują się jeszcze w liście B, to je tam dodajemy. Takie rozwiązanie pozwoli zaprezentować, że potrafimy poradzić sobie z napisaniem prostego algorytmu.

Pytanie 2
Modyfikacja stringa

Co się stanie po wykonaniu poniższego kodu?

```
a = ‘abcdefg’
a[1] = ‘X’
```

Do powtórzenia:

- *mutowalne i niemutowalne typy danych*
- *metoda join()*

Co sprawdza?

- Czy wiemy, że powyższy kod nie zadziała.
- Czy wiemy, które typy danych w Pythonie są niemutowalne i co to oznacza.
- Czy potrafimy "obejść” ograniczenia języka i uzyskać oczekiwany rezultat w inny sposób.

Jak podejść:

- Powiedzieć, dlaczego kod nie zadziała i jaki zwróci błąd (TypeError).
- Zaproponować, jak można uzyskać rezultat, którego prawdopodobnie oczekiwał autor kodu.
- Zmodyfikowanie elementu pod konkretnym indeksem w stringu jest możliwe, jeśli w tym celu przeprowadzimy rzutowanie (zamienimy go) na listę, która jest elementem mutowalnym, a następnie tę listę zamienimy na powrót w stringa, stosując metodę join() i przypiszemy do zmiennej, pod którą znajdował się pierwotny string).

Pytanie 3
Lista vs tupla

Napisz kod, który zaprezentuje najważniejsze różnice między listą a tuplą (krotką).

Co sprawdza?

- Czy znamy różnicę pomiędzy sposobem zapisu listy i tupli.
- Czy wiemy, która z nich jest mutowalna, a która niemutowalna i co to oznacza.

Do powtórzenia:

- *mutowalność*
- *tworzenie i właściwości list*
- *tworzenie i właściwości tupli*

Jak podejść:

- Dla zobrazowania różnic warto stworzyć listę i tuplę o identycznej zawartości i pokazać, że wizualnie odróżnia je sposób zapisu – lista w nawiasach [], a tupla w ().
- Następnie stworzyć kod, który próbuje modyfikować zawartość listy oraz tupli i pokazać, że w drugim przypadku uzyskamy błąd.
- Wyjaśnić, czym jest mutowalność i jakie struktury danych są niemutowalne.

Pytanie 4
Lista wielowymiarowa

Jakiej struktury danych użyłbyś do zamodelowania szafki, która ma 3 szuflady, a w każdej z nich znajdują się 3 przegródki?
Stwórz taki model i umieść stringa "długopis" w środkowej przegródce środkowej szuflady.

Do powtórzenia:

- *zagnieżdżanie list*
- *tworzenie list wielowymiarowych*
- *stosowanie indeksów w zagnieżdżeniach*

Co sprawdza?

- Czy wiemy, jak tworzyć zagnieżdżone listy, aby uzyskać oczekiwaną liczbę wymiarów (2D, 3D...)?
- Czy potrafimy stosować indeksy, aby odnieść się do elementów zagnieżdżonych list.

Jak podejść:

- Tworzymy pustą listę reprezentującą szafkę.
- W niej trzy puste listy reprezentujące szuflady.
- W każdej szufladzie trzy puste listy reprezentujące przegródki.
- Korzystając z odpowiednich indeksów, umieszczamy stringa "długopis" we wskazanym miejscu.
- Aby pokazać, że nasz kod zadziałał poprawnie, warto wydrukować listę reprezentującą szafkę szuflada po szufladzie – pozwoli to łatwo ocenić wizualnie efekt działania kodu.

Pytanie 5
Zagnieżdżone struktury

Z poniższej listy wypisz stringa ‘schowany’:

```
L = [[34, False], [0], [('abc', 123), \
{'a': 1, 'x': (True, 'schowany', 5)}]]
```

Co sprawdza?

- Czy potrafimy pozornie skomplikowane zadanie rozwiązać krok po kroku.
- Czy potrafimy stosować indeksy oraz klucze słownika, by operować na zagnieżdżonych strukturach danych.

Do powtórzenia:

- *stosowanie indeksów w zagnieżdżeniach*
- *odczyt wartości słownika przy użyciu klucza*

Jak podejść:

- Zadania tego typu najlepiej rozwiązywać iteracyjnie – krok po kroku.
- Należy zacząć od zidentyfikowania "najpłycej położonej" struktury danych, w której znajduje się poszukiwany string, i wypisać jej zawartość (tu lista pod indeksem 2 w liście L).
- Następnie zidentyfikować kolejną wewnętrzną dla niej strukturę danych, w której znajduje się string (tu słownik w dużej liście).
- Kontynuujemy "wyłuskiwanie" tak długo, aż uda nam się wypisać żądany element.

Pytanie 6
Klucze i wartości

Jakiej struktury danych użyłbyś do zapisania numerów telefonów wszystkich klientów firmy
– i odpowiadających im nazwisk? Wybierz strukturę tak, aby sprawdzenie właściciela numeru telefonu nie zajmowało wiele czasu.
Następnie stwórz przykładową strukturę przechowującą poniższe informacje:

```
123456789 - Jan Kot
999888777 - Anna Lis
111222333 - Jan Kot
```

Odczytaj nazwisko właściciela numeru 123456789.

Do powtórzenia:

- *tworzenie słowników*
- *złożoność obliczeniowa (bigO) odczytu informacji ze słownika*

Co sprawdza?

- Czy znamy właściwości słowników i wiemy, do przechowywania jakich danych się je stosuje.
- Czy znamy złożoność obliczeniową odczytu danych ze słownika.

Jak podejść:

- W tym zadaniu należy zastosować słownik, ponieważ:
  - Pozwala przechowywać pary odpowiadających sobie wartości, z których jedna jest unikalna (nr telefonu).
  - Pozwala na bardzo szybki O(1) odczyt wartości przypisanej do podanego klucza.

Pytanie 7
Co może być kluczem

Który z poniższych słowników został utworzony niepoprawnie?

```
A = {1: 1, 2: 4, 3: 9}
B = {'imie': 'Anna', 'nazwisko': 'Kowalska'}
C = {[4, 5]: [16, 25]}
D = {(4, 5): [16, 25]}
E = {{1:2}: 'jeden_dwa'}
```

Co sprawdza:

- Czy wiemy, jakie obiekty mogą, a jakie nie mogą być kluczami słownika.
- Czy potrafimy wyjaśnić, dlaczego tak się dzieje, odwołując się do implementacji słownika jako tablicy hashującej.

Do powtórzenia:

- *zasady tworzenia kluczy słownika*
- *implementacja słownika jako tablicy hashującej (mieszającej)*

Jak podejść:

- W pierwszym kroku udzielamy odpowiedzi, że niepoprawnie zostały utworzone słowniki C i E, ponieważ ich kluczami są mutowalne typy danych.
- Następnie warto w kilku słowach wyjaśnić, dlaczego w słownikach wymagane są niemutowalne klucze (spowodowane jest to implementacją słownika jako tablicy hashującej) – każdy klucz musi być unikalny.

Pytanie 8
Kolejność elementów słownika

Co zostanie wypisane w wyniku wykonania poniższego kodu?

```
D = {1: 'Ala', 2: 'ma', 3: 'kota'}
for key in D:
    print(D[key])
```

Do powtórzenia:

- *implementacja słownika a kolejność jego elementów*
- *różnice tej kwestii między wersjami Pythona*

Co sprawdza?

- Czy wiemy, że "typowe" słowniki nie zapamiętują kolejności elementów.
- Czy wiemy, dlaczego tak się dzieje.
- Czy potrafimy wskazać, od której wersji Pythona kolejność elementów jest zachowana.

Jak podejść:

- Pytanie jest podchwytliwe. Na pierwszy rzut oka spodziewamy się wypisania elementów słownika po kolei. Jednak implementacja słownika jako tablicy hashującej powoduje, że standardowy słownik nie zapamiętuje kolejności elementów. Jednak w wersjach Pythona od 3.6 w górę kolejność jest zachowana.
- Zatem odpowiedź na pytanie brzmi: zależy od wersji Pythona.

Pytanie 9
Czy klucz jest w słowniku?

Dla danego stringa x stwórz słownik przechowujący informację, ile razy dana litera wystąpiła w stringu.

```
x = "myszydokazująggdykotanieczują"
```

Co sprawdza?

- Czy potrafimy sprawdzić, czy dany element jest kluczem słownika?
- Czy potrafimy zaktualizować wartość przechowywaną pod danym kluczem?
- Czy potrafimy dodać nowy klucz do słownika.

