

MICRO:BIT – KARTKA Z KALENDARZA

Jolanta Sobera, Anna Szczërba-Zubek
Wydział Nauk Ścisłych i Technicznych
40-007 Katowice, Bankowa 14

jolanta.sobera@us.edu.pl, anna.szczërba-zubek@us.edu.pl

Abstract. *In the article, we present an example of programming classes in Python, where the main goal is to generate a string containing information about a specific date, the day of the week, and events correlated with the day provided as the day number of the year. The Zeller's algorithm was used.*

1. Wstęp

W poniższym artykule opisano przykładowy cykl ćwiczeń z języka Python, których wynikiem jest program na płytkę micro:bit. Zajęcia są przeznaczone dla uczniów szkół ponadpodstawowych. Wymagana jest znajomość list, słowników i funkcji. Celem jest przygotowanie przesuwającego się napisu „DZIŚ JEST PONIEDZIAŁEK 73 DZIEŃ ROKU. DO KONCA ROKU POZOSTAŁO 292 DNI. DZIEŃ LICZBY PI.” Podczas pracy wykorzystano IDLE do tworzenia i testowania kodów funkcji oraz stronę micro:bit Python Editor. Wykorzystano algorytmy: do sprawdzenia czy rok jest przestępny, Zellera do wyznaczenia nazwy dnia tygodnia.

2. Rok przestępny

Przyjmuje się, że twórcami pierwszego kalendarza byli Babilończycy. Bazował on na obserwacji księżyca, a zatem składał się z miesięcy liczących 28 dni. Babilończycy zauważyli pewne rozbieżności pomiędzy poszczególnymi miesiącami a porami roku i wydłużyli miesiące do 29 dni, a następnie wszystkie nieparzyste do 30 dni. Nie tylko Babilończycy opracowali kalendarz, mieli go też Chińczycy, czy Egipcjanie oraz Rzymianie. Lata przestępne zostały po raz pierwszy wprowadzone w 238 roku p.n.e. w Egipcie. Zaczęto tam uwzględniać dodatkowy dzień co cztery lata.

Z kolei rzymski rok początkowo liczył 10 miesięcy i rozpoczynał się w marcu. W czasie rządów Numa Pompiliusza do kalendarza dodano kolejne 2 miesiące – Februarius i Ianuarius. W 45 roku p.n.e. Juliusz Cezar wprowadził kolejny system podziału czasu, nazwany na jego cześć kalendarzem juliańskim. Ten system dzielił rok na 365 dni i co 4 lata wprowadzał jeden dodatkowy dzień. Dodatkowy dzień został dodany do

najkrótszego miesiąca, lutego, który w tym czasie był też ostatnim miesiącem roku. Błąd tej rachuby wynosi ok. 1 dobę na 128 lat.

Obecnie powszechnie stosuje się rachubę zgodną z kalendarzem gregoriańskim, wprowadzonym w 1582 roku bullą papieża Grzegorza XIII, w której wprowadzono modyfikację kalendarza juliańskiego mówiącą, że: nie uznaje się lat przestępnych wypadających na koniec wieku, z wyjątkiem tych, w których liczba stuleci jest podzielna przez 4. W myśl tej reguły latami przestępnymi są te, których numeracja:

- jest podzielna przez 4 i niepodzielna przez 100 lub
- jest podzielna przez 400.

Według tej reguły lata 1600 i 2000 były przestępnymi, a lata 1700, 1800, 1900 nie. Modyfikacja kasuje 15 lat przestępnych na każde 2000 lat co zmniejsza błąd kalendarza juliańskiego o 1 dobę na 133 lata, dlatego błąd tej rachuby wynosi 1 dobę na nieco ponad 3322 lata.

```
def czy_przestepny(rok: int) -> int:
    """
    Sprawdza, czy dany rok jest przestępny.

    Args:
        rok (int): Rok do sprawdzenia.

    Returns:
        int: 366 jeśli rok przestępny, 365 w przeciwnym wypadku.

    Raises:
        TypeError: Jeśli argument nie jest typu int.
        ValueError: Jeśli rok jest mniejszy niż 1582.

    Examples:
        >>> czy_przestepny(2000)
        366
        >>> czy_przestepny(1900)
        365
        >>> czy_przestepny(1581)
        Traceback (most recent call last):
          ...
        ValueError: Rok musi być nie mniejszy niż 1582.
    """
    if not isinstance(rok, int):
        raise TypeError("Argument musi być liczbą całkowitą (int).")
    if rok < 1582:
        raise ValueError("Rok musi być nie mniejszy niż 1582.")
    if (rok % 4 == 0 and rok % 100 != 0) or (rok % 400 == 0):
        return 366
    return 365

import doctest
doctest.testmod(verbose = True)
```

