

Algorytmy tekstowe

Wprowadzenie, Język Go, Unicode

dr inż. Marcin Ciura
mgc@agh.edu.pl

Wydział Informatyki, Akademia Górniczo-Hutnicza

Plan dzisiejszego wykładu

- Po co nam algorytmy tekstowe?
- Dlaczego Go?
- Regulamin przedmiotu
- Plan wykładów
- Krótki kurs języka Go
- Unicode
- Unicode w Go
- Podsumowanie

Po co nam algorytmy tekstowe?

Żeby wyszukiwać łańcuchy znaków w tekstach, które czytają ludzie

The screenshot shows the LibreOffice Writer application window titled "Saturnin.odt - LibreOffice Writer". The menu bar includes File, Edit, View, Insert, Format, Styles, Table, Form, Tools, Window, and Help. The toolbar contains various icons for document editing. The text area displays two paragraphs. The first paragraph ends with the word "Saturnin" highlighted in a red box. The second paragraph is a continuation of the text. At the bottom, a search bar is active, showing "Saturnin" as the search term, with a dropdown arrow, a "Find All" button, and a "Match Case" checkbox. The status bar at the very bottom indicates "Page 1 of 1", "Selected: 1 word, 8 characters", "Default Page Style", "Polish", and a zoom level of "140%".

Saturnin.odt - LibreOffice Writer

File Edit View Insert Format Styles Table Form Tools Window Help

Nie jestem co prawda entuzjastą wszystkich tych porównań i analogii, którymi doktor Vlach przeplata swe pełne temperamentu tyrady, przyznaję jednak, że w tym jego poglądowym przykładzie z kawiarnią, człowiekiem i tacą z pączkami coś jest. Daje on w pewnym stopniu wyobrażenie o tym, jakim człowiekiem jest **Saturnin**.

Doktor Vlach podzielił sobie mianowicie ludzi według tego, jak potrafią się zachować w na pół pustej kawiarni mając przed sobą tacę z pączkami. Wyobraźcie sobie wytworną kawiarnię w niedzielne przedpołudnie. Pogoda jest piękna, a gości w kawiarni niewielu. Zdążyliście już zjeść śniadanie, przeczytać wszystkie gazety, a teraz oparci wygodnie o miękkie poduszki boksu, spoglądacie w zadumie na tacę z pączkami. Nuda rozłazi się zwolna po wszystkich kątach kawiarni.

I oto teraz właśnie ma się okazać, do której kategorii ludzi według teorii doktora Vlachy należycie. Jeśli jesteście człowiekiem bez fantazji, bez zrozumienia dla dynamiki i bez poczucia humoru, będziecie patrzeć na owe pączki tępo i bezmyślnie, choćby do południa, by wreszcie podnieść się i dość na obiad.

Saturnin

Find All Match Case

Page 1 of 1 Selected: 1 word, 8 characters Default Page Style Polish 140%

Żeby wyszukiwać łańcuchy znaków w takich plikach, które są czytane przez ludzi i przez komputery