Do powtórzenia:

- *sprawdzanie, czy klucz jest w słowniku*
- *dodawanie klucza do słownika*
- *zwiększanie wartości pod kluczem*

Jak podejść:

- Zaczynamy od stworzenia pustego słownika.
- Następnie sprawdzamy każdą literę w stringu x i odnotowujemy, czy występuje już w słowniku jako klucz.
  - Jeśli tak – zwiększamy przypisaną do niego wartość.
  - Jeśli nie – dodajemy daną literę jako klucz, przypisując mu wartość równą 1.

Pytanie 10
Wartości słownika

Korzystając ze słownika stworzonego w poprzednim zadaniu, sprawdź, czy któraś z liter wystąpiła w stringu dokładnie 4 razy. Jeśli tak – wypisz True, jeśli nie – False.

```
litery = {'m': 1, 'y': 3, 's': 1, 'z': 3, 'd': 2,\
'o': 2, 'k': 2, 'a': 2, 'u': 2, 'j': 2, 'ą': 2,\
'g': 1, 't': 1, 'n': 1, 'i': 1, 'e': 1, 'c': 1}
```

Do powtórzenia:

- *sprawdzanie wartości słownika*
- *metoda values()*

Co sprawdza?

- Czy potrafimy zastosować metodę values(), aby sprawdzić wartości przechowywane w słowniku.
- Czy potrafimy sprawdzić, czy dany element znajduje się wśród wartości słownika.

Jak podejść:

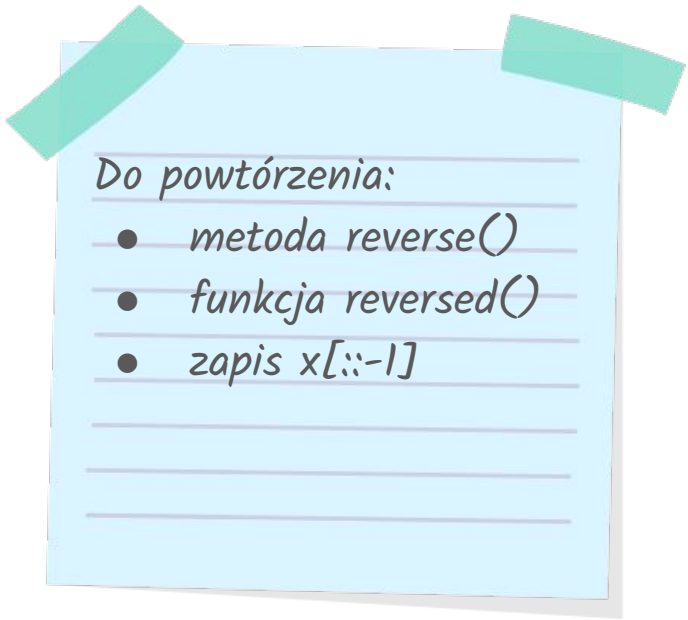
- Należy odczytać wartości przechowywane jako *wartości* (values) słownika.
- Następnie sprawdzić, czy znajduje się wśród nich cyfra 4.
- Napisać blok logiczny, który w zależności od rezultatu zwróci True lub False.

Pytanie 11
Odwracanie listy

Na podstawie listy jezyki stwórz listę jezyki_odwrocone zawierającą elementy listy jezyki w odwróconej kolejności.
jezyki = [‘Python’, ‘Java’, ‘C#’, ‘Ruby’]

Co sprawdza?

- Znajomość dostępnych w pythonie funkcji i metod odwracających listy (reverse() i reversed()).
- Znajomość zapisu wykorzystującego slice’y, który pozwala odwrócić obiekt iterowalny.
- Czy potrafimy rozwiązać zadanie algorytmicznie, przy użyciu pętli.



Jak podejść:

- Zacząć od najbardziej pythonowego sposobu odwracania obiektów iterowalnych, czyli zapisu wykorzystującego slice’y: x[::-1] – zwraca elementy x w odwróconej kolejności.
- Omówić metodę reverse() odwracającą obiekt w miejscu i funkcję reversed() zwracającą iterator.
- Zadanie można też rozwiązać algorytmicznie, stosując indeksy lub metodę insert().

Pytanie 12
Palindrom

Napisz funkcję sprawdzającą, czy podane słowo jest palindromem.
Uruchom funkcję, aby sprawdzić, czy palindromami są słowa "kajak" i "anakonda".

Palindrom – słowo, które czytane od początku i od końca brzmi dokładnie tak samo.

Do powtórzenia:

- *czym jest palindrom*
- *odwracanie obiektu przy użyciu slice'ów*
- *pętla while*

Co sprawdza:

- Czy wiemy, czym jest palindrom (ale jeśli nie, to rekruter nam podpowie).
- Czy potrafimy sprawdzić, czy dane słowo jest palindromem, pisząc odpowiedni algorytm.
- Czy potrafimy uzyskać to samo przy użyciu pythonowego "hacka" wykorzystującego slice'y.

Jak podejść:

- Warto zacząć od rozwiązania sprytnego, czyli przy użyciu zapisu x[::-1] stworzyć odwróconą wersję zadanego stringa i porównać je ze sobą. Da się to uzyskać w jednej linijce.
- Możemy być też poproszeni o napisanie algorytmu rozwiązującego to zadanie, wtedy należy sukcesywnie porównywać pierwszą i ostatnią literę, drugą i przedostatnią itd. i sprawdzać, czy są identyczne. W tym rozwiązaniu stosujemy pętlę while.

Pytanie 13
Range

Stwórz dwie listy:
A – zawierającą liczby od 1 do 10.
B – zawierającą co trzecią liczbę z zakresu od 100 do 1.
W obu przypadkach możesz napisać tylko jedną linijkę kodu.

Co sprawdza:

- Czy potrafimy napisać prostego range'a, aby uzyskać sekwencję danych.
- Czy potrafimy napisać bardziej zaawansowanego range'a z malejącą sekwencją danych i zdefiniowanym krokiem.

Do powtórzenia:

- *działanie range()*
- *parametry start, stop, krok w range()*

Jak podejść?

- Stworzenie listy A wymaga napisania prostego range'a z uwzględnieniem, że argument stop musi być o 1 większy niż liczba, na której chcemy się zatrzymać.
- Stworzenie listy B wymaga napisania range, gdzie argument startu jest większy niż argument stopu, a odstępy między kolejnymi elementami (krok) określamy liczbą ujemną.

Pytanie 14
Slice'y

Wypisz pierwsze 5 elementów listy L.
Wypisz co drugą literę stringa s, zaczynając od ostatniej i cofając się do początku.

```
L = [11, 22, 33, 44, 55, 66, 77, 88, 99, 1010]
s = 'a nMozh^tKysPW 9ęxi b$uML'
```

Do powtórzenia:

- *działanie slice'ów w Pythonie*
- *parametry startu, stopu i kroku w slice'ach*

Co sprawdza:

- Czy potrafimy napisać prostego slice'a, aby odczytać fragment sekwencji danych.
- Czy potrafimy napisać bardziej zaawansowanego slice'a, odczytującego dane od końca do początku i z określonym krokiem.

Jak podejść?

- Odczytanie fragmentu listy L wymaga napisania prostego slice'a (pamiętaj o tym, że argument stop musi być o 1 większy niż liczba, na której chcemy się zatrzymać).
- Odczytanie fragmentu stringa s wymaga napisania slice'a, w którym argument startu i kroku są liczbami ujemnymi (indeks ujemny w wypadku startu), a argumentu stopu nie podajemy.

Pytanie 15
Slice'y praktycznie

Napisz funkcję, która sprawdzi, czy podany string zaczyna się słowem "python" i kończy rozszerzeniem ".py".
Przetestuj nią stringi:

```
a = "python_moj_kod.py"
b = "python_notatki.txt"
```

Co sprawdza:

- Czy potrafimy użyć slice'ów do odczytania określonej liczby znaków z początku i końca stringa.
- Czy potrafimy porównać odczytane wartości z podanymi stringami, stosując operator ==.
- Czy potrafimy napisać zdanie logiczne, w którym dwa elementy muszą być prawdą, aby zdanie było prawdziwe (operator and).

