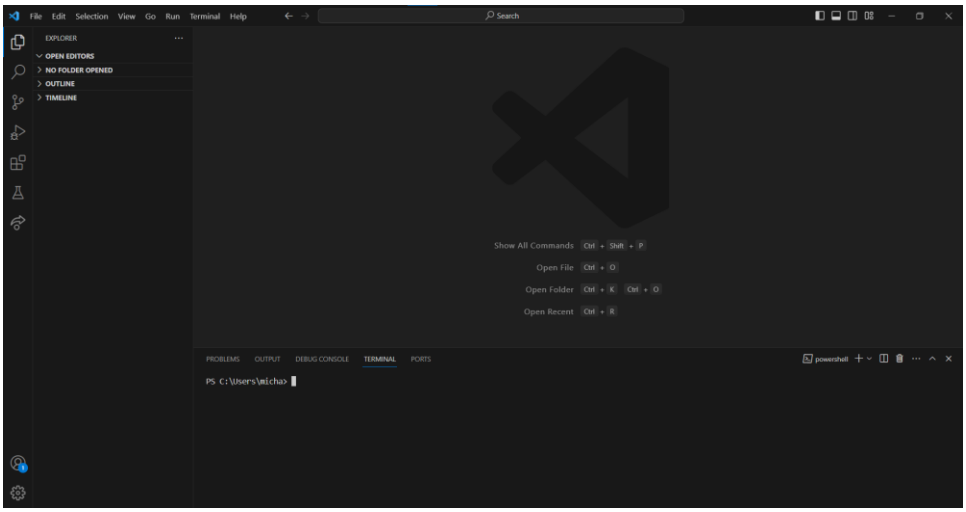
Sprawozdanie z laboratorium komputerowego

Zad 1

Celem zadania było napisać program, który numerycznie znajduje maksima i minima oraz miejsca zerowe danej funkcji.

Pisanie programu zaczynamy od otwarcia wybranego edytora kodu. W tym przypadku uzyto programu Visual Studio Code.



Następnie przystępujemy do pisania kodu. Przywołania wymaganej biblioteki oraz zdefiniowania wymaganych list i funkcji.

```
1   import math
2
3   #defining arrays which have from top:
4   #all the values possible
5   #all the values of x which result in f(x) = 0 (approx)
6   lista = []
7   listazer = []
8
9
10  #defining mathematical function
11  def f(x):
12      return (x**4)+5*(x**3)-20*(x**2)+1
13
```

Na zielono pokazane są komentarze, które zostały użyte do opisu funkcjonalności kodu, tak aby jego czytelność została zwiększona. Następnie przechodzimy do napisania funkcji, która wyznaczy nam miejsca zerowe jak i minima i maksima.

```
14  #function should only be feed integers
15  def oblicz(poczatek, koniec):
```

Funkcja ta przyjmuje początek i koniec przedziału, obydwie te wartości zostaną użyte do zdefiniowania pętli, dlatego też, wartości te muszą przyjmować tylko typ int. Warto też zauważyć, że początek nie powinien być większą liczbąod końca, dlatego wprowadzamy następujące polecenie:

```
if poczatek > koniec:
    print("koniec większy od początku")
    return
```

W ten sposób wychodzimy z funkcji i informujemy użytkownika, jeżeli koniec był większy od początku.

Następnie obliczamy miejsca zerowe funkcji. W tym przypadku miejsca zerowe będą przybliżeniami. W celu obliczenia miejsc zerowych, wystarczy podstawić wartości x do funkcji w naszym przedziale liczb, a następnie sprawdzić, który x jest równy (w przybliżeniu) 0. Osiągnięto to w następujący sposób:

```
30      while koniec > i:
31          i += 0.1
32          lista.append(f(i))
33          #print(lista[-1]) #easy debugging for array
34          #znajdź miejsca zerowe funkcji (przybliżone)
35          if int(lista[a]) == 0:
36              #b += 1
37              listazer.append(round(i, 2))
38              listatemp.append(listazer[b])
39              b += 1
40          a += 1
41      print("Liczby dla których funkcja przyjmuje wartość zerową:\n",listatemp)
42
```

Gdzie metoda .append jest używana do umieszczenia liczb x dla których funkcja przyjmuje wartość 0.

Następnie lista z tymi liczbami x jest wyświetlana do użytkownika.