Mozart
Sonata in E Minor, K. 304

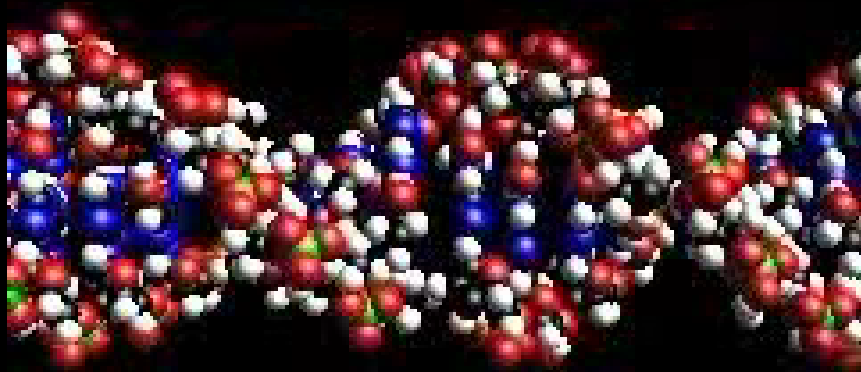
Violin. *Allegro.*
p

Piano. *Allegro.*
p

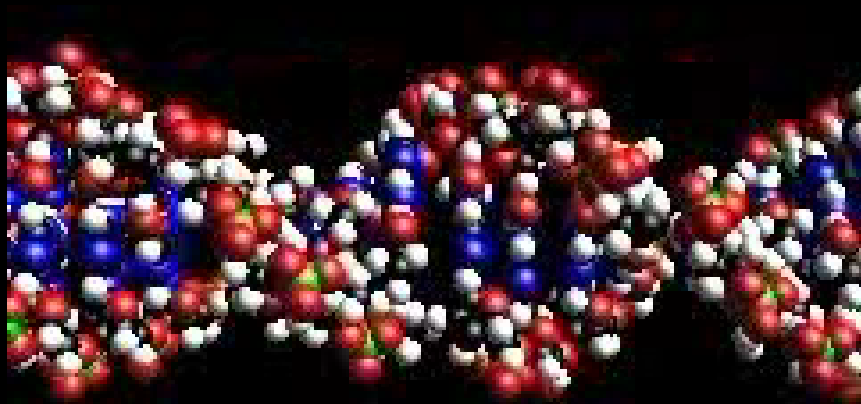
f *pespress.* *p*

The image displays a musical score for Mozart's Sonata in E Minor, K. 304, first movement. The score is for Violin and Piano. The tempo is Allegro. The key signature is one sharp (F#). The score is divided into two systems. The first system shows the beginning of the piece, with the Violin part starting on a treble clef and the Piano part on a grand staff (treble and bass clefs). The second system continues the piece, with the Violin part on a treble clef and the Piano part on a grand staff. The score includes various musical notations such as notes, rests, and dynamic markings (p, f, pespress.). Fingerings are indicated by numbers 1-5. The first system ends with a repeat sign. The second system ends with a final cadence.

Żeby wyszukiwać łańcuchy nukleotydów w łańcuchach DNA



Żeby łączyć krótsze łańcuchy DNA w cały łańcuch DNA



Żeby wykrywać plagiaty



Żeby wykrywać spam



Żeby kompresować dane



Dlaczego Go?

Dlaczego Go?

- Bo programy napisane w Go łatwo zrozumieć
- Bo w programach napisanych w Go nie mogę pomylić typów zmiennych
- Bo gdy program napisany w Go wykonał błędną instrukcję, przerywa pracę i podaje, gdzie w kodzie źródłowym leży błąd
- Bo mogę skompilować ten sam program napisany w Go zarówno pod Linuxem, jak w Windows i na Macintoshu

Dlaczego Go?

- Bo w programach napisanych w Go mogę wygodnie czytać dowolną część każdego łańcucha przez wycinek tego łańcucha
- Bo w programach napisanych w Go mogę wygodnie tworzyć i przetwarzać łańcuchy znaków Unicode

Regulamin przedmiotu

1. Żeby zaliczyć algorytmy tekstowe, wykonam 7 zadań laboratoryjnych

Regulamin laboratorium

1. Kiedy wykonuję zadanie i mam z tym jakiś problem, mówię prowadzącemu zajęcia o tym problemie
2. Jeśli nie skończę zadania podczas zajęć, wykonuję to zadanie po zajęciach, a potem wysyłam do prowadzącego zajęcia email podobny do tego emaila

From: Anna Abacka <anna.abacka@student.agh.edu.pl>
To: Marcin Ciura <mgc@agh.edu.pl>
Subject: Zadanie 1

Dzień dobry,
Wykonałam zadanie 1. Miałam problem z kodowaniem znaku nowej linii w Windows.
Pozdrawiam
Anna Abacka, grupa 8

Często zadawane pytania

P: Do kiedy mam czas, żeby wykonać każde zadanie?