Do powtórzenia:

- *stosowanie slice'ów*
- *operator == porównujący wartości dwóch elementów*
- *operator logiczny and*

Jak podejść:

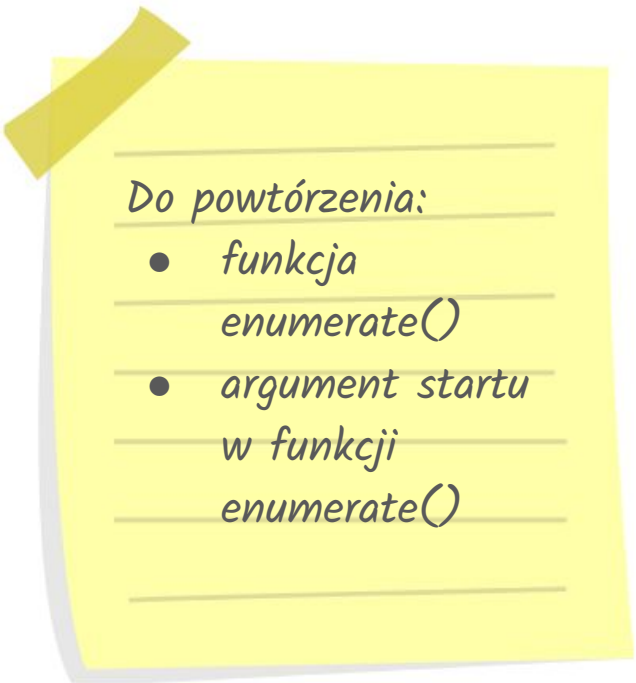
- Rozwiązanie zadania wymaga wykorzystania wiedzy z poprzedniego zadania, czyli umiejętnego zastosowania slice'ów.
- Należy napisać dwa slice'y:
  - Odczytujący pierwsze 5 znaków stringa.
  - Odczytujący ostatnie 4 znaki stringa.

  Następnie trzeba je porównać ze stringami "python" i ".txt".

Pytanie 16
Enumerate

Wypisz podaną listę imion, przed każdym dodając kolejny numer. Zacznij numerowanie od 1.

```
imiona = ['Adam', 'Stanisław', 'Maria', 'Zofia', 'Mikołaj']
```



Co sprawdza:

- Czy potrafimy zastosować funkcję enumerate(), aby "automatycznie" numerować kolejne elementy odczytywane przy użyciu pętli.
- Czy potrafimy ustawić argument startu tak, aby enumerate() zaczynała odliczanie od innej liczby niż 0.

Jak podejść:

- Najlepszym rozwiązaniem jest zastosowanie funkcji enumerate() z argumentem startu ustawionym na 1. Pozwoli nam to zgrabnie i szybko dodać numery do kolejnych elementów listy.
- Alternatywą (jeśli nie zna się enumerate) jest napisanie pętli, która w każdym obiegu będzie zwiększała wartość zewnętrznej zmiennej przechowującej licznik i używała tej zmiennej do numerowania kolejnych elementów listy.

Pytanie 17
min i max

Znajdź różnicę między największą a najmniejszą wartością na poniższej liście.
Zadbaj o to, aby rozwiązanie było efektywne.

```
A = [4, 5, 7, -3, 2, 8, -10, 15]
```

Co sprawdza:

- Czy wiemy, jaka jest złożoność obliczeniowa sortowania listy
  (i dlaczego nie opłaca się jej sortować w wypadku tego zadania).
- Jak działają funkcje min() i max() i jaka jest złożoność obliczeniowa wyszukania najmniejszego
  i największego elementu na liście.

Do powtórzenia:

- *funkcja min()*
- *funkcja max()*
- *złożoność obliczeniowa sortowania*

Jak podejść?

- Zadanie można rozwiązać, sortując listę i porównując ze sobą pierwszy i ostatni element, jednak jest to podejście nieefektywne, bo jego złożoność obliczeniowa będzie równa złożoności sortowania – O(n*logn).
- Znacznie lepszym podejściem jest wyszukanie najmniejszego i największego elementu przy użyciu funkcji min() i max(), które mają złożoność O(n).

Pytanie 18
Podzielność

Napisz funkcję, która będzie pobierać dwie liczby i sprawdzać, czy pierwsza z nich jest podzielna przez drugą.

Do powtórzenia:

- *dzielenie modulo – zapisywane przy użyciu operatora %*

Co sprawdza:

- Czy znamy operator dzielenia modulo % i czy wiemy, jaką wartość zwraca.
- Czy potrafimy zastosować operator dzielenia modulo, by sprawdzić podzielność liczb.

Jak podejść?

- Aby sprawdzić, czy a jest podzielne przez b, należy podzielić a przez b przy użyciu operatora modulo – %, który zwraca resztę z dzielenia.
- Jeśli otrzymany wynik będzie wynosił 0, to oznacza, że a jest podzielne przez b (nie ma reszty z dzielenia).
- Jeśli otrzymany wynik będzie różny od 0 – to oznacza, że a nie jest podzielne przez b (bo pojawiła się reszta z dzielenia).

Pytanie 19
Argumenty domyślne

Wyjaśnij, jak działa poniższa funkcja. Wyjaśnij, skąd wzięły się wyniki zwrócone przez poszczególne wywołania funkcji.

```
def dodaj_do_listy(n, lista=[]):
 lista.append(n)
 print(lista)

dodaj_do_listy(1)
dodaj_do_listy(2,[4,5])
dodaj_do_listy(3)
```

Co sprawdza:

- Czy rozumiemy działanie argumentów domyślnych.
- Czy wiemy, kiedy powstaje argument domyślny i jaka pułapka się w tym kryje.

Do powtórzenia:

- *czym są argumenty domyślne i jak się je stosuje*
- *kiedy powstaje argument domyślny*

Jak podejść?

- Jest to pytanie pułapka, które sprawdza, czy jesteśmy świadomi podstępnego mechanizmu działania argumentów domyślnych.
- Argument domyślny tworzony jest tylko **raz** podczas **definiowania** funkcji, a podczas kolejnych wywołań nie ulega zmianie i nie jest tworzony na nowo. W rezultacie trzecie wywołanie funkcji (drugie, w którym użyty został argument domyślny) powoduje wypisanie listy zawierającej elementy dodane zarówno podczas pierwszego, jak i trzeciego wywołania funkcji.

Pytanie 20
List comprehension

Co otrzymamy po wydrukowaniu poniższych zmiennych?

```
L = [1,2,3,4,5,6]
L1 = [x for x in range(5)]
L2 = [x**2 for x in L]
L3 = [x for x in L if x % 2 == 0]
L4 = ['Parzysta' if x%2 == 0 else 'Nieparzysta' for x in range(5)]
L5 = [(x, x+10) for x in L]
D1 = {x:x % 2 == 0 for x in L}
```

Do powtórzenia:

- *list comprehension*
- *dict comprehension*
- *stosowanie w nich wyrażeń logicznych*

Co sprawdza:

- Czy znamy zapis list comprehension i dict comprehension.
- Czy poradzimy sobie z zaawansowanymi zapisami wykorzystującymi warunki logiczne.

Jak podejść?

- Rozwiązanie zadania wymaga znajomości wyrażeń list comprehension i dict comprehension.
- Wyrażenia zawierające warunki logiczne (tworzące zmienne L3 i L4) warto przeanalizować dokładnie, ponieważ są dość zaawansowane.

Pytanie 21
‘is’ vs ==

Co wydrukuje się w wyniku wykonania poniższego kodu?

```
print(1 == True)
print(1 is True)
```

Do powtórzenia:

- *operator porównania wartości ==*
- *operator porównania identyczności ‘is’*
- *funkcja id()*

Co sprawdza:

- Czy wiemy, co sprawdza operator porównania wartości ==.
- Czy wiemy, co sprawdza operator porównania identyczności ‘is’.
- Czy wiemy, w jakich sytuacjach możemy oczekiwać zgodnych, a w jakich różnych wyników działania tych operatorów.

Jak podejść:

- Należy wyjaśnić, jaka jest różnica pomiędzy porównywaniem wartości danych elementów (i dlaczego 1 co do wartości równe jest True), a ich identyczności.
- Warto posłużyć się funkcją id() do zobrazowania różnicy w identyfikatorach obiektów pozornie identycznych, na przykład dwóch list mających identyczną zawartość.

Pytanie 22
False is False is False

Co zostanie wydrukowane w wyniku wykonania poniższego kodu?

```
print(False is False is False)
```

Co sprawdza:

- Czy wiemy, jak działają w Pythonie porównania łańcuchowe (dłuższe niż dwa elementy i jeden operator porównania).
- Czy potrafimy rozbić zdanie z operatorami łańcuchowymi na czynniki pierwsze i przeanalizować je tak, jak robi to interpreter Pythona.
- Czy znamy operator 'is'.

Do powtórzenia:

- *operator sprawdzania identyczności 'is'*
- *porównania łańcuchowe*

Jak podejść?

