

DOKUMENTACJA PROJEKTU W RAMACH PRZEDMIOTU PROGRAMOWANIE W JĘZYKU PYTHON.

Autor:

Filip Maciborski

Tytuł projektu:

Vocabulary tester

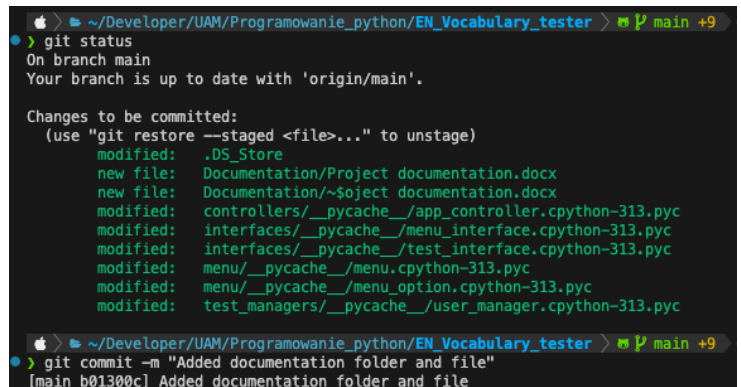
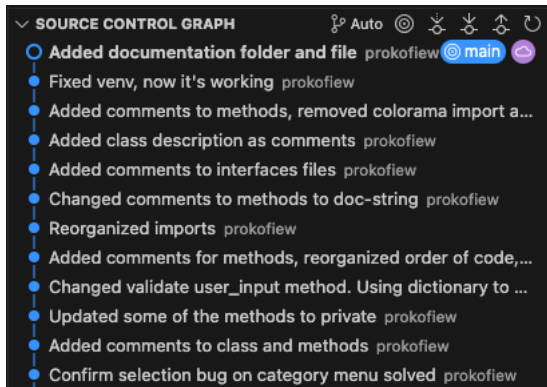
Tematy uwzględnione w projekcie:

1. Git.
2. Czytelny kod w Pythonie – PEP8.
3. Zaawansowane struktury danych – moduł Pandas.
4. Obsługa dat i czasu – moduł DateTime.
5. Programowanie funkcyjne.
6. Programowanie obiektowe.

1. Git i GitHub

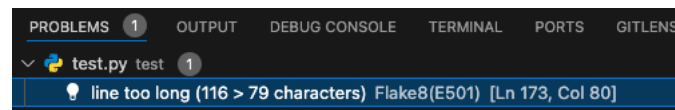
Projekt od pewnego momentu przechowywany był na GitHubie i także za jego pośrednictwem został przekazany wykładowcy. W trakcie pracy nad projektem, choć dopiero od pewnego momentu stosowany był Git.

Git został zainstalowany oraz odpowiednio skonfigurowany. Został także połączony poprzez klucz SSH z GitHubem.



2. Czytelny kod w Pythonie – PEP8.

Nad projektem pracowałem z wykorzystaniem Visual Studio Code. Jako dodatek zainstalowałem *Flake8* podkreślający wszystkie możliwe błędy jakie występują w kodzie niezgodnym z *PEP8* np.:



Zastosowałem się do wszystkich wskazań dodatku i dostosowałem kod do *PEP8*.

Przykłady (na podstawie klasy *NewTest*):

⇒ Dzielenie długich stringów na linie i stosowanie odpowiednich wcięć (klasa *NewTest*):

```
155     def start_test(self):
156         """ Starts the actual test,
157             displays summary of users choices,
158             displays important information about test"""
159
160         if self.test_language_version and self.selected_category:
161             Menu.clear_console()
162             print(f"Starting test.\nChosen options:\n"
163                 f"1. Translating from: {self.test_language_version}.\n"
164                 f"2. Category: "
165                 f"{self.selected_category if isinstance(
166                     self.selected_category, str) else ', '.join(
167                     map(str, self.selected_category))}.\n"
168                 f"3. Number of questions: {self.questions_amount}.\n"
169                 f"4. Test time limit: "
170                 f"{self.test_time_limit_in_seconds // 60} min.\n")
```