O: Do końca semestru. Im wcześniej wykonam każde zadanie, tym lepiej będę pamiętał, co trzeba było zrobić

Często zadawane pytania

P: Mam problem. Nie wiem, jak go rozwiązać. Co robić?

O: Mogę porozmawiać z inną osobą :-)

Często zadawane pytania

P: Czy mogę pić wodę na zajęciach?

O: Oczywiście. Na zdrowie :-)

Często zadawane pytania

P: Czy mogę rozmawiać na zajęciach?

O: Tak. Śmiało :-)

Często zadawane pytania

P: Czy mogę się śmiać na zajęciach?

O: Tak :-D

Plan wykładów

1. Sprawy organizacyjne, język Go, Unicode
2. Wyrażenia regularne i regexpy
3. Wyszukiwanie wzorca w tekście (1)
4. Wyszukiwanie wzorca w tekście (2)
5. Drzewa sufiksów i tablice sufiksów
6. Kompresja tekstu
7. Pomysłowe algorytmy tekstowe

1. Dan Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge University Press 1997 (część I: Exact String Matching: The Fundamental String Problem i część II: Suffix Trees and Their Uses)
2. Christian Charras i Thierry Lecroq, Exact String Matching Algorithms, <https://www-igm.univ-mlv.fr/~lecroq/string/>
3. Piotr Stańczyk, Algorytmika praktyczna: nie tylko dla mistrzów, Wydawnictwo Naukowe PWN 2009 (rozdział 6: Algorytmy tekstowe)
4. Władysław Skarbek, Metody reprezentacji obrazów cyfrowych, Akademicka Oficyna Wydawnicza PLJ 1993 (część II: Metody kompresji obrazów cyfrowych)

1. Maxime Crochemore i Wojciech Rytter, *Jewels of Stringology*, World Scientific 2003
2. Adam Drozdek, *Wprowadzenie do kompresji danych*, Wydawnictwo Naukowe PWN 2016
3. Thomas H. Cormen, Charles E. Leiserson, Donald L. Rivest, Clifford Stein, *Wprowadzenie do algorytmów*, Wydawnictwo Naukowe PWN 2012
4. Lech Banachowski, Krzysztof Diks, Wojciech Rytter, *Algorytmy i struktury danych*, Wydawnictwo Naukowe PWN 2024

Język Go



Ken Thompson (4.2.1943–), ken



Amerykański programista. Laureat Nagrody Turinga w 1983 roku za stworzenie UNIXa. Opracował algorytm budowania automatów skończonych na podstawie wyrażeń regularnych, stworzył język B, współtworzył system operacyjny Plan 9 i komputer szachowy Belle. Współwynałazł kodowanie UTF-8. W 2006 roku wraz z Robem Pike'iem i Robertem Griesemerem stworzył język Go. Pracuje dla Google.

Rob Pike (1956–), r



Kanadyjski programista, współtwórca języka Go. W 1981 roku stworzył pierwszy system okienkowy dla UNIXa. Współtworzył system operacyjny Plan 9, napisał edytory tekstu **sam** i **acme**. Współwynał kodowanie UTF-8. Wraz z Brianem Kernighanem napisał książki **Lekcja programowania: najlepsze praktyki** i **The UNIX Programming Environment**. Pracuje dla Google. Jego żona, Renée French, stworzyła świstaka języka Go.

Robert Griesemer (1964–)



Szwajcarski programista, współtwórca języka Go. Był doktorantem Niklausa Wirtha. Pracował nad maszyną wirtualną Javy i silnikiem JavaScriptu V8. Pracuje w Google.

