Języki Skryptowe

dokumentacja projektu Kalendarz Rzymski

Marcin Bogus, grupa 4/7

Politechnika Śląska, Wydział Matematyki Stosowanej, Kierunek Informatyka

22 stycznia 2023

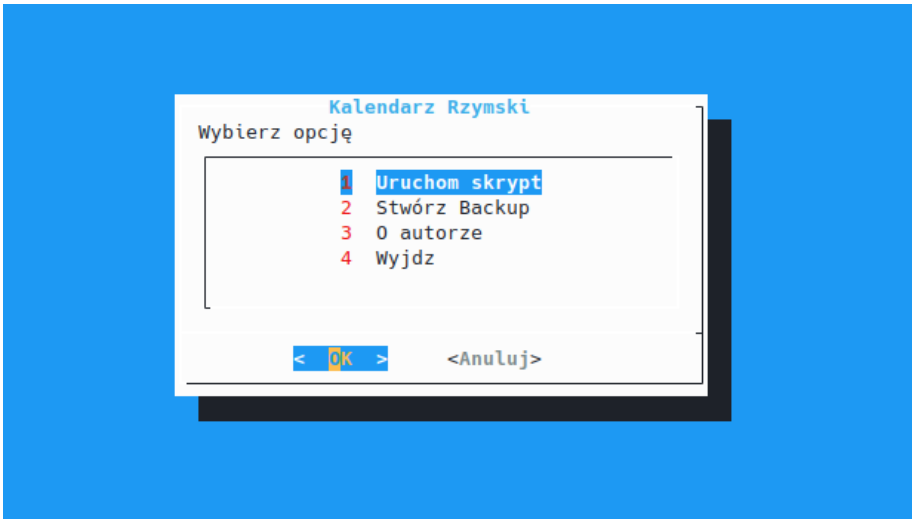
Część I

Opis programu

Cyfry. Należy napisać program, który zamienia liczby zapisane za pomocą cyfr arabskich na odpowiadające im liczby zapisane za pomocą cyfr rzymskich (i odwrotnie). Program ma sam rozpoznawać, w którą stronę ma nastąpić zamiana, a dodatkowo musi rozpoznawać poprawność zapisu liczb za pomocą cyfr rzymskich. Np. podając a) 1793 program zwraca MDCCXCIII; b) MCMXLVIII program zwraca 1948; c) IM program zwraca komunikat błędny zapis (a nie 999).

Instrukcja obsługi

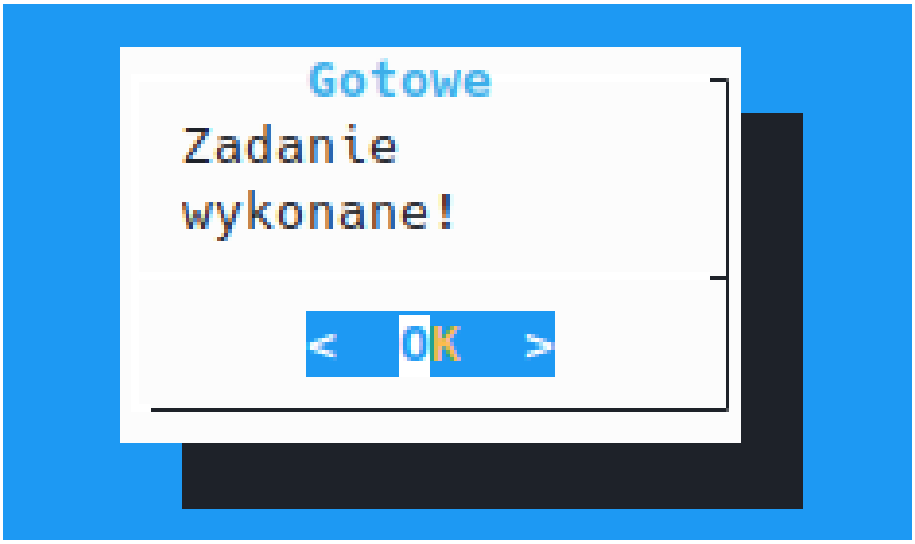
Aby uruchomić program należy włączyć skrypt run.sh otwierający menu obsługi naszego programu. Po uruchomieniu wyświetli się nam menu konsolowe które pokaże nam opcje które możemy wybrać w programie