Rysunek 1. Funkcja sprawdzająca czy rok jest przestępny

Pierwsze zadanie, które wykonujemy, to przygotowanie funkcji sprawdzającej czy dany rok jest rokiem przestępnym (Rysunek1). Wynikiem jest liczba dni w roku.

3. Dzień miesiąca i nazwa miesiąca

Kolejnym etapem jest dodanie dnia miesiąca i nazwy miesiąca odpowiadającemu zadanemu numerowi dnia w roku. Napišemy funkcję *data*, której argumentami będą numer dnia w roku (1–365/366) i liczba dni w roku (365 lub 366). Funkcja ta będzie zwracała krotkę zawierającą: numer dnia w miesiącu, numer miesiąca (1–12) oraz nazwę miesiąca słownie. Aby rozwiązać problem przygotujemy dwie listy *ile_dni_miesiace* i *miesiace*, które zawierają odpowiednio liczbę dni w poszczególnych miesiącach i nazwy miesięcy. Lista *ile_dni_miesiace* jest aktualizowana w przypadku gdy rok jest przestępny (linie 17- 22 Rysunek 2. Wykorzystamy pętlę *while*. Zmienną *numer*, zmniejszamy o kolejną pobraną wartość z listy *ile_dni_miesiace*, tak długo aż będzie mniejsza od kolejnej wartości na liście. Kolejną wartość z listy będziemy pobierać korzystając z indeksu *miesiac_index* ustawionego przed pętlą na 0. Licznik ten zwiększa się o 1 przy każdym obiegu pętli. Przykład dla numeru dnia 100 jest opracowany w tabeli Tabela 1, a kod funkcji, z notką dokumentacyjną, zaprezentowany na Rysunku 2. Obydwie funkcje kopiujemy do pliku *funkcje.py* stworzonego w środowisku micro:bit w zakładce Project za pomocą przycisku Create File (Rysunek 3).

<i>miesiac_index</i>	<i>numer</i>	Nazwa miesiąca pobrana z tabeli <i>miesiace</i>
0	$100 - 31 = 69$	styczen
1	$69 - 28 = 41$	luty
2	$41 - 31 = 10$	marzec
3	$10 - 30 < 0$	kwiecień

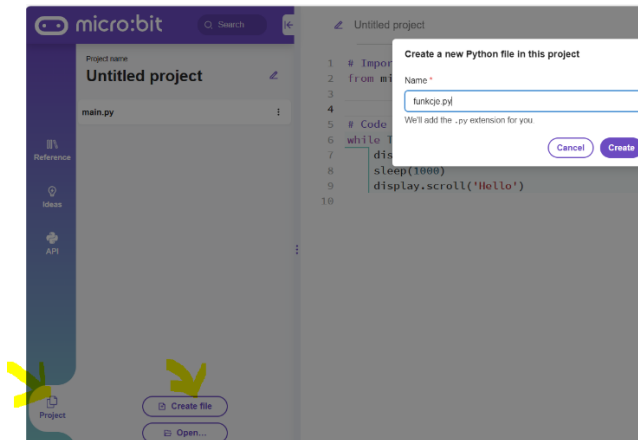
Tabela 1. Przykład obliczeń dla numeru dnia 100

```

1 def data(numer: int, ile_dni: int = 365):
2     """
3     Funkcja wyznacza numer dnia oraz odpowiadający mu miesiąc (cyfrą i słownie).
4
5     """
6     if not isinstance(numer, int) or not isinstance(ile_dni, int):
7         raise TypeError("Błąd typu: oba argumenty muszą być typu int.")
8     if numer <= 0 or numer > ile_dni or ile_dni not in {365, 366}:
9         raise ValueError("Niepoprawny argument.")
10
11     miesiace = [
12         "styczen", "luty", "marzec",
13         "kwiecień", "maj", "czerwiec",
14         "lipiec", "sierpień", "wrzesień",
15         "październik", "listopad", "grudzień"
16     ]
17     dni_w_miesiacu = [
18         31, 28, 31, 30, 31, 30,
19         31, 31, 30, 31, 30, 31
20     ]
21     if ile_dni == 366:
22         dni_w_miesiacu[1] = 29
23
24     miesiac_index = 0
25     while numer > dni_w_miesiacu[miesiac_index]:
26         numer -= dni_w_miesiacu[miesiac_index]
27         miesiac_index += 1
28
29     return numer, miesiac_index + 1, miesiace[miesiac_index]