- Powyższe zdanie zawiera operator łańcuchowy, czyli trzy elementy połączone dwoma operatorami porównania.
- Aby przeanalizować je poprawnie, należy rozbić zdanie na dwuelementowe zdania połączone operatorem sumy logicznej **and**.
- Zatem zdanie rozpisujemy wg wzoru:
  A is B is C –> (A is B) and (B is C).

Pytanie 23
Lambda

Czym jest lambda?
Napisz przykładowy kod wykorzystujący lambdę.

Co sprawdza:

- Czy wiemy, czym jest lambda (inaczej funkcja anonimowa).
- Czy potrafimy utworzyć lambdę.
- Czy wiemy, w jakich sytuacjach należy lambdy stosować
  (a w jakich lepiej użyć klasycznych funkcji mających nazwy).

Do powtórzenia:

- *czym jest lambda*
- *jak stworzyć lambdę*
- *gdzie stosuje się lambdy*

Jak podejść?

- Aby zaprezentować zastosowanie lambdy, należy stworzyć możliwie krótki fragment kodu, w którym lambda będzie wykorzystana.
- Najlepiej użyć jej w funkcjach filter(), map() lub sorted()
  – na przykład w wypadku tej ostatniej możemy napisać lambdę, która pozwoli posortować listę zawierającą dwuelementowe tuple w taki sposób, aby kryterium sortowania był drugi element każdej tupli (lambda x:x[1]).

Pytanie 24
Kopiowanie listy

Co zostanie wypisane w wyniku wykonania poniższego kodu?

```
A = [1,2,3,4,5]
B = A
C = A[:] # C = list(A)
B[0] = 111
print(A, B, C)
```

Do powtórzenia:

- *czym jest referencja do danych w pamięci*
- *jak uzyskać kopię listy*

Co sprawdza:

- Czy wiemy, że zmienne przechowują referencje do danych, a nie same dane.
- Co się stanie, jeśli dwie zmienne będą wskazywać na te same dane w pamięci.
- Jak skopiować listę, aby utworzyć w pamięci jej wersję niezależną od listy, z której kopiujemy.

Jak podejść:

- Odpowiadając na to pytanie, należy rozpocząć od tego, że w Pythonie zmienne przechowują referencje do danych, a nie same dane i w związku z tym przypisanie do siebie dwóch zmiennych nie oznacza utworzenia kopii danych. Co więcej, modyfikacja danych przy użyciu jednej zmiennej wpłynie na wartości wyświetlane przez drugą zmienną (która wskazuje na te same dane) (przykład list A i B).
- Warto też wspomnieć, jak utworzyć niezależną kopię danych (tu lista C utworzona przy użyciu slice'a).

Pytanie 25
Zmienne lokalne i globalne

Co stanie się w wyniku wykonania poniższego kodu?

```
x = 10
def f():
 global x
 x = 111
 y = 12
 print(x, y)
f()
print(x)
```

Co sprawdza:

- Czy wiemy, czym są, gdzie występują i jaki mają zakres "obowiązywania" zmienne lokalne.
- Czy wiemy, czym są i w jakim zakresie widoczne są zmienne globalne.
- Do czego służy słowo kluczowe 'global'.

Do powtórzenia:

- *zmienne lokalne*
- *zmienne globalne*
- *słowo kluczowe global*

Jak podejść?

- Należy uważnie przeanalizować, co dzieje się ze zmienną x podczas działania kodu.
- Uruchomienie funkcji f() powoduje zmianę zmiennej globalnej x (dzięki słowu kluczowemu global). Następnie funkcja f() drukuje ustawione wartości x i y.
- Po wykonaniu funkcji kolejna linijka ponownie drukuje zmienną x – która jednak została zmieniona **globalnie**, a zatem jej wartość została trwale nadpisana.

Pytanie 26
Operacje na plikach

Stwórz plik o nazwie "moj_plik.txt".
Wpisz do niego liczby od 1 do 100, każdą w nowej linijce.
Otwórz plik i zapisz jego zawartość do listy `z_pliku`.

Do powtórzenia:

- *blok with()*
- *funkcja open() i jej parametry*
- *funkcje write()*
- *funkcja readlines()*

Co sprawdza:

- Czy potrafimy stosować blok with służący do operacji na plikach.
- Czy potrafimy wpisywać dane do pliku.
- Czy potrafimy odczytać dane z pliku.
- Czy znamy metody open(), write() i readlines().

Jak podejść?

- Należy zacząć od utworzenia bloku with, w którym przy użyciu funkcji open() tworzymy plik z prawami do zapisu.
- Następnie przy użyciu pętli for i funkcji with wpisujemy do niego kolejne liczby z zakresu 1–100.
- Na końcu otwieramy kolejny blok with, tym razem z prawami do odczytu i przy użyciu funkcji readlines() odczytujemy zawartość pliku do listy.

Pytanie 27
Zagnieżdżone pętle

Objętość graniastosłupa oblicza się na podstawie wzoru: V = a * b * h. Długości boków podstawy to a i b, zaś h to wysokość. Poniższy kod znajduje największy graniastosłup jaki możemy utworzyć z elementów list A, B i H. Ile operacji zostanie wykonane w wyniku uruchomienia tego kodu? W jaki sposób można by to zadanie rozwiązać bardziej efektywnie?

```
import random
A = [random.randint(0,100) for i in range(5)]
B = [random.randint(0,100) for i in range(5)]
H = [random.randint(0,100) for i in range(5)]

max_v = 0
for a in A:
 for b in B:
 for h in H:
 if a * b * h > max_v:
 max_v = a * b * h
print(max_v)
```

Do powtórzenia:

- *funkcja randint z biblioteki random*
- *działanie pętli for*

Jak podejść:

- Aby policzyć liczbę operacji wykonanych w pętlach, należy pomnożyć przez siebie liczbę operacji wykonanych w każdej pętli.
- Prostsze rozwiązanie polega na wykorzystaniu funkcji max(), by znaleźć największy element każdej listy.

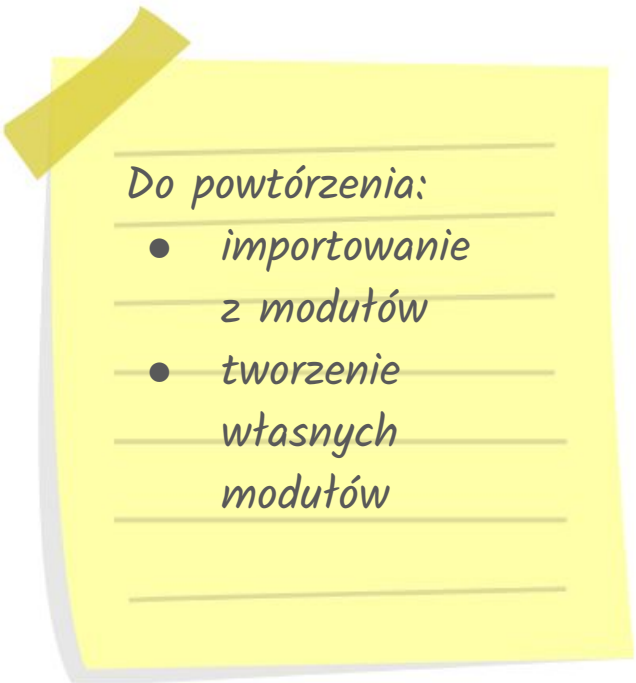
Co sprawdza:

- Czy znamy funkcję randint z biblioteki random.
- Czy wiemy, jak wyznaczyć złożoność obliczeniową kodu zawierającego zagnieżdżone pętle..

Pytanie 28
Moduły

Jaki błąd popełniono w poniższym kodzie? Co zrobić, aby go uniknąć? Stwórz moduł `kserokopiarka` zawierający funkcję, która podany string wydrukuje dwa razy.
Użyj tej funkcji w kodzie poniżej.

```
from drukarka import wydrukuj_imie
def wydrukuj_imie(imie):
 print(imie)
```



Co sprawdza:

- Czy zdajemy sobie sprawę z mechanizmu przesłaniania nazw funkcji i zmiennych.
- Czy wiemy, jak działa importowanie z modułów.
- Czy potrafimy stworzyć własny moduł i zaimportować z niego funkcję.

Jak podejść?

- Należy zacząć od wskazania błędu w kodzie powyżej – zaimportowana funkcja `wydrukuj_imie` zostaje przesłonięta przez lokalną funkcję `wydrukuj_imie`.
- Następnie tworzymy własny moduł (a więc osobny plik z rozszerzeniem .py), w nim funkcję kserokopiarka i importujemy ją do kodu w analogiczny sposób do tego, w jaki została zaimportowana funkcja `wydrukuj_imie`.