Rysunek 1: Główne menu programu

Możliwe wybory są następujące:

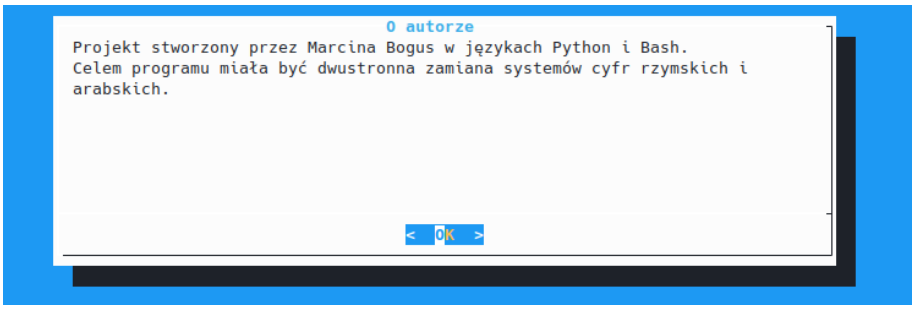
1. Uruchom skrypt - Uruchamia program pobierając przy tym wszystkie dane z katalogu input i tworzy nam plik w formacie rrrr-mm-dd-hh-mm-ss.html, który można zobaczyć w dowolnej przeglądarce



Rysunek 2: Przykładowy komunikat o pomyślnej próbie uruchomienia programu

W przypadku braku wymaganych plików do uruchomienia programu, bądź błędnych plików input, program wyświetli odpowiedni komunikat z opisem problemu

2. Stwórz backup - tworzy kopię zapasową w folderze "backup/data-backupu/"
3. O autorze - Wypisuje dane o autorze i celach programu



