Dr inż. Dariusz Michalski. Formularz samooceny do projektu z języków skryptowych

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nr | Obszar | Wymaganie | KOD |  | Przyznane | Pkt |
|  |  |  |  |  | pkt | max |
| 1 | UI | JEST |  | ☐ |  |  |
|  |  | Wprowadzanie | name = input("Nazwa ćwiczenia: ") muscle = input("Grupa mięśniowa: ") | ☐ |  |  |
|  |  | danych |  |  |  | 2 |
|  |  | Wyświetlanie | for i, plan in enumerate(plans):  print(f"{i + 1}. {plan.name}") | ☐ |  |  |
|  |  | danych |  |  |  | 2 |
|  |  | Zmiana danych | sessions.append(session) | ☐ |  | 2 |
|  |  | Wyszukiwanie | while selected:  match = next((e for e in exercises if e.name == selected), None)  if match:  plan.add\_exercise(match)  print("✓ Dodano.") | ☐ |  |  |
|  |  | danych |  |  |  | 2 |
|  |  | Przedstawienie | def show\_statistics(sessions):  total\_volume = reduce(lambda acc, s: acc + sum(x.reps \* x.weight for x in s.series), sessions, 0)  print(f"\nCałkowita objętość treningowa: {total\_volume:.2f} kg") | ☐ |  |  |
|  |  | wyników |  |  |  | 2 |
| 2 | Podstawy | Zmienne | exercises, plans = load\_data() sessions = load\_sessions() | ☐ |  | 2 |
|  |  | typy danych | def \_\_init\_\_(self, name: str, muscle\_group: str): | ☐ |  | 2 |
|  |  | komentarze | # Klasa reprezentująca plan treningowy z listą ćwiczeń class WorkoutPlan:  def \_\_init\_\_(self, name: str):  self.name = name  self.exercises = [] | ☐ |  | 1 |
|  |  | operatory | total\_volume = reduce(lambda acc, s: acc + sum(x.reps \* x.weight for x in s.series), sessions, 0) | ☐ |  | 1,5 |
|  |  | Instrukcje |  | ☐ |  |  |
|  |  | warunkowe (if, elif, | if choice == '1':  name = input("Nazwa ćwiczenia: ")  muscle = input("Grupa mięśniowa: ")  exercises.append(Exercise(name, muscle))  print("✓ Ćwiczenie dodane.")  elif choice == '2':  plan\_name = input("Nazwa planu: ")  plan = WorkoutPlan(plan\_name)  for ex in exercises:  print(f"- {ex.name} ({ex.muscle\_group})")  selected = input("Wpisz nazwę ćwiczenia do dodania (Enter = koniec): ") |  |  |  |
|  |  | else) |  |  |  | 3 |
|  |  | Instrukcje iteracyjne |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  | for | for ex in exercises:  print(f"- {ex.name} ({ex.muscle\_group})") | ☐ |  | 2 |
|  |  | while | while selected:  match = next((e for e in exercises if e.name == selected), None)  if match:  plan.add\_exercise(match)  print("✓ Dodano.")  else:  print("️Nie znaleziono ćwiczenia.")  selected = input("Dodaj kolejne ćwiczenie (Enter = koniec): ") | ☐ |  | 2 |
|  |  | Operacje wejścia | choice = input("Wybierz opcję: ") | ☐ |  |  |
|  |  | (input) |  |  |  | 1,5 |
|  |  | Operacje wyjścia | print("️Nie znaleziono ćwiczenia.") | ☐ |  |  |
|  |  | (print) |  |  |  | 1,5 |
|  |  | Funkcje z | def load\_data():  exercises = []  plans = []  try:  # odczyt ćwiczeń  with open(os.path.join(DATA\_DIR, 'exercises.json'), 'r') as f:  exercises\_json = json.load(f)  exercises = [Exercise(\*\*e) for e in exercises\_json]  # odczyt planów i przypisanych ćwiczeń  with open(os.path.join(DATA\_DIR, 'plans.json'), 'r') as f:  plans\_json = json.load(f)  for p in plans\_json:  plan = WorkoutPlan(p['name'])  for e in p['exercises']:  plan.add\_exercise(Exercise(\*\*e))  plans.append(plan)  except Exception as e:  print(f"Błąd odczytu danych: {e}")  return exercises, plans | ☐ |  |  |