Pytanie 29
Pakiety

Do czego w Pythonie służy plik: __init__.py ?
Stwórz krótki kod prezentujący jego zastosowanie.

Co sprawdza:

- Czy wiemy, że pliki __init__.py służą do przekształcenia folderu z plikami w pakiet pythona.
- Czy potrafimy samodzielnie stworzyć pakiet.
- Czy potrafimy importować z pakietów.

Do powtórzenia:

- *czym są pakiety w pythonie*
- *jak stworzyć pakiet*
- *jak importować z pakietu*

Jak podejść?

- Aby zaprezentować, do czego służy plik __init__.py, należy stworzyć własny pakiet.
- W tym celu tworzymy pusty folder, a w nim umieszczamy pusty plik __init__.py oraz dowolny inny plik Pythona (z rozszerzeniem.py) – który stanie się jednym z modułów pakietu.
- Następnie w nowym pliku importujemy ten moduł, odwołując się do nazwy pakietu (np. from <nazwa pakietu> import <nazwa modułu>).

Pytanie 30
Operatory ‘is’, ‘not’, ‘in’

Jaką wartość przyjmie poniższe zdanie logiczne? Wyjaśnij proces jego ewaluacji i znaczenie poszczególnych słów: is, not, in.

```
print(1 is not True in [1,2,3])
```

Do powtórzenia:

- *operator ‘is’*
- *operator ‘not’*
- *operator ‘in’*
- *porównania łańcuchowe w Pythonie*

Co sprawdza:

- Czy wiemy, jak działa operator sprawdzania identyczności ‘is’.
- Czy wiemy, jak działa operator negacji logicznej ‘not’.
- Czy wiemy, jak działa operator sprawdzania przynależności do sekwencji ‘in’.
- Czy wiemy, jak Python traktuje porównania łańcuchowe.

Jak podejść?

- Kod, który mamy przeanalizować, zawiera porównanie łańcuchowe, zatem w pierwszym kroku należy rozpisać go na dwa krótsze zdania logiczne połączone operatorem and (tak jak zrobi to Python).
- Rozpisujemy wg schematu: A is B in C –> (A is B) and (B in C).
- Następnie określamy wartość logiczną krótszych zdań, a potem wartość całości, pamiętając, że operator and wymaga, aby zdania logiczne po jego lewej i prawej stronie były prawdą.

Pytanie 31
map i filter

Otrzymujesz listę nazwisk, jakie klienci wprowadzili w formularz na stronie internetowej.
Użyj funkcji filter(), aby usunąć z niego wszystkie wpisy, które nie są stringami.
Użyj funkcji map(), aby przerobić nazwiska tak, żeby wszystkie były zapisane poprawnie z wielkimi literami tylko na początku imienia i nazwiska.

```
nazwiska = ['jan kot', 18, 'ANNA KRÓL', 'jÓzef BYK', ['nie', 'wasza','sprawa'], 'ROBERT wąŻ']
```

Co sprawdza:

- Czy potrafimy stosować funkcję filtrującą listę – filter().
- Czy potrafimy stosować funkcję mapującą listę – map().
- Czy potrafimy użyć lambdy w obu tych funkcjach, aby uzyskać oczekiwane rezultaty.

Do powtórzenia:

- *funkcja map()*
- *funkcja filter()*
- *lambda*

Jak podejść?

- W pierwszym kroku filtrujemy listę `nazwiska` przy użyciu funkcji filter() i lambdy, która sprawdza, czy dany element jest stringiem (i jeśli nie, to zostanie on wyfiltrowany, czyli usunięty z listy).
- W drugim kroku, przy użyciu funkcji map(), na wszystkich elementach listy stosujemy metody lower() i title(), aby uzyskać stringi zapisane w zadany sposób.

Pytanie 32
Dekoratory

Do czego w Pythonie służą dekoratory? Napisz dekorator, który będzie dodawał trzy gwiazdki przed i po efekcie działania udekorowanej funkcji.

Do powtórzenia:

- *do czego służą dekoratory*
- *zastosowanie symbolu @ w dekoratorach*
- *tworzenie dekoratora*

Co sprawdza:

- Czy wiemy, czym są i do czego służą dekoratory.
- Czy potrafimy napisać prosty dekorator.
- Czy potrafimy zastosować dekorator, aby udekorować nim inną funkcję (a więc czy znamy zapis wykorzystujący @).

Jak podejść?

- W pierwszym kroku tworzymy dekorator – a więc funkcję, która pobiera inną funkcję. W jej wnętrzu tworzymy kolejną funkcję, która dekoruje, a następnie zwraca funkcję przekazaną jako argument dekoratora.
- W drugim kroku pokazujemy działanie naszego kodu poprzez udekorowanie dowolnej innej funkcji naszym dekoratorem (stosując zapis z @).

Pytanie 33
Generatory

Do czego w Pythonie służy słowo kluczowe yield ?
Napisz przykładowy kod wykorzystujący yield.

Co sprawdza:

- Czy wiemy, że słowo kluczowe yield zarezerwowane jest dla generatorów i jego zastosowanie powoduje przekształcenie funkcji w generator.
- Czy potrafimy napisać prosty generator.

Do powtórzenia:

- *słowo kluczowe yield*
- *działanie generatorów*
- *tworzenie generatorów*

Jak podejść?

- Aby zobrazować, do czego służy yield, musimy napisać prosty generator – najlepiej aby zajmował się on zwracaniem elementów jakiejś sekwencji – np. liczb lub kolejnych liter stringa.
- Nasz generator od zwykłej funkcji będzie różnił się tym, że zamiast słowa return zastosujemy yield.
- Następnie tworzymy zmienną, do której przypisujemy obiekt generatora.
- Aby odczytać kolejne elementy zwracane przez generator, używamy funkcji next(), której jako parametr przekazujemy obiekt generatora.

Pytanie 34
assert

Napisz fragment kodu, w którym zobrazujesz użycie słowa kluczowego assert.
Wyjaśnij, jaka jest rola testów jednostkowych i czym charakteryzuje się dobry test jednostkowy.

Do powtórzenia:

- *do czego służy słowo kluczowe assert*
- *jak napisać test jednostkowy*
- *jak uruchamiać testy (Pytest lub Unittest)*

Co sprawdza:

- Czy wiemy, jak działa słowo kluczowe assert i jak użyć go do napisania testu jednostkowego.
- Czy wiemy, jaka jest konwencja tworzenia nazw testów.
- Czy potrafimy uruchomić i odczytać wyniki testów przy użyciu Pytest lub Unittest.
- Czy wiemy, do czego służą testy jednostkowe i jak napisać dobry test.

Jak podejść?

- Na początku tworzymy prostą funkcję, którą będziemy mogli przetestować – np. funkcję dodająca dwie liczby.
- Następnie tworzymy kilka testów, pamiętając o odpowiednim nazewnictwie (dodaniu ‘test_’ na początku nazwy testu) i zasadzie testowania jednej rzeczy na raz.
- Następnie uruchamiamy testy.
- Na końcu omawiamy, dlaczego napisaliśmy takie a nie inne testy i czym charakteryzuje się dobry test jednostkowy.

Pytanie 35
__init__

Do czego w Pythonie służy __init__ ? Czym różni się od __init__.py ?
Napisz fragment kodu wykorzystujący __init__.

Co sprawdza:

- Czy wiemy, czym jest i jak tworzy się konstruktor klasy.
- Czy potrafimy napisać klasę z prostym konstruktorem.
- Czy potrafimy – przy użyciu tej klasy i konstruktora – stworzyć obiekt klasy.

Do powtórzenia:

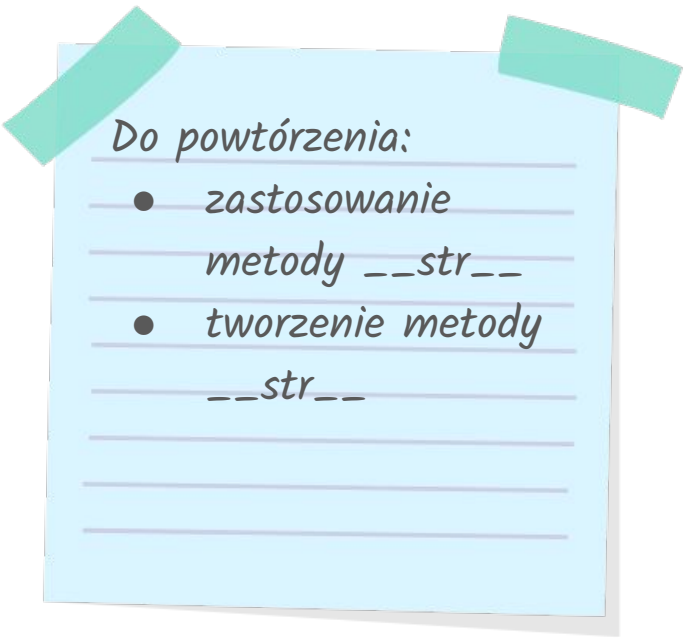
- *tworzenie konstruktora klasy*
- *tworzenie obiektu klasy przy użyciu konstruktora*

Jak podejść?