```

Rysunek 2. Funkcja wyznaczająca numer dnia i nazwę miesiąca

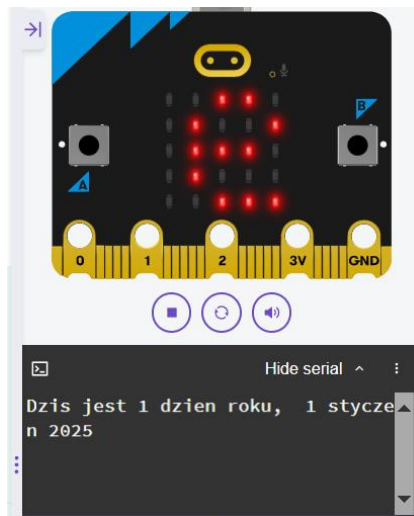


Rysunek 3. Środowisko micro:bit. Tworzenie pliku z funkcjami.

W oknie głównym można napisać pierwszy test w środowisku micro:bit. Uzupełniona została nazwa projektu *Kalendarz*. W linii 2 importujemy funkcje *czy_przestępny* i *data*. W kodzie formatujemy napis, wywołujemy zaimportowane funkcje. Aby przyspieszyć można użyć funkcji: *print* i *sleep* (linie 14 i 15 na Rysunku 4), a dla cierpliwych jest *display.scroll* (linia 16 na Rysunku 4). Wynik widzimy na Rysunku 5.

```
1 from microbit import *
2 from funkcje import czy_przestepny, data, zeller
3
4 nr_dnia_w_roku = 1
5 rok = 2025
6
7 while True:
8     napis = "Dzis jest " + str(nr_dnia_w_roku) + " dzien roku, "
9     ile_dni = czy_przestepny(rok) # 365 lub 366
10    numer_dnia, nr_miesiaca, nazwa_miesiaca = data(nr_dnia_w_roku, ile_dni)
11    napis += " " + str(numer_dnia)
12    napis += " " + nazwa_miesiaca
13    napis += " " + str(rok)
14    print(napis)
15    sleep(1000)
16    display.scroll(napis)
```

Rysunek 4. Pierwszy test



Rysunek 5. Wynik pierwszego testu.

4. Nazwa dnia tygodnia

Kolejne zadanie to wyznaczenie nazwy dnia tygodnia na podstawie numeru dnia w roku. Przedstawimy dwa rozwiązania. Pierwsze bardzo proste, ale dobre tylko dla danego roku, drugie to algorytm niemieckiego matematyka Christiana Zeller (1822-1899). Został on opracowany odpowiednio dla kalendarza juliańskiego

i gregoriańskiego. Prezentujemy tylko wersję dla kalendarza gregoriańskiego. Rozwiązania przedstawimy w postaci funkcji.

4.1. Wersja podstawowa

Przygotujemy funkcję, której argumentami będą analogicznie jak poprzednio: *numer_dnia* w roku (1–365/366) i liczba dni w roku (365 lub 366). Funkcja zawiera listę *dni_tygodnia*, zawierającą kolejne dni tygodnia. Dni tygodnia jest 7, zatem wyznaczając resztę z dzielenia *numeru_dnia* przez 7 otrzymamy liczbę całkowitą z przedziału [0, 6].

```

1 def dzien_tygodnia(numer: int, ile_dni: int = 365) -> str:
2     """
3     Funkcja wyznacza dzień tygodnia dla danego numeru dnia w roku.
4
5     Args:
6         numer (int): Numer dnia w roku (1-365/366).
7         ile_dni (int): Liczba dni w roku (365 lub 366). Domyślnie 365.
8
9     Returns:
10        str: Nazwa dnia tygodnia.
11
12     Raises:
13        TypeError: Jeśli argument nie jest typu int.
14        ValueError: Jeśli numer dnia jest poza zakresem.
15
16     Examples:
17        >>> dzien_tygodnia(32)
18        'sobota'
19        >>> dzien_tygodnia(387)
20        Traceback (most recent call last):
21        ...
22        ValueError: Liczba spoza zakresu.
23    """
24    dni_tygodnia = [
25        "wtorek",
26        "sroda",
27        "czwartek",
28        "piatek",
29        "sobota",
30        "niedziela",
31        "poniedzialek"
32    ]
33    if not isinstance(numer, int):
34        raise TypeError("Niepoprawny typ.")
35    if numer <= 0 or numer > ile_dni:
36        raise ValueError("Liczba spoza zakresu.")
37
38    return dni_tygodnia[(numer) % 7]