Krótki kurs języka Go

- `:=` / `=`
- `:=` / `var`
- najpierw nazwa, potem typ
- `map`
- tablice
- wycinki
- `...`
- `for`
- `if`
- jak testować programy

Deklaracje zmiennych

Tak deklaruje takie zmienne, które mają wartości początkowe

```
zmienna[, zmienna...] := wartość-początkowa[, wartość-początkowa...]
```

Tak deklaruje takie zmienne, które nie mają określonych wartości początkowych

```
var zmienna[, zmienna...] typ
```

Deklaracje zmiennych a przypisanie

Tak deklaruje takie zmienne, które mają wartości początkowe

```
a, b := "Kraków", "Polska"
```

```
s := "Cześć, piękny świecie 😊"
```

Odróżniam symbol := od operatora przypisania =

```
tmp = a
```

```
a = b
```

```
b = tmp
```

```
a, b = b, a
```

```
s = strings.ToUpper(s)
```

Najpierw nazwa, potem typ

```
var s string
```

```
var pat, text []byte
```

Zmienna **s** to łańcuch

Zmienne **pat** i **text** to wycinki, których elementy to liczby typu **byte**

Mapy: `map`

Mapy odwzorowują **klucze** na **wartości**

```
var wordCounter map[string]int  
digitNames := map[int]string{0: "zero", 1: "jeden"}
```

Zmienna **wordCounter** to mapa, której klucze to łańcuchy, a wartości to liczby typu **int**

Zmienna **digitNames** to mapa, która odwzorowuje cyfry 0 i 1 na ich nazwy

Tę część wykładu opracowałem na podstawie artykułu Andrew Gerranda

Go Slices: usage and internals

<https://go.dev/blog/slices-intro>

Tablice

Tablice mają stałą długość



```
array := [5]int
```

Zmienna `intArray` to tablica, która zawiera 5 liczb całkowitych

W języku Go o wiele częściej używa się wycinków niż tablic

Wycinki



Każdy wycinek to struktura, która ma trzy pola

- wskaźnik na 0. element tego wycinka
- liczbę elementów w tym wycinku, czyli jego długość
- pojemność tego wycinka

Mogę zmieniać długość i pojemność wycinka

Mogę tworzyć wycinki dzięki **wyrażeniom wycinkowym**.

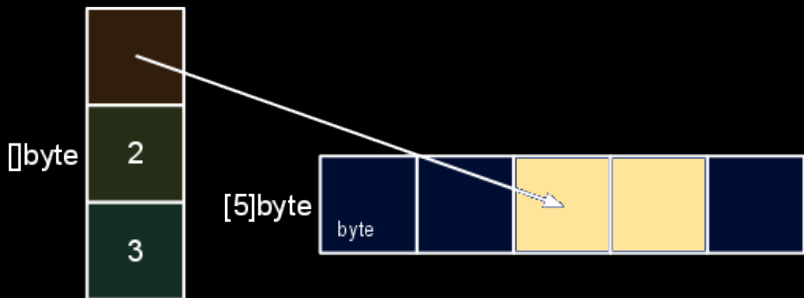
Takie formy wyrażeń wycinkowych są najczęstsze

(łańcuch | tablica | wycinek) [[od]:[do+1]]

Wynik wyrażenia wycinkowego zwraca elementy łańcucha, tablicy lub wycinka od elementu o indeksie **od** do elementu o indeksie **do**

Wynik wyrażenia wycinkowego to wycinek. Elementy tego wycinka mają indeksy od 0 do (**do** — **od**)