- Na początku warto zaznaczyć, że __init__(), czyli konstruktor klasy, nie ma nic wspólnego (poza podobieństwem nazwy) z __init__.py, czyli plikiem służącym do tworzenia pakietów w Pythonie.
- Następnie tworzymy prosta klasę (np. Pies) i w niej konstruktor pobierający podstawowe dane (np. imię i rasę psa).
- Następnie tworzymy minimum dwa różne obiekty klasy Pies (np. maly_pies i duzy_pies), aby zobrazować zastosowanie konstruktora do tworzenia różnych obiektów tej samej klasy.

Pytanie 36
__str__

Wykorzystajmy klasę Pies i obiekt maly_pies z poprzedniego pytania.
Co się stanie, gdy wypiszemy print(maly_pies)?
Co zrobiłbyś, aby wydrukowana w ten sposób informacja zawierała imię i rasę psa?



Co sprawdza:

- Czy wiemy, do czego służy metoda __str__.
- Czy wiemy, jak napisać własną metodę __str__ w tworzonej przez nas klasie.
- Czy wiemy, jaka metoda wywoływana jest, gdy używamy funkcji print() do wypisania informacji o obiekcie i co zostanie wypisane, jeśli nasza klasa nie będzie zawierała metody __str__.

Jak podejść?

- Należy zacząć od wyjaśnienia, do czego służy metoda __str__ i jak wygląda efekt zastosowania funkcji print() na obiekcie klasy, jeśli ta klasa nie posiada metody __str__.
- Następnie wewnątrz naszej klasy tworzymy prostą metodę __str__, która zwraca stringa zawierającego informacje o obiekcie klasy.

Pytanie 37
Dziedziczenie – pytanie

Co zostanie wypisane w wyniku uruchomienia poniższego kodu?

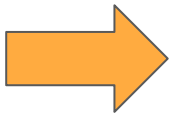
```
class ZwierzeLadowe:
 def przedstaw_sie(self):
 print("Jestem zwierzęciem lądowym.")
 def biegaj(self):
 print("Biegam tu i tam.")

class ZwierzeMorskie:
 def przedstaw_sie(self):
 print("Jestem zwierzęciem morskim.")
 def plywaj(self):
 print("Pływam tu i tam.")

class SwinkaMorska(ZwierzeLadowe, ZwierzeMorskie):
 pass

swinka = SwinkaMorska()
swinka.przedstaw_sie()
swinka.biegaj()
swinka.plywaj()
```

Omówienie pytania na kolejnej stronie



Pytanie 37
Dziedziczenie – omówienie pytania

Co sprawdza:

- Czy wiemy, czym jest dziedziczenie.
- Czy wiemy, w jaki sposób klasa potomna dziedziczy metody z klasy nadrzędnej.
- Czy wiemy, w jakiej kolejności następuje dziedziczenie w wypadku dziedziczenia z więcej niż jednej klasy.

Do powtórzenia:

- *czym jest dziedziczenie*
- *kolejność dziedziczenia w wypadku dziedziczenia z więcej niż jednej klasy*

Jak podejść?

- Omawiając kod, należy zwrócić uwagę na to, że klasa SwinkaMorska dziedziczy zarówno z klas ZwierzeLadowe, jak i ZwierzeMorskie.
- Następnie należy wskazać, że skoro klasa ZwierzeLadowe została wymieniona jako pierwsza wewnątrz nawiasów, w których wskazujemy klasy, z których dziedziczymy, to ma ona priorytet nad klasą ZwierzeMorskie.
- W związku z tym metoda przedstaw_sie, która jest obecna w obu klasach nadrzędnych, będzie dziedziczona **tylko** z klasy ZwierzeLadowe.
- Kolejne dwie metody (biegaj i plywaj) nie występują w obu klasach, zatem każda z nich zostanie odziedziczona z tej klasy, w której występuje.

Pytanie 38
@staticmethod i @classmethod

Do czego służą dekoratory @staticmethod i @classmethod?

Do powtórzenia:

- *dekorator @staticmethod*
- *dekorator @classmethod*

Co sprawdza:

- Czy wiemy, czym jest metoda statyczna i w jakich sytuacjach się ją stosuje.
- Czy wiemy, czym jest metoda klasowa i w jakich sytuacjach się ją stosuje.
- Czy wiemy, jakie słowa kluczowe są używane w definicjach metod statycznych, klasowych i zwykłych.

Jak podejść?

- Aby wyjaśnić, czym są metody statyczne (oznaczane dekoratorem @staticmethod) i klasowe (oznaczane dekoratorem @classmethod), musimy stworzyć klasę, w której będą występowały oba typy metod.
- W klasie tej musimy pokazać, że metody statyczne nie są "świadome" istnienia wewnątrz klasy i z niej nie korzystają, a metody klasowe są "świadome" bycia wewnątrz klasy, ale nie działają na jej obiektach.

Pytanie 39
FizzBuzz

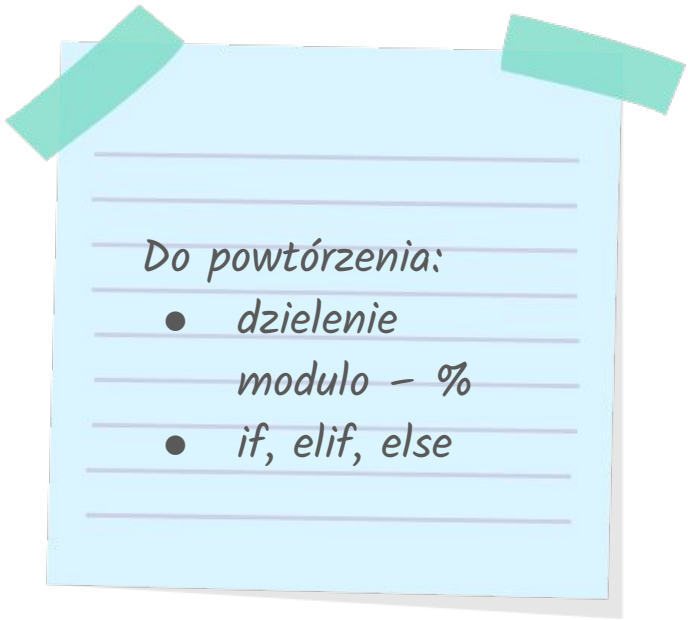
Napisz program, który dla kolejnych liczb z zakresu od 1 do n wypisze:

- "Fizz" – jeśli liczba będzie podzielna przez 3.
- "Buzz" – jeśli liczba będzie podzielna przez 5.
- “ FizzBuzz" – jeśli liczba będzie podzielna przez 3 i 5.

Jeśli nie zajdzie żaden z tych przypadków, to tylko wypisz liczbę.
Zadanie to znane jest pod nazwą: FizzBuzz.

Co sprawdza:

- Czy potrafimy wykorzystać dzielenie modulo do sprawdzania podzielności przez dane liczby.
- Czy potrafimy napisać blok logiczny zawierający słowa if, elif i else.



Jak podejść?

- Zadanie to można rozwiązać na kilka sposobów.
- Jednym z nich jest wypisywanie stringów "Fizz”, "Buzz” i "FizzBuzz”, jeśli znajdziemy liczbę, która dzieli się przez 3, 5 lub 15.
- Drugim jest konstruowanie stringa, do którego doklejamy "Fizz” lub "Buzz”, a "FizzBuzz” tworzy się automatycznie, jeśli string jest podzielny jednocześnie przez 3 i przez 5.

Pytanie 40
Ciąg Fibonacciego

Ciąg Fibonacciego to ciąg liczb, którego:

- Zerowy element wynosi 0.
- Pierwszy element wynosi 1.
- Każdy kolejny element jest sumą dwóch poprzedzających go elementów.

Napisz funkcję, która zwróci n-ty element ciągu Fibonacciego.