|  |  | parametrami i |  |  |  |  |
|  |  | wartościami |  |  |  |  |
|  |  | zwracanymi |  |  |  | 2 |
|  |  | Funkcje | def create\_plan\_from\_json(plan\_json):  plan = WorkoutPlan(plan\_json['name'])  for e in plan\_json['exercises']:  if 'exercises' in e: # podplan (rekurencja)  subplan = create\_plan\_from\_json(e)  plan.add\_exercise(subplan)  else:  plan.add\_exercise(Exercise(\*\*e))  return plan | ☐ |  |  |
|  |  | rekurencyjne |  |  |  | 3 |
|  |  | Funkcje przyjmujące |  | ☐ |  |  |
|  |  | inne funkcje jako |  |  |  |  |
|  |  | argumenty | def filter\_exercises(plan, predicate):  """  Rekurencyjnie zwraca listę ćwiczeń z planu i podplanów,  które spełniają warunek zdefiniowany w predicate (funkcja przyjmująca Exercise i zwracająca bool).  """  filtered = []  for e in plan.exercises:  if isinstance(e, WorkoutPlan): # podplan — rekurencyjnie  filtered.extend(filter\_exercises(e, predicate))  else:  if predicate(e):  filtered.append(e)  return filtered |  |  | 3 |
|  |  | Dekoratory | def log\_start\_end(func):  def wrapper(\*args, \*\*kwargs):  print(f"Rozpoczynam funkcję '{func.\_\_name\_\_}'...")  result = func(\*args, \*\*kwargs)  print(f"Funkcja '{func.\_\_name\_\_}' zakończona.")  return result  return wrapper | ☐ |  | 1,5 |
| 3 | Kontenery | Użycie listy | self.exercises = [] | ☐ |  | 2 |
|  |  | Użycie słownika | json.dump([  {'name': p.name, 'exercises': [e.\_\_dict\_\_ for e in p.exercises]}  for p in plans ], f) | ☐ |  | 2 |
|  |  | Użycie zbioru | for i, ex in enumerate(exercises):  print(f"{i + 1}. {ex.name} ({ex.muscle\_group})") | ☐ |  | 1,5 |
|  |  | Użycie krotki | unique\_muscles = set(e.muscle\_group for e in exercises) | ☐ |  | 1,5 |
| 4 | Przestrzenie | Zastosowano | def main():  exercises, plans = load\_data()  sessions = load\_sessions() | ☐ |  |  |
|  | nazw | zmienne lokalne |  |  |  | 1,5 |
|  |  | Zastosowano | BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  DATA\_DIR = os.path.join(BASE\_DIR, 'data') | ☐ |  |  |
|  |  | zmienne globalne |  |  |  | 1,5 |
|  |  | Zastosowano | def save\_data(exercises, plans):  try:  os.makedirs(DATA\_DIR, exist\_ok=True)  with open(...) as f:  ...  except Exception as e:  print(f"Błąd: {e}") | ☐ |  |  |
|  |  | zakresy funkcji |  |  |  | 1,5 |
|  |  | Zastosowano | self.name = name  self.muscle\_group = muscle\_group | ☐ |  |  |
|  |  | zakresy klas |  |  |  | 1,5 |
| 5 | Moduły i | Projekt podzielony | from models.exercise import Exercise from models.workout\_plan import WorkoutPlan, TrainingSession from utils.file\_handler import save\_data, load\_data, save\_sessions, load\_sessions from utils.analysis import show\_statistics, filter\_exercises | ☐ |  |  |
|  | pakiety | na moduły (import, |  |  |  |  |
|  |  | \_\_init\_\_) |  |  |  | 2 |