Po wykonaniu tego kroku, można przejść do obliczenia min i max w przedziale. Powinno być to stosunkowo proste, tablica ze wszystkimi wartościami została już utworzona, aby sprawdzić, gdzie wystąpiło miejsce zerowe, można więc użyć tej listy do sprawdzenia wartości max i min, problem występuje jednak w tym, że funkcja na krańcach przedziału zazwyczaj będzie posiadała najwyższą wartość i/lub najniższą. W takim razie musimy jeszcze sprawdzić, czy wartości pomiędzy sprawdzaną wartością rosną lub maleją. Jeżeli obydwie wartości obok maleją to oznacza to, że trafiliśmy na max, jeżeli obydwie wartości rosną to znaczy, że trafiliśmy na min. W ten sposób nie musimy martwić się już o wartości znajdujące się na końcu przedziału. W kodzie wygląda to następująco:

```
45      #check for the local maximum and minimum
46      for h in range(len(lista) - 1):
47          if h >= 1 and h != len(lista):
48              if lista[h] > lista[h-1] and lista[h] > lista[h+1]:
49                  maxl = lista[h]
50                  print("Maximum lokalne:", maxl)
51              elif lista[h] < lista[h-1] and lista[h] < lista[h+1]:
52                  minl = lista[h]
53                  print("Minimum lokalne:",minl)
```

Warto jednak zauważyć, że w danym przedziale, nie zawsze musi występować minimum i maksimum lokalne, w takim razie wystarczy sprawdzić, czy obliczyliśmy jedno z nich i jeżeli tego nie zrobiliśmy poinformować o tym użytkownika:

```
54      #if min or max not found, print to user
55      if maxl == None:
56          print("brak maximum lokalnego")
57      elif minl == None:
58          print("brak minimum lokalnego")
59      return
```

Return w teorii nie jest potrzebne w tej funkcji, ponieważ funkcja nie zwraca żadnych wartości, ale warto to zrobić dla czystości kodu oraz dla uniknięcia problemów z pamięcią komputera.

Na końcu wystarczy tylko przyjąć wartości podane przez użytkownika, następnie sprawdzić czy wartości te są prawidłowe. Można to zrobić za pomocą polecenia try;except :

```
62  print("Podaj początek przedziału liczb dla którego chcesz wyznaczyc min i max oraz miejsca zerowe")
63  x = input()
64  print("Podaj koniec przedziału")
65  y = input()
66  try:
67      oblicz(int(x), int(y))
68  except:
69      print("Podano nieprawidłowe dane")
70
```

Zawartość konsoli wygląda następująco:

```
Podaj początek przedziału liczb dla którego chcesz wyznaczyc min i max oraz miejsca zerowe
-10
Podaj koniec przedziału
10
Liczby dla których funkcja przyjmuje wartość zerową:
 [-0.3, -0.2, -0.1, 0.1, 0.2, 0.3, 2.6]
Minimum lokalne: -520.8303999999998
Maximum lokalne: 1.0
Minimum lokalne: -24.142399999999995
```

Zad 2

Zadanie drugie polegało na sprawdzeniu czy dla podanego słowa podany został anagram. Funkcja powinna zwrócić True jeżeli dla danego słowa podany anagram jest rzeczywiście anagramem i False dla sytuacji odwrotnej.

Na początku trzeba zacząć od przyjęcia danych od użytkownika i zdefiniowaniu funkcji:

```
1  #user input, no need to check if user gives good data types as input() function is always == str
2  print("wprowadz slowo")
3  slowo = input()
4  print("wprowadz anagram slowa")
5  anagram = input()
6
7
8  def sprawdz(slowo, anagram):
```

Warto zauważyć, że jeżeli ilość znaków w słowie jest nie taka sama jak w anagramie, to wiadomo, że podany anagram nie jest anagramem.

```
    if len(slowo) != len(anagram):
        return False
```