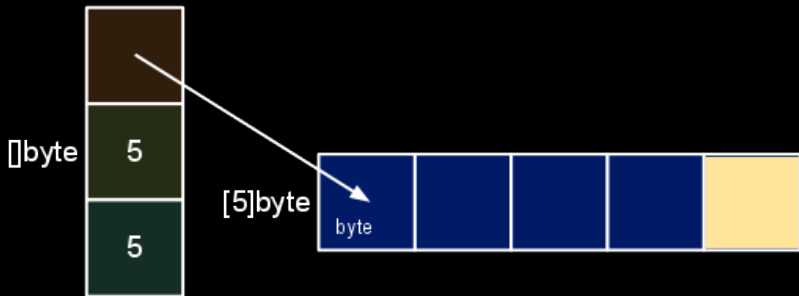
Wycinki



```
slice := array[2:4]
```

Zmienna **slice** to wycinek tablicy **array** od 2. do 3. elementu włącznie

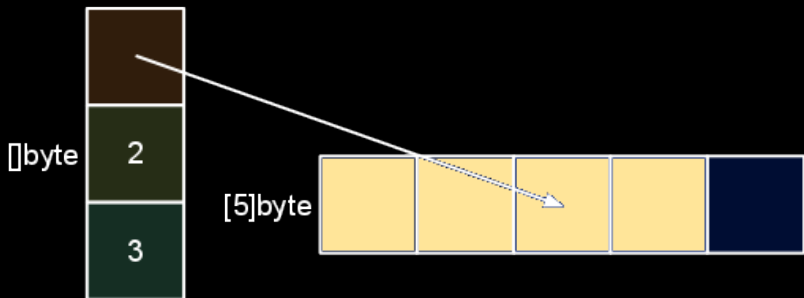
Wycinki



```
slice := array[4:]
```

Zmienna **slice** to wycinek tablicy **array** od 4. do ostatniego elementu włącznie

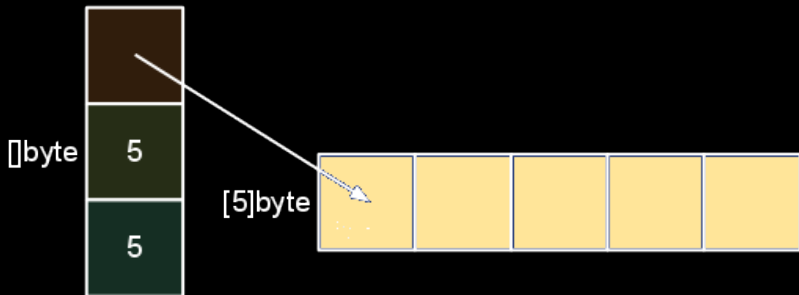
Wycinki



```
slice := array[:4]
```

Zmienna **slice** to wycinek tablicy **array** od 0. do 3. elementu włącznie

Wycinki



```
slice := array[:]
```

Zmienna **slice** to wycinek tablicy **array** od 0. do ostatniego elementu

Wycinki

*// Search porównuje wycinek `pat` z kolejnymi wycinkami
// wycinka `text`. Jeśli wycinek `pat` jest równy pewnemu
// wycinkowi wycinka `text`, Search zwraca indeks zerowego
// elementu tego wycinka wycinka `text`. Jeśli wycinka `pat`
// nie ma w wycinku `text`, Search zwraca -1*

```
func Search(pat, text []byte) int {  
    for i := 0; i + len(pat) <= len(text); i++ {  
        if slices.Equal(pat, text[i:i+len(pat)]) {  
            return i  
        }  
    }  
    return -1  
}
```

Mogę deklarować takie wycinki, które mają wartość początkową

```
slice := []int{2, 0, 2, 4}  
fmt.Println(slice)
```

```
[2 0 2 4]
```


Funkcja wbudowana `append`

Mogę myśleć o wycinkach jak o takich tablicach, których długość można zmieniać

Zmienna `numbers` przechowuje liczby typu `int` od 1 do 999 999

```
numbers := []int{}  
for i := 1; i < 1_000_000; i++ {  
    numbers = append(numbers, i)  
}
```

Funkcja wbudowana `append`

Funkcja `append` dodaje elementy na końcu wycinka. Jeśli pojemność wycinka jest zbyt mała, funkcja `append` kopiuje elementy tego wycinka w inne miejsce pamięci. Dlatego trzeba przypisać wynik funkcji `append` do jakiejś zmiennej

```
slice := []int{2, 0, 2, 4}
fmt.Println(slice)
slice = append(slice, 8)
fmt.Println(slice)
slice2 := append(slice, 0, 0)
fmt.Println(slice2)
```

```
[2 0 2 4]
```

```
[2 0 2 4 8]
```

```
[2 0 2 4 8 0 0]
```

Funkcja wbudowana `append`

Gdy chcę dodać elementy na początku wycinka, też korzystam z funkcji wbudowanej `append`

```
slice := []int{2, 0, 2, 4}
fmt.Println(slice)
slice = append([]int{3}, slice...)
fmt.Println(slice)
```

```
[2 0 2 4]
```

```
[3 2 0 2 4]
```

Funkcje ze zmienną liczbą argumentów...