Strona **1** z **3**

Dr inż. Dariusz Michalski. Formularz samooceny do projektu z języków skryptowych

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nr | Obszar | Wymaganie | KOD |  | Przyznane | Pkt |
|  |  |  |  |  | pkt | max |
|  |  | Własne | Utils/file\_handler.py | ☐ |  |  |
|  |  | pakiety/funkcje |  |  |  |  |
|  |  | pomocnicze w |  |  |  |  |
|  |  | osobnych plikach |  |  |  |  |
|  |  | .py |  |  |  | 2 |
| 6 | Obsługa | Obsługa wyjątków | def load\_data():  exercises = []  plans = []  try:  # odczyt ćwiczeń  with open(os.path.join(DATA\_DIR, 'exercises.json'), 'r') as f:  exercises\_json = json.load(f)  exercises = [Exercise(\*\*e) for e in exercises\_json]  …..  except Exception as e:  print(f"Błąd odczytu danych: {e}") | ☐ |  |  |
|  | błędów | (try, except, finally) |  |  |  | 2 |
|  |  | Użycie assert do | assert name.strip(), "Nazwa ćwiczenia nie może być pusta" | ☐ |  |  |
|  |  | testów i walidacji |  |  |  | 1,5 |
| 7 | Łańcuchy | Operacje na |  | ☐ |  |  |
|  | znaków | stringach (m.in. | print(f"{i + 1}. {plan.name}")  print(f"✓ Usunięto plan: {removed.name}") |  |  |  |
|  |  | formatowanie, |  |  |  |  |
|  |  | dzielenie, |  |  |  |  |
|  |  | wyszukiwanie) |  |  |  | 2 |
| 8 | Obsługa plików | Odczyt z plików .txt, |  | ☐ |  |  |
|  |  | .csv, .json, .xml |  |  |  |  |
|  |  | (min. 1) |  |  |  | 2 |
|  |  | Zapis do plików .txt, | with open(os.path.join(DATA\_DIR, 'exercises.json'), 'r') as f:  exercises\_json = json.load(f) | ☐ |  |  |
|  |  | .csv, .json, .xml |  |  |  |  |
|  |  | (min. 1) |  |  |  | 2 |
| 9 | OOP | Klasy | class WorkoutPlan: | ☐ |  | 2 |
|  |  | Metody | def add\_exercise(self, exercise):  self.exercises.append(exercise) | ☐ |  | 2 |
|  |  | Konstruktory | class Exercise:  def \_\_init\_\_(self, name, muscle\_group):  self.name = name  self.muscle\_group = muscle\_group | ☐ |  | 2 |
|  |  | Dziedziczenie |  | ☐ |  | 2 |
| 10 | Programowanie | map | exercises = map(lambda s: s.exercise, filtered\_series) | ☐ |  | 1,5 |
|  | funkcyjne | filter | filtered\_series = filter(lambda s: s.weight > min\_weight, all\_series) | ☐ |  | 1,5 |
|  |  | lambda | total\_volume = reduce(lambda acc, s: acc + sum(x.reps \* x.weight for x in s.series), sessions, 0) | ☐ |  | 1,5 |
|  |  | reduce | total\_volume = reduce(lambda acc, s: acc + sum(x.reps \* x.weight for x in s.series), sessions, 0) | ☐ |  | 1,5 |
| 11 | Wizualizacja | Wygenerowano | plt.figure(figsize=(10, 6))  plt.bar(exercises, weights, color='skyblue')  plt.xticks(rotation=45, ha='right')  plt.ylabel("Całkowity ciężar [kg]")  plt.title("Całkowity ciężar podniesiony dla ćwiczeń")  plt.tight\_layout()  plt.savefig(filename)  plt.close()  print(f"Wykres zapisano do pliku: {filename}") | ☐ |  |  |
|  | danych | wykres (np. |  |  |  |  |
|  |  | matplotlib, seaborn) |  |  |  | 2 |
|  |  | Zapisano wykres do | plt.savefig(filename) | ☐ |  |  |
|  |  | pliku graficznego |  |  |  |  |
|  |  | (.png lub .jpg) |  |  |  | 1,5 |
| T12 | Testowanie | Testy jednostkowe |  | ☐ |  |  |
|  |  | (assert, unittest, | class TestExercise(unittest.TestCase):  def test\_exercise\_creation(self):  ex = Exercise("Przysiad", "Nogi")  self.assertEqual(ex.name, "Przysiad")  self.assertEqual(ex.muscle\_group, "Nogi") |  |  |  |
|  |  | pytest) |  |  |  | 1,5 |
|  |  | Testy funkcjonalne | class TestFunctionalFlow(unittest.TestCase):  def test\_full\_workflow(self): | ☐ |  | 1,5 |
|  |  | Testy Integracyjne | class TestFileHandler(unittest.TestCase):  def test\_save\_and\_load\_data(self): | ☐ |  | 1,5 |
|  |  | Testy graniczne / | class TestEdgeCases(unittest.TestCase):  def test\_invalid\_reps\_weight(self):  with self.assertRaises(TypeError):  Serie("przysiad", "dziesięć", "sto") | ☐ |  |  |
|  |  | błędne dane |  |  |  | 1,5 |
|  |  | Testy wydajności | class TestPerformance(unittest.TestCase):  def test\_show\_statistics\_performance(self): | ☐ |  |  |
|  |  | (np. czas wykonania, |  |  |  |  |
|  |  | timeit) |  |  |  | 1,5 |
|  |  | Testy pamięci | class TestMemoryUsage(unittest.TestCase):  def test\_memory\_usage\_show\_statistics(self): | ☐ |  |  |
|  |  | memory\_profiler |  |  |  | 1,5 |
|  |  | Test jakości kodu | class TestCodeQuality(unittest.TestCase):  def test\_pylint\_score(self): | ☐ |  |  |
|  |  | (flake8, pylint) |  |  |  | 1,5 |
| 13 | Wersjonowanie | Repozytorium GIT |  | ☐ |  | 1 |
|  |  | Historia commitów |  | ☐ |  | 1 |