```

Rysunek 6. Funkcja wyznaczająca dzień tygodnia.

Liczba ta będzie indeksem w tablicy, umożliwiającym odczytanie nazwy dnia tygodnia figurującej pod tym indeksem. Z tego powodu drugi element na liście, ten o indeksie 1, to nazwa dnia tygodnia, która przypada 1 stycznia roku dla którego przygotowujemy kalendarz. W roku 2025 jest to środa. Wszystkie dni w roku 2025, dla których w wyniku wyznaczenia reszty z dzielenia numeru dnia przez 7

- otrzymujemy 1, to środy,

- otrzymujemy 2 to czwartki,
- reszta równa 3 koreluje z piątkiem itd.

Resztę 0, otrzymamy dla siódmego dnia w roku, zatem na pierwszym miejscu na liście musi być wpisany wtorek. Pełną funkcję z obsługą błędów oraz docstringiem i testem przedstawia Rysunek 6.

4.2. Algorytm Zellera

Algorytm pozwala obliczyć dzień tygodnia dla dowolnej daty po roku 1582. Argumentami przesłanymi do funkcji będą: numer dnia, numer miesiąca i rok. Dlatego właśnie w funkcji *data* zwracaliśmy miesiąc jako liczbę i napis. Napis już wykorzystywaliśmy w pierwszym teście, liczba przyda się teraz. Ponadto będziemy wykorzystywać dwie listy. Elementami pierwszej są liczby będące ilością dni w każdym miesiącu rozpoczynając od stycznia. Tablica ta jest uaktualniana, gdy rok jest rokiem przestępnym (linie 22- 25 Rysunek 7). Lista ta, jest wykorzystywana w celu sprawdzenia poprawności numeru dnia w danym miesiącu (linie 27-28 Rysunek 7). Elementami drugiej listy *dni_tygodnia* są dni tygodnia, tym razem jednak pierwszym elementem jest sobota, drugim niedziela itd. (linie 38-40 Rysunek 7). Ponadto w algorytmie dla miesięcy styczeń i luty, które są wprowadzane jako 1 i 2 odpowiednio zmieniamy ich wartość na 13 i 14, a wartość rok zmniejszamy o 1 (linie 30-32 Rysunek 7). Pozostało już tylko skorzystać ze wzoru:

$$h = \left(d + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + y + \left\lfloor \frac{y}{4} \right\rfloor - \left\lfloor \frac{y}{100} \right\rfloor + \left\lfloor \frac{y}{400} \right\rfloor \right) \bmod 7$$

gdzie d jest numerem dnia w miesiącu, m oznacza numer miesiąca, a y – oznacza rok. Wynik h jest numerem dnia w tygodniu (0 oznacza sobotę, 1 niedzielę, 2 poniedziałek, 3 to wtorek, 4 środa, 5 czwartek, a 6 to piątek), co tłumaczy dlaczego pierwszy dzień na liście to sobota.