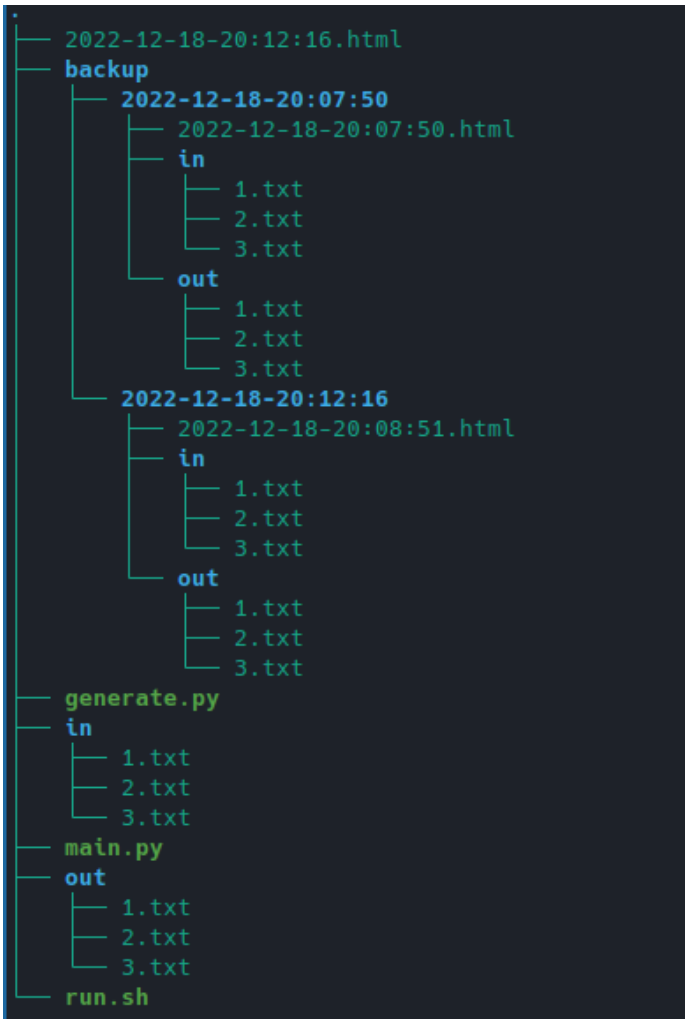
Rysunek 3: Informacje o programie

4. Wyjdź - Zamyka menu, kończąc tym samym program.

Struktura danych programu

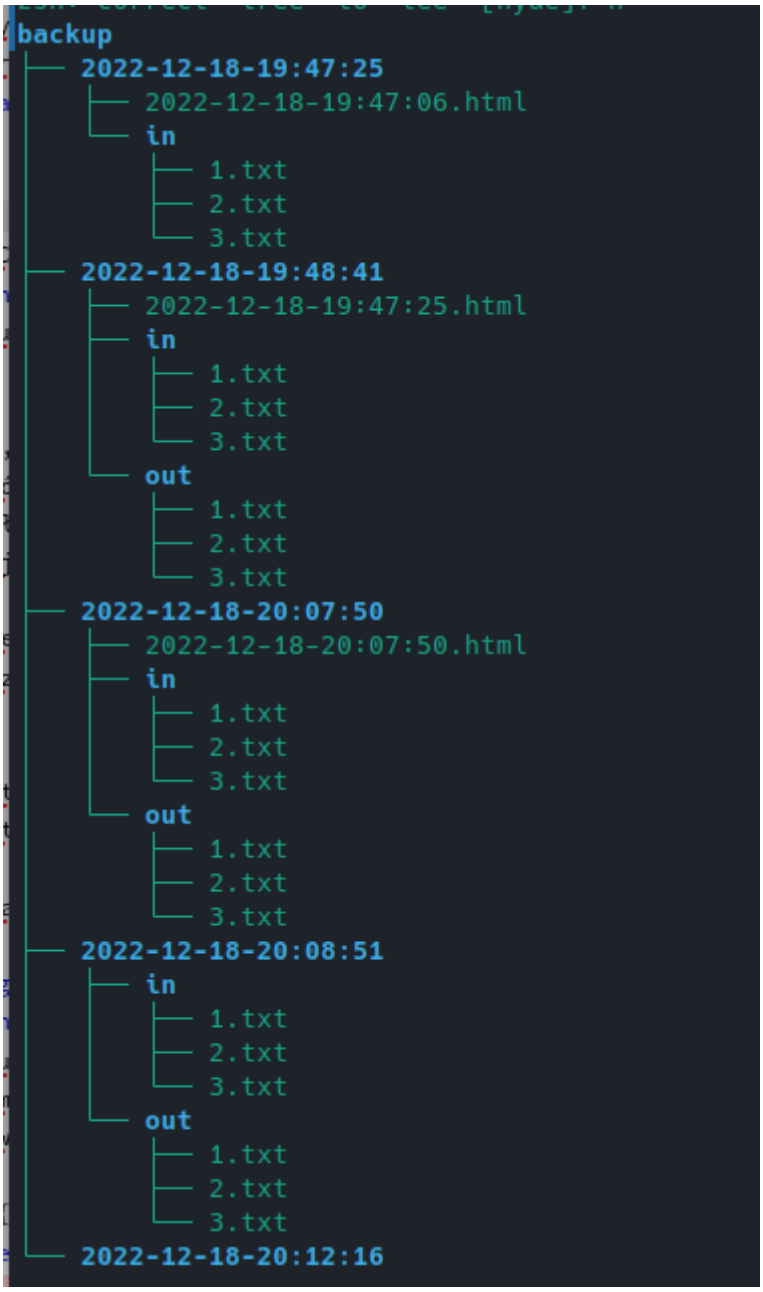
Program składa się z następującej struktury danych, wymaganych do prawidłowego uruchomienia aplikacji:

- run.sh - Skrypt bash będący menu, którym uruchamia się program, wyświetla informacje o programie jak i tworzy kopie zapasową danych otrzymanych w wyniku wykonania tegoż programu
- generate.py - Skrypt python odpowiedzialny za pobranie informacji o plikach w folderze "in"i "out"a następnie przemienienie go w plik raportowy
- main.py - Skrypt python z algorytmem, którego dane są podawane jako argument wejściowy programu
- Katalog in zawierający pliki z liczbami do konwersji:
    - *.txt - Plik z pojedyńczą cyfrą dowolnie rzymską lub arabską
- Katalog out zawierający pliki z liczbami po konwersji:
    - *.txt - Plik z pojedyńczą cyfrą dowolnie rzymską lub arabską



Rysunek 4: Struktura danych programu w formie drzwa

- Katalog "backups"z kopiami raportów i danych wejściowych/wyjściowych w folderach nazywanych datami



Rysunek 5: Przykładowa struktura zapisu backupu

Część II

Opis działania

Skrypt run.sh posiada menu które dokona działania wraz z wyborem użytkownika:

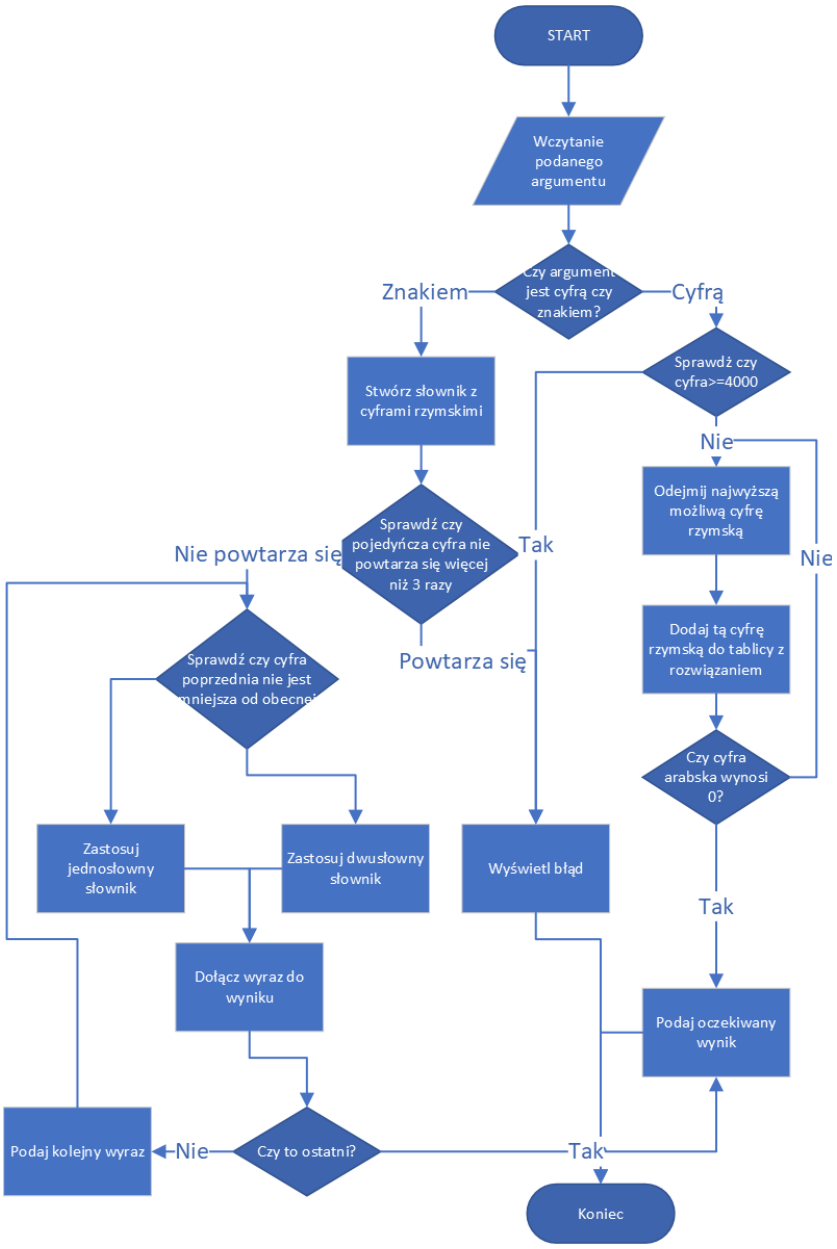
Jeśli zostanie wybrana opcja "Uruchom skrypt", skrypt zacznie zliczać pliki w folderze "in"i każdy z tych plików zostanie wczytany z osobna a następnie za pomocą skryptu main.py w postaci argumentu wejściowego zostanie wprowadzony argument, który następnie skrypt main.py zwróci w postaci wyniku wyjściowego, który zostanie zapisany w folderze out o tej samej nazwie co plik wejściowy. Następny proces to włączenie skryptu generate.py, który na podstawie podanej daty w argumencie oraz plików w folderach "in"i "out"sporządzi raport w pliku rrrr-mm-dd-hh-mm-ss.html. Po zakończeniu procesu zwróci w okienku wiadomość o ukończeniu zadania.