Do powtórzenia:

- *funkcje rekurencyjne*
- *bigO rekurencji*
- *warunek zakończenia pracy funkcji rekurencyjnej*

Co sprawdza:

- Czy potrafimy napisać prostą funkcję rekurencyjną, która będzie znajdować określone wyrazy ciągu Fibonacciego.
- Czy wiemy, jaka jest złożoność obliczeniowa takiego rozwiązania.
- Czy potrafimy zaproponować rozwiązanie o lepszej złożoności obliczeniowej.

Jak podejść?

- Klasycznym rozwiązaniem tego zadania jest napisanie funkcji rekurencyjnej, jednak warto wspomnieć o tym, że ma ona bardzo wysoką złożoność obliczeniową.
- Warto zaprezentować też alternatywne rozwiązanie, o złożoności liniowej O(n), w którym w pętli for kolejne wyrazy ciągu są wyznaczane dynamicznie na podstawie poprzednich.

Pytanie 41
Wypisanie zawartości katalogu

Napisz funkcję, która dla podanego katalogu wypisze znajdujące się w nim pliki.
Pamiętaj, że katalog może zawierać podkatalogi, do których funkcja również musi zajrzeć.

Co sprawdza:

- Czy potrafimy napisać funkcję rekurencyjną, która przejrzy zawartość katalogu.
- Czy znamy funkcje z biblioteki standardowej służące do operacji na katalogach.
- Czy znamy funkcję os.path.join() służącą do tworzenia ścieżek poprawnych dla każdego systemu operacyjnego.

Do powtórzenia:

- *funkcje rekurencyjne*
- *funkcje:*
  - *os.listdir()*
  - *os.path.join*
  - *os.path.isdir*

Jak podejść?

- Należy napisać funkcję rekurencyjną, która przejrzy wszystkie pliki wewnątrz podanego katalogu i:
  - Dla plików niebędących katalogami – wypisze ich nazwę.
  - Dla plików będących katalogami – wywoła ponownie samą siebie.

Pytanie 42
Wyszukiwanie binarne

Przy użyciu wyszukiwania binarnego sprawdź, czy liczba 341 znajduje się w posortowanej liście P.

```
P = [-10, -7, -5, -3, 0, 3, 5, 21, 68, 341, 500]
```

Do powtórzenia:

- *algorytm wyszukiwania binarnego*
- *bigO wyszukiwania binarnego*

Co sprawdza:

- Czy znamy i potrafimy zastosować jeden z najczęściej stosowanych w programowaniu algorytmów, jakim jest wyszukiwanie binarne.
- Czy znamy bigO (złożoność obliczeniową) wyszukiwania binarnego i rozumiemy, dlaczego jest to bardzo dobra złożoność.

Jak podejść?

- Aby zastosować algorytm wyszukiwania binarnego, musimy działać na posortowanej sekwencji elementów (tu liczb).
- W każdej iteracji algorytmu sprawdzamy, czy element środkowy sekwencji jest mniejszy, większy czy równy temu, którego szukamy.
- Jeśli element jest mniejszy lub większy, to odrzucamy połowę sekwencji, w której na pewno nie ma szukanej przez nas wartości i ponownie testujemy element będący w środku tej połowy, która nam została.

Pytanie 43
bigO

Uszereguj podane złożoności obliczeniowe algorytmów od najlepszej do najgorszej:

O(1)
O(log(n))
O(n)
O(n * log(n))
O(n**2)
O(2**n)

Do powtórzenia:

- *notacja bigO*
- *złożoność obliczeniowa najpopularniejszych algorytmów*

Co sprawdza:

- Czy wiemy, jak czytać notację bigO.
- Czy potrafimy określić, jaka złożoność jest lepsza, a jaka gorsza?
- Czy potrafimy podać przykłady algorytmów mających określone złożoności obliczeniowe.

Jak podejść?

- Dla każdej z podanych złożoności obliczeniowych warto przytoczyć przykład operacji lub algorytmu, które mają tę konkretną złożoność.
- Wyjaśnić lub nawet naszkicować, jak wielkie różnice występują pomiędzy dobrymi i złymi złożonościami.

Pytanie 44
Dynamiczne typowanie

Co to znaczy, że Python jest językiem dynamicznie typowanym?

Co sprawdza:

- Czy potrafimy zaprezentować w kilku linijkach, że Python jest dynamicznie typowany.
- Czy wiemy, jaka jest różnica między statycznym a dynamicznym typowaniem.
- Czy znamy metodę type hits, która pozwala użyć statycznego typowania w Pythonie.

Do powtórzenia:

- *funkcja type()*
- *type hints – metoda pozwalająca użyć statycznego typowania w Pythonie*

Jak podejść?

- Odpowiadając na to pytanie, warto omówić zalety i wady zarówno statycznego, jak i dynamicznego typowania.
- Warto wspomnieć o tym, dlaczego twórcy Pythona zdecydowali się zastosować w nim dynamiczne typowanie oraz w jakich sytuacjach się ono nie sprawdza się i dlaczego warto rozważyć stosowanie type hints.

Pytanie 45
PEP8

W jaki sposób zadbasz o to, aby twój kod był czytelny i łatwy do zrozumienia dla innych programistów?

Do powtórzenia:

- *najważniejsze punkty dokumentu PEP8 – Style Guide for Python Code*

Co sprawdza:

- Czy wiemy, czym jest PEP8 – Style Guide for Python Code.
- Czy potrafimy wymienić najważniejsze reguły zdefiniowane w PEP8.

Jak podejść?

- Odpowiadając na to pytanie, należy podkreślić rolę pisania czystego, czytelnego kodu w pracy programisty.
- Warto dodać, że jesteśmy świadomi faktu, że programista znacznie częściej czyta kod, niż go pisze, a zatem czytelność kodu jest niezwykle istotna.
- Następnie możemy omówić kilka najważniejszych założeń PEP8 (np. tworzenie opisowych nazw funkcji i zmiennych) i powiedzieć, dlaczego sprawiają one, że kod staje się łatwiejszy do zrozumienia.

Pytanie 46
pip

W jaki sposób importujesz zewnętrzne moduły w Pythonie?

Do powtórzenia:

- *czym są moduły i pakiety*
- *jak korzystać z pip*
- *czym jest pypi.org*

Co sprawdza:

- Czy znamy narzędzie pip i potrafimy zastosować je do instalowania zewnętrznych pakietów/bibliotek?
- Czy znamy stronę pypi.org, czyli największy internetowy zbiór pakietów do Pythona wraz z dokumentacją.

Jak podejść?

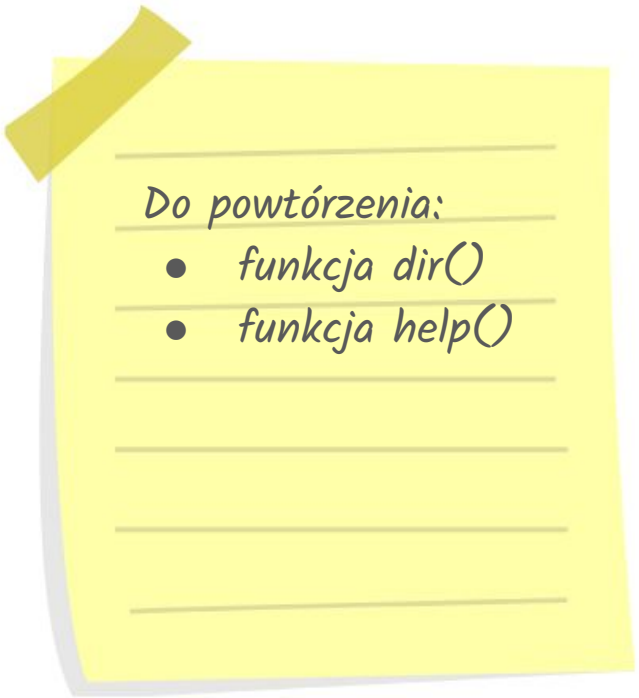
- Wyjaśnić, jaka jest rola zewnętrznych pakietów w Pythonie (szybciej i mądrzej jest użyć sprawdzonej biblioteki napisanej przez kogoś innego, niż tworzyć wszystko od zera).
- Wyjaśnić, gdzie szukać informacji o zewnętrznych pakietach i jak je instalować.

Pytanie 47
dir i help

Do czego w Pythonie służą funkcje dir() i help()?

Co sprawdza:

- Czy wiemy, do czego służy i jak stosować funkcję dir().
- Czy wiemy, do czego służy i jak stosować funkcję help().