⇒ 'Zawijanie' długich wywołań funkcji z zachowaniem wcięć:

```
85         # creating list of categories sorted by category_id
86         categories = (
87             self.data[["category_id", "category_name"]]
88             .drop_duplicates()
89             .sort_values("category_id")
90             .values
91             .tolist()
92     )
```

⇒ Komentarze opisujące zadania/działanie metod jako multiline doc-string:

```
210     def get_results(self, correct_answers, user_answers, expressions):
211         """ gets questions, correct answers
212            and user answers into a DataFrame, normalizes text,
213            compares test data and points wrong and correct answers,
214            modifies DataFrame to display text information (map),
215            calculates points and percentage"""
216
```

⇒ Prawidłowe uporządkowanie argumentów funkcji w formie listy (klasa *NewTest*):

```
251     def end_test(self, test_data):
252         """ passing data to result manger """
253         result_manager = ResultManager(
254             self.test_datetime,
255             self.user_name,
256             self.point_score,
257             self.questions_amount,
258             self.percentage_score,
259             self.test_duration,
260             self.test_time_limit_in_seconds,
261             test_data
262         )
```

⇒ Uporządkowane importy (klasa *NewTest*):

```
1  import datetime
2  import pandas as pd
3
4  from colorama import Fore
5  from interfaces.test_interface import Test
6  from menu.menu import Menu
7  from menu.menu_option import MenuOption
8  from test_managers.text_formatter import TextFormatter
9  from test_managers.question_manager import QuestionManager
10 from test_managers.result_manager import ResultManager
11 from test_managers.time_manager import TimeManager
12 from test_managers.file_manager import FileManager
13 from test_managers.user_manager import UserManager
14
```

⇒ Prawidłowe nazewnictwo klas i metod:

```
16 class NewTest(Test):
17     def __init__(self, data_file, data, main_menu):
18         self.main_menu = main_menu # allowing user to use main menu
19         self.data_file = data_file # database file path
20         self.data = data # Data from database as DataFrame
21         self.test_language_version = None
22         self.selected_category = None
```

```

76
77     def choose_all():
78         self.selected_category = "All categories"
79         self.question_manager.set_category(self.selected_category)
80         self.__get_questions_amount()
81
82     def back_to_language():
83         self.__set_language()

```

3. Zaawansowane struktury danych – moduł Pandas.

W projekcie został wykorzystany moduł *Pandas*. W największym zakresie wykorzystana została *DataFrame* w oparciu, o który program funkcjonuje:

- ⇒ Praca z plikiem Excel i arkuszami pliku w których została utworzona baza danych programu (klasa *AppController*):

Pobranie danych z pliku:

```

112     def __data_load(self):
113         """ Loads database file
114         and establishes sheets as properties """
115         try:
116             file_data = pd.ExcelFile(TEST_DATABASE)
117             self.dictionaries = file_data.parse(sheet_name="categories")
118             self.vocabulary = file_data.parse(sheet_name="vocabulary")
119             self.data = pd.merge(
120                 self.vocabulary,
121                 self.dictionaries,
122                 left_on="category",
123                 right_on="category_id")
124         except FileNotFoundError:
125             print("Database file: tester_database.xlsx not found.")
126             sys.exit()
127

```

Zapis danych do pliku:

```

143     def __save_to_database(self, data_frame, sheet_name):
144         """ Updating database file with new vocabulary"""
145         with pd.ExcelWriter(
146             TEST_DATABASE, mode="a", if_sheet_exists="overlay"
147         ) as writer:
148             data_frame.to_excel(writer, sheet_name=sheet_name, index=False)

```

- ⇒ Wykorzystanie *DataFrame* do zarządzania przepływem danych, podczas tworzenia pytań i zbierania odpowiedzi użytkownika, dodawania nowych danych do bazy danych.