Jeśli została wybrana opcja "Stwórz backup"skrypt run.sh pobierze aktualną datę i stworzy w folderze backup folder rrrr-mm-dd-hh-mm-ss i wrzuci do niego plik raportowy wraz folderami "in"i "out". Zakończony proces zostanie oznajmiony okienkiem z zakończonego zadania.

W opcji "O autorze"dostaniemy monit z informacją, kto stworzył projekt i jaki był jego cel.

W opcji "Wyjdź"lub klikniemy anuluj, skrypt zakończy swoje działanie.

Algorytm



Rysunek 6: Algorytm w postaci schematu blokowego

Implementacja systemu

Wykorzystałem do napisania skryptów:
-Pythona w wersji 3.11 oraz dodatkowe biblioteki z PiP
-Bash w wersji 5.2 z dodatkowymi aplikacjami

Skrypt run.sh wykorzystuje menu będące komendą Dialog, której domyślnie nie ma w systemie Linux. Owe menu jest zapętlone w pętli while, aż do momentu wybrania opcji 4 lub kliknięcia anuluj.

- dialog

```
while choice=$(dialog --title "Kalendarz Rzymski" --menu "Wybierz opcj " 12 45 25 1 "Uruchom skrypt" 2 "Stw rz Backup" 3 "O autorze" 4 "Wyjdz" 2>&1 >/dev/tty)
```

Skrypt generate.py wykorzystuje bibliotekę tabulate, która jest wyposażona w opcję eksportowania tabelki w html. Dodatkowo do tego pliku dodawana jest informacja o dacie wykonania raportu, która dla zgodności czasu jest brana z argumentu wejściowego podawana przez skrypt run.sh

- tabulate

```
p = tabulate(i,headers, tablefmt='html')
o = open(sys.argv[1],"w",encoding="utf-8")
o.write(p)
o.close()
```

Wykorzystane biblioteki i przykłady ich użycia

- os

```
os.chdir("out")
# zmieniamy katalog w ktorym pracujemy
for letter in os.listdir():
# robimy liste plikow zeby moc zebrac z nich dane
```

- tabulate

```
p = tabulate(i,headers, tablefmt='html')
# tworzymy finalowa tabelke, ktora wyeksportujemy do html
```

- numpy

```
i=numpy.transpose(table)
# Transponujemy zagniezdzona liste
```

- regex

```python
import re
regex = r"^M{0,3}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I{0,3})$"
def validate_roman(string: str):
    result = re.match(regex, string)
    return result is not None
#Potrzebne do walidacji kolejnosci cyfr rzymskich
```

Testy

Dane wejściowe:

1.txt = 25
2.txt = 12
3.txt = XXII

Dane wyjściowe:

1.txt = XXV
2.txt = XII
3.txt = 22

> # Raport utworzono o 2022-12-18-23:14:13
>
> | Input | Output |
> |---|---|
> | 25 | XXV |
> | 12 | XII |
> | XXII | 22 |