Tak definiuję i wywołuję funkcję ze zmienną liczbą argumentów

Wewnątrz funkcji **Sum** typ parametru **nums** to **[]int**

```
// Sum zwraca sumę liczb `nums`
```

```
func Sum(nums ...int) {  
    total := 0  
    for _, n := range nums {  
        total += n  
    }  
    return total  
}
```

```
func main() {  
    fmt.Println(Sum(3, 4, 5))  
}
```

Pętla **for**

W Go jest tylko 1 rodzaj pętli: pętla **for**

Pętla **for** ma 4 formy

`for` [*pierwsza-instrukcja*]; [*warunek-zakończenia*]; [*instrukcja-końcowa*]

`for` *warunek-zakończenia*

`for`

`for` *wyrażenie-range*

Pętla **for**

Tak zapisuję pętlę **for**, która ma 3 składniki

```
for i := 0; i < 10; i++ {  
    fmt.Printf("%d ", i)  
}
```

0 1 2 3 4 5 6 7 8 9

Pętla **for**

Tak zapisuję pętlę **for**, która ma 1 składnik. Ta pętla odwraca kolejność elementów tablicy **array**

```
i, j := 0, len(array)-1
for i < j {
    array[i], array[j] = array[j], array[i]
    i++
    j--
}
```

Instrukcja **for**

Tak zapisuję nieskończoną pętlę **for**

```
for {  
    fmt.Println("pętla nieskończona")  
}
```

pętla nieskończona
pętla nieskończona
pętla nieskończona
pętla nieskończona
pętla nieskończona
pętla nieskończona
pętla nieskończona
pętla nieskończona
pętla nieskończona
pętla nieskończona
pętla nieskończona

Słowo kluczowe **range**

Oto najczęstsze formy wyrażenia **range**:

for *indeks*, *element* := range (*tablica* | *wycinek* | *łańcuch*)

for *klucz*, *wartość* := range *mapa*

for *liczba-całkowita* := range *liczba-całkowita*

Słowo kluczowe **range**

```
// Rot13 zastępuje każdą dużą literę w łańcuchu `word`  
// przez odpowiadającą jej małą literę, a potem zastępuje  
// każdą literę alfabetu angielskiego przez tę literę,  
// która znajduje się w alfabecie angielskim 13 pozycji  
// później lub 13 pozycji wcześniej
```

```
func Rot13(word string) string {  
    r := ""  
    for _, c := range strings.ToLower(word) {  
        if c >= 'a' && c <= 'z' {  
            if c <= 'm' {  
                r += string(c + 13)  
            } else {  
                r += string(c - 13)  
            }  
        } else {  
            r += string(c)  
        }  
    }  
    return r  
}
```

Słowo kluczowe **range**

```
type Pair struct {  
    word  string  
    points int  
}
```

Tak tworzę wycinek, którego elementy to struktury typu **Pair**. Każdy element tego wycinka składa się z klucza mapy **wordCounter** i odpowiadającej mu wartości

```
func MakeArrayOfPairs(wordCounter map[string]int) []Pair {  
    r := []Pair{}  
    for w, p := range wordCounter {  
        r = append(r, Pair{w, p})  
    }  
    return r  
}
```

Słowo kluczowe **range**

Tak też mogę zapisać pętlę `for i := 0; i < 10; i++`

```
for i := range 10 {  
    fmt.Println(i)  
}
```