Strona **2** z **3**

Dr inż. Dariusz Michalski. Formularz samooceny do projektu z języków skryptowych

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Nr | Obszar | Wymaganie | KOD |  |  | Przyznane | Pkt |
|  |  |  |  |  |  | pkt | max |
|  |  | Link do GitHub |  |  | ☐ |  | 1 |
|  |  | Opis commitów |  |  | ☐ |  | 1 |
| 14 | Dokumentacja | Plik README.md |  |  | ☐ |  |  |
|  |  | (cel, autorzy, |  |  |  |  |  |
|  |  | uruchamianie) |  |  |  |  | 1,5 |
|  |  | Przykładowe dane |  |  | ☐ |  |  |
|  |  | wejściowe i |  |  |  |  |  |
|  |  | wyjściowe |  |  |  |  | 2 |
|  |  | Diagram klas lub | LiftLog/  ├── main.py  ├── models/  │ ├── \_\_init\_\_.py  │ ├── exercise.py  │ ├── workout\_plan.py  ├── utils/  │ ├── \_\_init\_\_.py  │ ├── file\_handler.py  │ ├── analysis.py  ├── data/  │ ├── exercises.json  │ ├── plans.json  │ ├── sessions.json  ├── tests/  │ ├── \_\_init\_\_.py  │ ├── test\_workout.py |  | ☐ |  |  |
|  |  | struktura modułów |  |  |  |  | 2 |
|  |  |  |  | SUMA |  | |  |

Strona **3** z **3**