Łączenie danych:

```

138     def __join_data_frames(self, base_data_frame, added_data_frame):
139         """ Private method to join DataFrames with pd.concat """
140         return pd.concat([
141             base_data_frame, added_data_frame], ignore_index=True)

```

Tworzenie nowej kategorii słów:

```
150 def __create_new_category(self, new_category):
151     """ Creating new category_id and
152     DataFrame for new vocabulary"""
153     new_category_id = int(self.dictionaries["category_id"].max() + 1)
154     new_category_data_frame = pd.DataFrame(
155         {
156             "category_id": [new_category_id],
157             "category_name": [new_category]
158         }
159     )
```

Operacje na kolumnach w celu pozyskania wyników testu (klasa *NewTest*):

- **map()** – zmiana wartości bazując w serii na podstawie podanej innej wartości
- **apply()** – stosowanie funkcji do wierszy lub kolumn
- **astype()** - zmiana typu danych w series
- **mean()** – średnia procentowa kolumny
- **sum()** – obliczenie sumy kolumny

```
def get_results(self, correct_answers, user_answers, expressions):
    """ gets questions, correct answers and user answers into a DataFrame, normalizes text,
    compares test data and points wrong and correct answers,
    modifies DataFrame to display text information (map),
    calculates points and percentage"""

    test_data = pd.DataFrame({
        "Questions": expressions,
        "Correct answers": correct_answers,
        "Your answers": user_answers,
    })

    # Normalize text
    test_data["Normalized Correct"] = test_data[
        "Correct answers"].apply(TextFormatter.normalize_text)
    test_data["Normalized User"] = test_data[
        "Your answers"].apply(TextFormatter.normalize_text)

    # Comparing data
    test_data["Correct/Wrong"] = test_data[
        "Normalized Correct"] == test_data["Normalized User"]
    test_data["Points"] = test_data["Correct/Wrong"].astype(int)

    # Map to text
    test_data["Correct/Wrong"] = test_data[
        "Correct/Wrong"].map({True: "Correct", False: "Wrong"})

    # Calculating the results
    self.percentage_score = test_data["Points"].mean() * 100
    self.point_score = test_data["Points"].sum()

    return test_data
```

4. Obsługa dat i czasu – moduł *DateTime*.

Moduł *DateTime* został w programie wykorzystany do pobrania daty i godziny przeprowadzenia testu, a także zmierzenia czasu jego trwania. W oparciu o limit czasu ustawiany przez użytkownika w minutach, obliczany jest limit czasu na sekundy. Stworzony dekorator mierzy czas trwania testu. Na podstawie porównania wartości określone jest to czy użytkownik zmieścił się w limicie czasowym.

Przykłady (klasa *NewTest*):

- ⇒ Pobranie daty i czasu podczas tworzenia instancji klasy *NewTest* – wybranie z menu głównego opcji -> ‘Start New Test’

```
16 class NewTest(Test):
17     def __init__(self, data_file, data, main_menu):
18         self.main_menu = main_menu # allowing user to use main menu
19         self.data_file = data_file # database file path
20         self.data = data # Data from database as DataFrame
21         self.test_language_version = None
22         self.selected_category = None
23         self.test_datetime = datetime.datetime.now()
```

- ⇒ Ustalenie limitu czasu dla testu – bez wykorzystania *DateTime*, ale ważne w późniejszym użyciu (klasa *TimeManager*)

```
36 def set_test_time_limit(self):
37     try:
38         test_time_limit = int(input("Enter test time limit in minutes: "))
39         self.test_time_limit_in_seconds = test_time_limit * 60
40         return True
41     except ValueError:
42         message = "Invalid value. Enter a number"
43         print(self.text_formatter.colorize(message, Fore.RED))
44         self.display_sleep(1.5)
45         return self.set_test_time_limit()
```

- ⇒ Pomiar czasu trwania testu (wrapper w *TimeManager*):

```
13 @staticmethod
14 def measure_time(func):
15     @wraps(func)
16     def wrapper(instance, *args, **kwargs):
17         start_time = datetime.datetime.now()
18         result = func(instance, *args, **kwargs)
19         end_time = datetime.datetime.now()
20         instance.test_duration = (end_time - start_time).total_seconds()
21         return result
22     return wrapper
```