0

1

2

3

4

5

6

7

8

9

Instrukcja warunkowa **if**

Instrukcja warunkowa **if** ma 2 formy

if *wyrażenie*

if *instrukcja; wyrażenie*

1. Testuję osobno każdą funkcję
2. Staram się testować każdą funkcję tak, że kiedy test się wykonuje, program przechodzi przez każdą ścieżkę kodu tej funkcji

Jak testować programy napisane w Go

```
package main
```

```
import (  
    "testing"  
)
```

```
func TestRot13(t *testing.T) {  
    in := "Ten trębach na wieży kościoła gra urwany hejnal"  
    want := "gra geęonpm an jvrżł xbśpvbłn ten hejnal urwany"  
    if got := Rot13(in); got != want {  
        t.Errorf("Rot13(%#v) = %#v want %#v", in, got, want)  
    }  
}
```

- A Tour of Go, czyli przewodnik po Go
<https://go.dev/tour/>
- Kurs, którego twórcą jest Mateusz Szczyrzyca
<https://devopsiarz.pl/programowanie-w-go/>
- Szwedzkie archiwum zadań programistycznych
<https://open.kattis.com/>

Unicode

Émile Baudot (11.9.1845–28.3.1903)



Francuski inżynier. W 1870 roku wynalazł kod, nazwany jego nazwiskiem. Od nazwiska Baudot pochodzi nazwa jednostki częstości modulacji, **bod** (Bd).

(No Model.)

11 Sheets—Sheet 6.

J. M. E. BAUDOT.

PRINTING TELEGRAPH.

No. 388,244.

Patented Aug. 21, 1888.

Fig. 24.

	1	2	3	4	5
A	+	-	-	-	-
B	-	+	+	+	-
C	+	-	+	+	-
D	+	+	+	+	-
E	+	+	-	-	-
F	+	+	-	-	-
G	-	+	+	+	-
H	+	+	-	+	-
I	-	+	+	-	-
J	+	-	-	+	-
K	+	-	-	+	+
L	+	+	-	+	+
M	-	+	-	+	+
N	-	+	+	+	+
O	+	+	+	+	-
P	+	+	+	+	+
Q	+	+	+	+	+
R	-	-	+	+	+
S	-	-	+	+	+
T	+	-	+	-	+
U	+	-	+	-	+
V	+	+	+	-	+
W	-	+	+	-	+
X	-	-	+	-	+
Y	-	-	-	-	+
Z	+	+	-	-	+
0	+	-	-	+	+
1	-	-	-	+	-
2	-	-	-	-	+
3	-	-	-	-	-

INVENTOR:

Jean Maurice Emile Baudot

Taśma perforowana



ASCII

American Standard Code for Information Interchange

	__0	__1	__2	__3	__4	__5	__6	__7
00_	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL
01_	BS	HT	LF	VT	FF	CR	SO	SI
02_	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB
03_	CAN	EM	SUB	ESC	FS	GS	RS	US
04_		!	"	#	\$	%	&	'
05_	()	*	+	,	-	.	/
06_	0	1	2	3	4	5	6	7
07_	8	9	:	;	<	=	>	?
10_	@	A	B	C	D	E	F	G
11_	H	I	J	K	L	M	N	O
12_	P	Q	R	S	T	U	V	W
13_	X	Y	Z	[\]	^	_
14_	`	a	b	c	d	e	f	g
15_	h	i	j	k	l	m	n	o
16_	p	q	r	s	t	u	v	w
17_	x	y	z	{		}	~	DEL

Dygresja 1: Końce wierszy

CR LF carriage return+line feed \r\n

VAX/VMS > CP/M > DOS > Windows, także HTTP, FTP, email,...