Przykład dla daty 1.07.2025.

$$h = \left(1 + \left\lfloor \frac{13(7+1)}{5} \right\rfloor + 2025 + \left\lfloor \frac{2025}{4} \right\rfloor - \left\lfloor \frac{2025}{100} \right\rfloor + \left\lfloor \frac{2025}{400} \right\rfloor \right) \bmod 7$$

$$h = \left(1 + \left\lfloor \frac{104}{5} \right\rfloor + 2025 + [506.25] - [20.25] + [5.0625] \right) \bmod 7$$

$$h = (1 + 20 + 2025 + 506 - 20 + 5) \bmod 7$$

$$h = 2537 \bmod 7 = 3,$$

co oznacza wtorek. Kod funkcji jest na Rysunku 7.

```

1 def zeller(dzien: int, miesiac: int, rok: int) -> str:
2     """
3     Implementuje algorytm Zellera do wyznaczania dnia tygodnia dla daty gregoriańskiej.
4
5     Args:
6         dzien (int): Dzień miesiąca (1-31).
7         miesiac (int): Miesiąc (1-12).
8         rok (int): Rok (>= 1582, początek kalendarza gregoriańskiego).
9
10    Returns:
11        str: Nazwa dnia tygodnia (np. "poniedziałek").
12
13    Raises:
14        ValueError: Jeśli podano nieprawidłową datę.
15
16    Note:
17        Zakłada, że rok jest w kalendarzu gregoriańskim i korzysta z funkcji `czy_przestepny(rok)`
18    """
19    if not all(isinstance(i, int) for i in (dzien, miesiac, rok)):
20        raise TypeError("Wszystkie argumenty muszą być liczbami całkowitymi.")
21
22    if miesiac < 1 or miesiac > 12 or rok < 1582:
23        raise ValueError("Nieprawidłowy miesiąc lub rok.")
24
25    dni_w_miesiacu = [31, 28, 31, 30, 31, 30,
26                      31, 31, 30, 31, 30, 31]
27    if czy_przestepny(rok):
28        dni_w_miesiacu[1] = 29
29
30    if dzien <= 0 or dzien > dni_w_miesiacu[miesiac - 1]:
31        raise ValueError("Niepoprawny numer dnia.")
32
33    if miesiac < 3:
34        miesiac += 12
35        rok -= 1
36
37    d, m, y = dzien, miesiac, rok
38
39    h = (d + (13 * (m + 1)) // 5 + y + (y // 4) - (y // 100) + y//400) % 7
40
41    dni_tygodnia = [
42        "sobota", "niedziela", "poniedzialek",
43        "wtorek", "sroda", "czwartek", "piatek"
44    ]
45
46    return dni_tygodnia[h]

```

Rysunek 7. Algorytm Zellera

5. Wydarzenia

Kolejnym etapem jest przygotowanie słownika zawierającego numery dni i skorelowane z nimi wydarzenia. Słownik, którego kluczami będą numery wybranych dni, a wartościami wydarzenia lub listy wydarzeń umieszczamy w funkcji. Argumentami tej funkcji będą: *numer* oznaczający numer dnia w roku i *ile_dni*, oznaczające ilość dni w roku. Warunek ten jest sprawdzany w linii 35 prezentowanego na Rysunku 8 kodu. Omawiana funkcja korzysta z metody *get*, która gdy znajdzie klucz w słowniku, to zwróci powiązaną z nim wartość, a jeżeli klucza nie ma w słowniku zwróci wartość drugiego argumentu (linia 38 Rysunek 8). Analogicznie jak powyżej kopiujemy funkcję do pliku *funkcje.py* uzupełniamy polecenie *import* i wywołujemy w programie. Ponieważ wynikiem funkcji *wydarzenie* jest lista, aby dodać do napisu kolejne elementy zastosowano pętlę *for* (linie 17- 18 Rysunek 9).

```

1 def wydarzenie(numer: int, ile_dni: int = 365):
2     """
3     Zwraca listę wydarzeń powiązanych z danym dniem w roku.
4
5     Args:
6         numer (int): Numer dnia w roku (1-365 lub 366).
7         ile_dni (int): Liczba dni w roku (365 lub 366). Domyślnie 365.
8
9     Returns:
10        list[str]: Lista wydarzeń przypisanych do danego dnia. Pusta lista, jeśli brak wyda
11
12     Raises:
13        TypeError: Jeśli argumenty nie są typu int.
14        ValueError: Jeśli numer dnia jest poza zakresem lub liczba dni w roku nie jest 365,
15
16     Examples:
17        >>> wydarzenie(1)
18        ['NOWY ROK', 'URODZINY BABCI']
19        >>> wydarzenie(23)
20        []
21        >>> wydarzenie(370)
22        Traceback (most recent call last):
23        ...
24        ValueError: Niepoprawny argument.
25    """
26    wydarzenia = {
27        1: ["NOWY ROK", "URODZINY BABCI"],
28        6: ["TRZECH KRÓLI"],
29        21: ["DZIEŃ BABCI"],
30        73: ["DZIEŃ PI"]
31    }
32
33    if not isinstance(numer, int) or not isinstance(ile_dni, int):
34        raise TypeError("Błąd typu – oba argumenty muszą być liczbami całkowitymi.")
35    if numer <= 0 or numer > ile_dni or ile_dni not in {365, 366}:
36        raise ValueError("Niepoprawny argument.")
37
38    return wydarzenia.get(numer, [])