Rysunek 7: Utworzony raport na podstawie danych

Pełen kod aplikacji

main.py

```
import sys
import re
regex = r"^M{0,3}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I{0,3})$"
def validate_roman(string: str):
    result = re.match(regex, string)
    return result is not None

def convert(y):
    if y.isnumeric(): # arabska
        x = int(y)
        res = []
        if x>=4000:
            print("Za duza liczba")
        else:
            while x != 0:
                if x >= 1000:
                    x -= 1000
                    res.append("M")
                elif x >= 900:
                    x -= 900
                    res.append("CM")
                elif x >= 500:
                    x -= 500
                    res.append("D")
                elif x >= 400:
                    x -= 400
                    res.append("CD")
                elif x >= 100:
                    x -= 100
                    res.append("C")
                elif x >= 90:
                    x -= 90
                    res.append("XC")
                elif x >= 50:
                    x -= 50
                    res.append("L")
                elif x >= 40:
                    x -= 40
                    res.append("XL")
                elif x >= 10:
                    x -= 10
                    res.append("X")
                elif x >= 9:
                    x -= 9
                    res.append("IX")
                elif x >= 5:
                    x -= 5
                    res.append("V")
                elif x == 4:
                    x -= 4
```

```
                    res.append("IV")
                else:
                    x -= 1
                    res.append("I")
            res = "".join(res)
            print(res)
    else:   #rzymska
      if not validate_roman(y):
          raise IOError("Zla cyfra rzymska")
      translations = {
          "I": 1,
          "V": 5,
          "X": 10,
          "L": 50,
          "C": 100,
          "D": 500,
          "M": 1000,
          }
      first_value = translations[y[0]]
      if len(y) == 1:
          print(first_value)
      result = first_value if translations[y[1]] <= first_value else -
         first_value
      for idx, letter in [x for x in enumerate(y)][2:]:
          previous = y[idx-1]
          if translations[letter] > translations[previous]:
                result -= translations[previous]
            #  assuming it was added once already
          else:
                result += translations[previous]
      result += translations[y[-1]]
      print(result)
if __name__ == "__main__":
    convert(sys.argv[1])
```

generate.py

```
import os
import sys
from tabulate import tabulate
import numpy
os.chdir("out")
y =[]
z =[]
for letter in os.listdir():
    x= open(letter,'r',encoding="utf-8").read()
    y.append(x)
os.chdir("..")
os.chdir("in")
for letter in os.listdir():
    x= open(letter,'r').read()
    z.append(x)
y.reverse()
```

```
z.reverse()
table =z,y
i=numpy.transpose(table)
headers =("Input","Output")
os.chdir("..")
p = tabulate(i,headers, tablefmt='html')
o = open(sys.argv[1],"w",encoding="utf-8")
header = "<h1>Raport wygenerowano o: "+sys.argv[1].strip(".html")+"</h1>
    "
o.write(header)
o.write(p)
o.close()
```

---

run.sh

```
#!/bin/bash
date=$(date "+%Y-%m-%d-%H:%M:%S")
while choice=$(dialog --title "Kalendarz Rzymski" --menu "Wybierz opcj
    " 12 45 25 1 "Uruchom skrypt" 2 "Stw rz Backup" 3 "O autorze" 4 "
    Wyjdz" 2>&1 >/dev/tty)
  do
  case $choice in
      1) cd in
          count=$(ls -1q *|wc -l)
          cd ..
          rm *.html
          touch "$date.html"
          cd in
          for ((i=1;i<=$count;i++)) ; do
              :
              input=$(cat $i.txt)
              cd ..
              result=$(python3 main.py $input)
              cd out
              echo $result > "$i.txt"
              cd ..
              cd in
          done
          cd ..
          python3 generate.py "$date.html"
          dialog --title "Gotowe" --msgbox "Zadanie wykonane!" 6 20;;
      2) mkdir backup/$date
          mv *.html "backup/$date/"
          cp -R in backup/$date/
          cp -R out backup/$date/
          dialog --title "Gotowe" --msgbox "Zadanie wykonane!" 6 20;;
      3)  dialog --title "O autorze" --msgbox "Projekt stworzony przez
         Marcina Bogus w j zykach Python i Bash.
Celem programu mia a by  dwustronna zamiana system w cyfr rzymskich i
    arabskich." 12 80;;
        *) exit;;
  esac
done
```