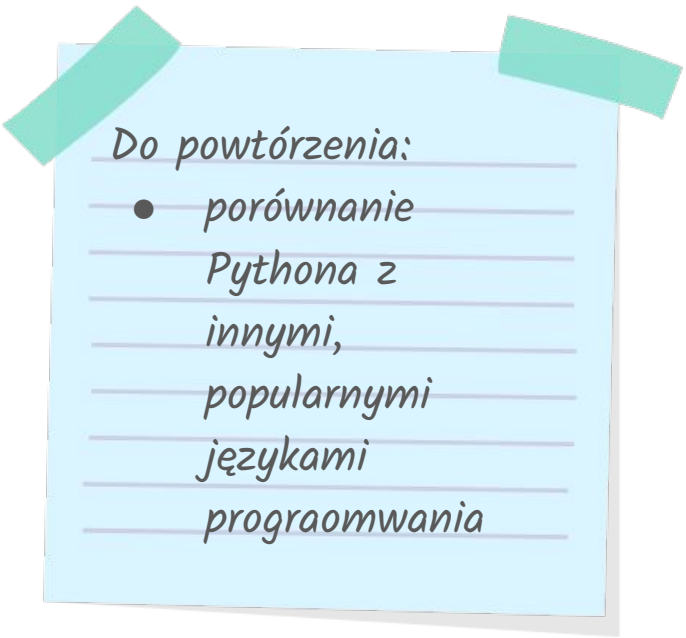


Jak podejść?

- Pytanie ma na celu sprawdzenie, czy jesteśmy samodzielnymi programistami i wiemy, jak szukać informacji na temat obiektów, na których pracujemy (funkcja dir()) i metod, które są dostępne w Pythonie (funkcja help()).
- Warto też wspomnieć, że w obecnych czasach powszechnego dostępu do Internetu i stackoverflow rola funkcji dir() i help() zmalała, bo często łatwiej wyszukać informacje bezpośrednio w sieci.

Pytanie 48
Najważniejsze cechy Pythona

Jakie są najważniejsze cechy Pythona?
Jakie są zalety i wady Pythona?
Dlaczego zdecydowałeś(aś) się programować w Pythonie?



Co sprawdza:

- Czy znamy najważniejsze cechy Pythona (dynamiczne typowanie, prostą składnię, stosowanie wcięć zamiast nawiasów klamrowych).
- Czy wiemy, jaka jest wydajność Pythona w porównaniu z innymi językami.
- Czy jesteśmy świadomi wad i zalet języka.
- Czy potrafimy powiedzieć, w jakich zastosowaniach Python się nie sprawdza.

Jak podejść?

- Odpowiadając na to pytanie, warto zaznaczyć, które z cech Pythona sprawiły, że zdecydowaliśmy się programować właśnie w tym języku.
- Można pozwolić sobie tu na prywatne opinie – co w Pythonie lubimy, a czego nie. Dzięki temu pokażemy, że pracując lub pisząc projekty, nabyliśmy doświadczenie, które pozwoliło nam wyrobić sobie opinię o zaletach i wadach tego języka.

Pytanie 49
Źródła wiedzy

Skąd czerpiesz wiedzę o programowaniu w Pythonie?

Co sprawdza:

- Jakie jest nasze podejście do nauki i samorozwoju.
- Czy śledzimy aktualności związane z Pythonem.
- Czy naprawdę interesujemy się programowaniem w Pythonie, czy tylko chcemy się go nauczyć, aby dostać pracę.

Do powtórzenia:

- *3–4 naszym zdaniem najciekawsze źródła wiedzy o programowaniu w Pythonie*

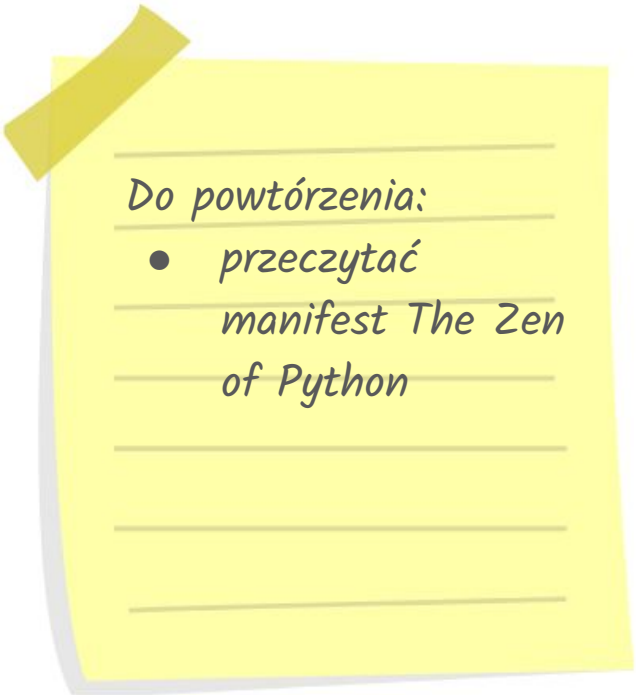
Jak podejść?

- Odpowiadając na to pytanie, mamy okazję pokazać, jakie jest nasze podejście do nauki i ciągłego doskonalenia swojej wiedzy programistycznej.
- Warto wspomnieć o ciekawych, być może mało znanych źródłach wiedzy – rekruter, z którym rozmawiamy, to też programista i z pewnością zainteresuje się poleconymi przez nas stronami czy blogami.
- To pytanie to doskonała okazja, żeby porozmawiać z rekruterem jak kolega z kolegą, pokazać, że możemy wnieść do przyszłego zespołu nie tylko pracę, ale też wiedzę, z której skorzystają inni programiści.

Pytanie 50
The Zen of Python

Co się stanie po uruchomieniu poniższego kodu?

```
import this
```



Co sprawdza:

- Czy interesujemy się Pythonem na tyle, aby znać ukryty w nim "*easter egg*", którym jest efekt wykonania kodu `import this`.
- Czy przeczytaliśmy The Zen of Python i potrafimy wymienić kilka z zawartych w nim zasad.

Jak podejść?

- Uruchomienie kodu `import this` – z pliku lub wprost z konsoli – powoduje wyświetlenie manifestu stworzonego przez twórców Pythona, zatytułowanego The Zen of Python.
- The Zen of Python to zbiór porad na temat pisania czytelnego kodu i mądrego programowania.
- Pryncypia zawarte w The Zen of Python są bardzo wartościowe i na rozmowie powinniśmy być w stanie przytoczyć przynajmniej kilka z nich – powiedzieć, dlaczego uważamy je za ważne i jak wdrażamy je podczas programowania.

Kilka słów na koniec

Gratulacje!

Jeśli dotarłeś do tego miejsca, to prawdopodobnie przerobiłeś (lub przerobiłaś) wszystkie 50 pytań.

Z pewnością nie było łatwo. Wielu z zamieszczonych tu zadań nie rozwiązuje się w 5 minut, wiele wymaga chwili zastanowienia i całej serii prób i błędów, aby w końcu uzyskać poprawne i dobrze napisane rozwiązanie.

Mam jednak dobrą wiadomość. Podziękujesz sobie za cały ten wysiłek, gdy podczas rozmowy kwalifikacyjnej staniesz przed tablicą i zostaniesz poproszony o rozwiązanie zadania, które robiłeś już wcześniej. Gwarantuję, że to bardzo przyjemny moment, gdy zamiast dać się zaskoczyć rekruterowi pytaniem, to Ty zaskakujesz go tym, jak dobrze jesteś przygotowany.

A teraz czas umawiać się na rozmowy!

Jestem przekonana, że pójdzie Ci dobrze.

Zajrzyj do mnie

Jeśli spodobał Ci się ten ebook i masz ochotę widywać się ze mną częściej, to zapraszam do siebie!

Na Facebooku umieszczam informacje o nowych projektach, którymi się zajmuję. Czasem wrzucam też krótkie filmiki na tematy związane z nauką i zdobyciem pierwszej pracy jako programista.

Na swojej stronie publikuję bardziej rozbudowane artykuły i analizy. Będę zajmować się tu takimi kwestiami jak na przykład jak i gdzie się uczyć (i czy warto wydawać pieniądze na bootcampy), a także mam zamiar szczegółowo omówić cały proces, jaki musisz przejść, by zostać programistą. Uważam, że wymaga on aż 8 kroków, a większość ludzi skupia się tylko na jednym – nauce języka. Mam zamiar powalczyć z tym przekonaniem i pokazać, co jeszcze trzeba zrobić, aby skutecznie zmienić pracę.

Na Udemy znajduje się mój kurs, w którym pokazuję szczegółowe rozwiązania wszystkich zadań, które pojawiły się w tym ebooku. Jeśli chcesz, żebym przeprowadziła Cię przez nie krok po kroku, to koniecznie tam zajrzyj!