```

Rysunek 8. Funkcja wydarzenie

```

➔ | 📄 Kalendarz
1 from microbit import *
2 from funkcje import czy_przestepny, data, zeller, wydarzenie
3
4 nr_dnia_w_roku = 1
5 rok = 2025
6
7 while True:
8     napis = "Dzis jest " + str(nr_dnia_w_roku) + " dzien roku, "
9     ile_dni = czy_przestepny(rok) # 365 lub 366
10    numer_dnia, nr_miesiaca, nazwa_miesiaca = data(nr_dnia_w_roku, ile_dni)
11    nazwa_dzien_tygodnia = zeller(numer_dnia, nr_miesiaca, rok)
12    napis += " " + str(numer_dnia)
13    napis += " " + nazwa_miesiaca
14    napis += " " + str(rok)
15    napis += ", " + nazwa_dzien_tygodnia
16    napis += ". Do końca roku pozostało: " + str(ile_dni - nr_dnia_w_roku) + " dni."
17    for el in wydarzenie(nr_dnia_w_roku, ile_dni):
18        napis += " " + el
19    print(napis)
20    sleep(1000)
21    display.scroll(napis)

```

Rysunek 9. Dodanie funkcji: zeller i wydarzenie

6. Nawigacja

Ostatnią funkcjonalnością jest dodanie możliwości sprawdzenia co działo się kilka dni wcześniej lub co będzie się działo za kilka dni. Wykorzystamy dwa przyciski A oraz B i funkcję `was_pressed` wywoływaną na rzecz jednego z przycisków. Funkcja ta zapamięta akcję przyciśnięcia przycisku podczas przewijania tekstu i kolejne wyświetlenie będzie już zaktualizowane. Na Rysunku 10 prezentujemy ostatni fragment kodu.

```
21 | display.scroll(napis)
22 |
23 | if button_a.was_pressed():
24 |     nr_dnia_w_roku -= 1
25 |     if nr_dnia_w_roku == 0:
26 |         rok = rok - 1
27 |         if rok < 1582:
28 |             print("Koniec kalendarza - rok < 1582")
29 |             break
30 |         nr_dnia_w_roku = czy_przestepny(rok)
31 |
32 | if button_b.was_pressed():
33 |     nr_dnia_w_roku += 1
34 |     if nr_dnia_w_roku > ile_dni:
35 |         rok += 1
36 |         nr_dnia_w_roku = 1
```

Rysunek 10. Ostatni fragment głównego pliku.

7. Modyfikacje

Każda grupa z którą pracowaliśmy dodawała swoją funkcjonalność.

Najciekawsze z nich to obliczanie liczby dni do wakacji, czy możliwość wyświetlenia daty juliańskiej, czyli liczby dni, która upłynęła od 1 stycznia roku 4713 p.n.e. Datę tę ustalił w 1583 r. Joseph Scaliger, nazywając na cześć swojego ojca Juliusza Cezara Scaligero nowy sposób liczenia dat „dniami juliańskimi”. Według tej rachuby można przypisać każdemu dniowi następującemu po tej dacie jednoznaczny numer stosowany głównie do obliczeń astronomicznych.

Literatura

1. Kalendarz juliański - wprowadzony przez Juliusza Cezara w 45 r przed naszą erą, <https://kalendarze.net.pl/kalendarz-julianski>, [dostęp 7.06.2025]
2. Historia kalendarza – kto go wymyślił i jakie rodzaje kalendarzy były w historii? National Geographic, <https://www.national-geographic.pl/historia/historia-kalendarza-kto-go-wymyslil-i-jakie-rodzaje-kalendarzy-byly-w-historii/> [dostęp 7.06.2025]
3. Zeller's congruence - Wikipedia, https://en.wikipedia.org/wiki/Zeller%27s_congruence [dostęp 7.06.2025]