- ⇒ Pomiar czasu uruchomiony dla funkcji, która wyświetla pytania i pobiera odpowiedzi użytkownika, czyli w momencie, gdy użytkownik przechodzi do rozwiązywania testu (klasa *NewTest*, metoda *def submit_answer()* wywołana w metodzie *def start_test()*):

```
188 @TimeManager.measure_time # TimeManager decorator to measure test time
189 def submit_answer(self, questions_data):
190     """ responsible for displaying questions and gathering answers,
191     checking if test was stopped"""
192     user_answers = []
193     print(self.text_formatter.colorize(
194         """ To stop test, enter: \"Stop test\" ***\n\",
195         Fore.LIGHTYELLOW_EX))
```

- ⇒ Wyświetlanie poszczególnych elementów *DateTime*, osobno daty i czasu w odpowiednim formacie (klasa *ResultManager* metoda *def display_test_outcome()*):

```
44     def display_test_outcome(self):
45         print(f"Test date: {self.test_datetime.strftime('%d-%m-%Y')}")
46         print(f"Test time: {self.test_datetime.strftime('%H:%M:%S')}")
47         print(f"User: {self.user_name}")
```

5. Programowanie funkcyjne.

Program wykorzystuje paradygmat programowania funkcyjnego. Jest to poniekąd efekt tego, że zastosowałem podejście obiektowe.

- ⇒ Funkcja wyższego rzędu – funkcja przyjmująca inną funkcję jako argument (wrapper w klasie *TimeManager*):

```
13     @staticmethod
14     def measure_time(func):
15         @wraps(func)
16         def wrapper(instance, *args, **kwargs):
17             start_time = datetime.datetime.now()
18             result = func(instance, *args, **kwargs)
19             end_time = datetime.datetime.now()
20             instance.test_duration = (end_time - start_time).total_seconds()
21             return result
22         return wrapper
```

- ⇒ Funkcja wyższego rzędu, przyjmuje tekst i kolor jako argument, a następnie zwraca zmodyfikowany tekst – przetwarzanie danych w stylu funkcyjnym (klasa *TextFormatter*).

```
22     def colorize(self, text, color):
23         """
24         Applies the specified color to the text.
25         """
26         return f"{color}{text}{Style.RESET_ALL}" if color else text
```

- ⇒ Funkcja czysta, nie modyfikuje stanu obiektu, wynik funkcji zależy wyłącznie od przekazanych argumentów, zwraca ten sam wynik dla tych samych danych wejściowych (klasa *TextFormatter*):

```
9      @staticmethod
10     def normalize_text(text):
11         """
12         Removes diacritics, converts to lowercase, and strips whitespace.
13         """
14         if not isinstance(text, str):
15             return text
16
17         # Remove leading and trailing whitespace
18         stripped_text = text.strip()
19
20         # Normalize text to decompose diacritic characters
21         normalized_text = unicodedata.normalize("NFD", stripped_text)
22
23         # Remove diacritic marks
24         without_diacritics = "".join(
25             char for char in normalized_text
26             if unicodedata.category(char) != "Mn"
27         )
28
29         # Replace specific characters and convert to lowercase
30         final_text = without_diacritics.replace('ł', 'l').lower()
31
32         return final_text
```

- ⇒ Funkcja czysta, jedynym efektem jest wykonanie polecenia systemowego, nie zależy od stanu obiektu i go nie modyfikuje (klasa *Menu*):

```
20     @staticmethod
21     def clear_console():
22         if os.name == "nt":
23             os.system("cls")
24         else:
25             os.system("clear")
```


6. Programowanie obiektowe.

Program wykorzystuje kilka cech programowania obiektowego. M.in.:

- ⇒ Interfejsy i ich implementacja. Wykorzystane zostały dwa interfejsy (klasy abstrakcyjne), które wymuszają implementacje konkretnych metod podczas tworzenia klasy implementującej interfejs:

```
1 from abc import ABC
2 from abc import abstractmethod
3
4 # Test interface with abstract methods
5
6
7 class Test(ABC):
8
9     @abstractmethod
10     def start_test(self):
11         pass
12
13     @abstractmethod
14     def get_questions_and_answers_data(self):
15         pass
16
17     @abstractmethod
18     def submit_answer(self, answer):
19         pass
20
21     @abstractmethod
22     def get_results(self):
23         pass
24
25     @abstractmethod
26     def end_test(self):
27         pass
28
29     @abstractmethod
30     def save_results(self):
31         pass
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

- ⇒ Enkapsulacja. Zarówno niektóre metody oraz pola klas ukrywane są przed użytkownikiem i dostęp do nich jest ograniczony, gdyż z punktu widzenia użytkownika szczegóły implementacji nie są istotne lub ważne jest by użytkownik lub inne klasy nie mogły ich swobodnie modyfikować (klasa *NewTest*):

```
21 self.__test_language_version = None
22 self.__selected_category = None
23 self.__test_datetime = datetime.datetime.now()
24 self.__point_score = 0
25 self.__percentage_score = 0
26 self.__user_name = None
27 self.__test_time_limit_in_seconds = None
```

Użytkownik nie ma w ogóle możliwości modyfikacji np. `self.__point_score` lub `self.__percentage_score`. Pola te modyfikowane są przez odpowiednie metody w klasie.

Użytkownik ma możliwość modyfikowania pól `self.__user_name` oraz `self.__test_language` ale odbywa się to za pośrednictwem odpowiednich metod, które mają dostęp do tych pól. Same metody są również prywatne.

- W tym przypadku niektóre klasy korzystają z metod wewnętrznie i nie ma potrzeby by były dostępne spoza klasy. Metoda `def __initiate_language_menu()` wywoływana jest w konstruktorze i służy do utworzenia menu wyboru języka. Żadna inna klasa nie musi z tej metody korzystać.

```

33     self.__initiate_language_menu()
34     self.__initiate_category_menu()
35     self.__initiate_test()
36
37     def __initiate_language_menu(self):
38         """ initiates language menu,
39         if the language is choosen,
40         directs user to category setup"""
41         def choose_en():
42             self.__test_language_version = "EN"
43             self.__set_category()
44
45         def choose_pl():
46             self.__test_language_version = "PL"
47             self.
48             (variable) __main_menu: Any
49             __main_menu
49         def back
50             self.__main_menu.display()
51
52         self.language_menu = Menu(
53             title="Choose Test Language",
54             controller=None)
55
56         self.language_menu.add_option(
57             1, MenuOption("EN -> PL", action=choose_en))
58         self.language_menu.add_option(
59             2, MenuOption("PL -> EN", action=choose_pl))
60         self.language_menu.add_option(
61             3, MenuOption("Back to Main Menu", action=back_to_main))

```

⇒ Kompozycja. Najlepszym przykładem jest klasa *NewTest*, która korzysta z kilku innych klas w celu obsługi najważniejszych elementów testu:

```

28     self.__text_formatter = TextFormatter()
29     self.__question_manager = QuestionManager(data, self.__text_formatter)
30     self.__time_manager = TimeManager()
31     self.__file_manager = FileManager()
32     self.__user_manager = UserManager()

```

Klasa ta w konstruktorze inicjuje kilka obiektów które realizują:

1. **TextFormatter** – kolorowanie i normalizacja tekstu odpowiedzi, tak aby odpowiedzi mogły być udzielane w języku polskim bez ‘polskich znaków’
2. **Questionmanager** – odpowiada za określenie ilości pytań i stworzenie list pytań i odpowiedzi.
3. **TimeManager** – odpowiada za ustalanie limitu czasu testu, mierzenie czasu przebiegu testu, odliczanie do rozpoczęcia testu i utrzymanie uśpienia ekranu np. podczas wyświetlania komunikatów o błędach
4. **FileManager** – odpowiada za zapis danych do pliku
5. **UserManager** – odpowiada za interakcję z użytkownikiem, ustawienie imienia, walidację wprowadzanych przez niego wartości w terminalu.