Jeżeli jednak ilość znaków jest prawidłowa, to przechodzimy dalej. W przypadku tego programu, najłatwiejszym rozwiązaniem, jest po prostu sprawdzenie dla każdej pojedynczej litery w podanym słowie, czy istnieje ta literka w anagramie. Jednak, w słowach czasem mogą występować aż po dwie lub trzy takie same litery. Aby uniknąć problemu, gdzie powtarzające się znaki mogą zmylić nasz program, po znalezieniu, że dana sprawdzana litera, jest taka sama jak w anagramie, usuwamy tą sprawdzoną literę z anagramu. Warto zauważyć, że usuwanie liter musi być osiągnięte za pomocą stringu bez spacji tzn. “” a nie stringiem ze spacją “ “ ponieważ spacja to też znak. Pomimo tak wielu uwag taka funkcjonalność programu zajmuje tylko kilka linijek kodu:

```
18      for x in slowo:
19          for h in anagram:
20              if x == h:
21                  liczba_prawd += 1
22                  temp = anagram.replace(x, "", 1)
23                  anagram = temp
24                  break
```

Następnie wystarczy tylko sprawdzić czy ilość prawd jest taka sama jak ilość znaków, wtedy wiemy, że występują wszystkie takie same litery.

```
26      if liczba_prawd == len(slowo):
27          return True
28      else:
29          return False
```

Na koniec uruchamiamy funkcje

```
31  print(sprawdz(slowo, anagram))
```

Zawartość konsoli wygląda następująco:

```
wprowadz slowo
dlugi wyraz z wieloma pauzami-i-znakami.
wprowadz anagram slowa
ldugi wyraz-z-wieloma pauzami i imakanz.
True
```

Co potwierdza działanie programu.

Zad 3

Zadanie trzecie polegało na napisaniu dwóch funkcji, jedna z których ma sprawdzić czy w danej tabeli sa przynajmniej dwa takie same elementy, i zwrócić True jeżeli występują dwa elementy, druga funkcja tymczasem ma za zadanie obliczyć przybliżone prawdopodobieństwo, że w klasie 17 osobowej, przynajmniej dwie osoby mają tą samą datę urodzin.

Warto zacząć od wprowadzenia biblioteki random, która zostanie później użyta do drugiego podpunktu zadania:

```
1   import random as rn
```

Możemy również wprowadzić tabele testową za pomocą, której sprawdzimy, czy funkcja działa.

```
2   #debug array to check if the first function that checks for the same elements in an array works
3   debugarray = [1, 2, 3, 4, 5, 10, 6, 7, 8, 9]
```

Następnie tworzymy samą funkcje, która działa na podobnej zasadzie jak ta z zadania drugiego:

```
6   #function to check if there are at least two of the same element in an array
7   def sprawdz(lista):
8       for x in lista:
9           for h in range(len(lista)):
10              if lista.index(x) != h:
11                  if x == lista[h]:
12                      return True
```

Teraz aby obliczyc prawdopodobieństwo w drugiej funkcji, musimy najpierw utworzyć tabele zawierającą losowe dane, które są typem int i mieszczą się w przedziale od 1 do 365.

```
16  def paradoks(ilosc_uczniow, dokladnosc):
17      iloscok = 0
18      for h in range(dokladnosc):
19          templista = []
20          #create the class of people
21          for i in range(ilosc_uczniow):
22              data = rn.randint(1, 365)
23              templista.append(data)
```

Następnie wykorzystujemy wcześniej napisaną funkcję aby sprawdzić, czy dana tabela zawiera dwie te same daty:

```
25              if sprawdz(templista):
26                  iloscok += 1
```

Teraz aby obliczyc prawdopodibeństwo musimy tylko podzielić ilość wszystkich klas spełniających ten warunek, przez ilość wszystkich utworzonych klas.

```
27      #return probability
28      return (iloscok/dokladnosc)
```

Następnie za pomocą polecenia print drukujemy do konsoli, aby sprawdzić działanie pierwszej funkcji i wynik drugiej.

```
30  print("Sprawdzamy czy funkcja prawidlowo działa:", sprawdz(debugarray))
31  print("umieść dokładność z jaką chcesz obliczyc prawdopodobieństwo, że dwie osoby mają taką samą date urodzin")
32  g = input()
33  print("Prawdopodobieństwo wynosi:",paradoks(17, int(g)))
```

W konsoli otrzymujemy:

```
Sprawdzamy czy funkcja prawidlowo działa: False
umieść dokładność z jaką chcesz obliczyc prawdopodobieństwo, że dwie osoby mają taką samą date urodzin
1000
Prawdopodobieństwo wynosi: 0.324
```

Co kończy zadanie.