LF line feed \n

Unix

Żeby zastąpić windowsowe końce wierszy unixowymi końcami wierszy,
używam programu **tr**

tr -d '\r' < wejście > wyjście

Dygresja 2: sekwencje specjalne ANSI

\033[30;47m czarno na białym

\033[91;44m czerwono na niebieskim

\033[0m znów normalnie

Dygresja 3: kod zakończenia programu

```
PS1='\[\e]0;\w\a\e[34;1m\]\A\[\e[$((${??31:32}));1m\]\w\[\e[0m\]\$ '
```

```
16:56~/Documents$ ls *.pdf
```

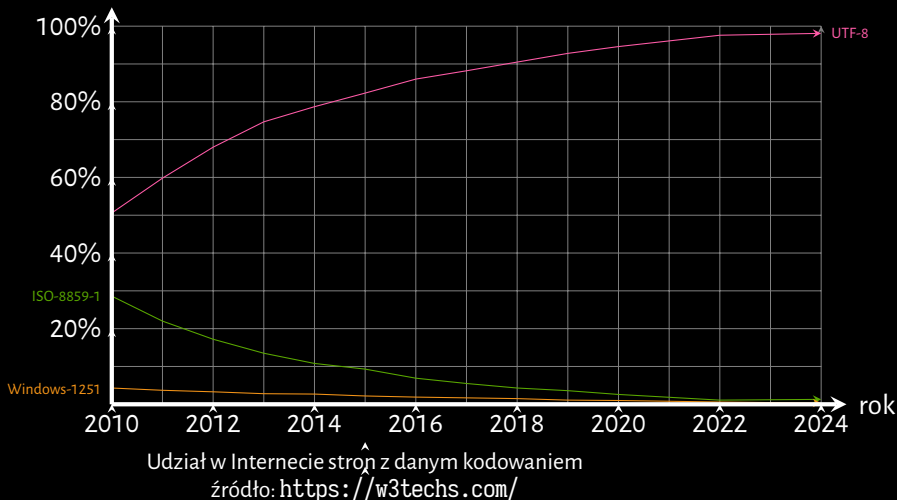
```
wyklad1.pdf      wyklad2.pdf
```

```
16:57~/Documents$ ls *.xls
```

```
ls: cannot access '*.xls': No such file or directory
```

```
16:57~/Documents$
```


Strony kodowe



Żeby zmienić kodowanie pliku na UTF-8, używam programu **iconv**

```
iconv -f strona-kodowa -t utf8 < wejście > wyjście
```

Alfabet cyfrowy c.k. sieci telefonicznych przed 1918 rokiem w Austrii

a	à	ä	b	c	ç	č	d	d'	e	é	ě	f
1	2	3	4	5	6	7	8	9	10	11	12	13
g	h	ch	i	j	k	ck	l	ł	m	n	ň	
14	15	16	17	18	19	20	21	22	23	24	25	
o	ö	p	q	r	ř	s	š	t				
26	27	28	29	30	31	32	33	34				
t'	u	ü	v	w	x	y	z	ž				
35	36	37	38	39	40	41	42	43				

Co to jest Unicode?

Unicode to międzynarodowy standard kodowania, reprezentowania i przetwarzania tekstów. W skład Unicode'u wchodzi między innymi:

- Zbiór niezmiennych **punktów kodowych**. Oznaczam te punkty tak: **U+hhhh** lub tak: **U+hhhhhh**. Na przykład znakowi **A** odpowiada punkt kodowy **U+0041**
- Przepis na to, jak odwzorować punkty kodowe małych liter na punkty kodowe dużych liter
- Przepis na to, jak odwzorować punkty kodowe dużych liter na punkty kodowe małych liter
- Przepis na to, jak składać punkty kodowe w złożone punkty kodowe
- Przepis na to, jak rozkładać złożone punkty kodowe na punkty kodowe
- Przepis na to, jak sortować łańcuchy znaków Unicode'u. Mogę dostosować ten przepis do każdego języka

Każdy **punkt kodowy** to liczba. Tej liczbie odpowiada znak. Znakiem może być

- znak alfabetu: łacińskiego, greckiego, cyrylicy,...
- znak pism spółgłoskowych: arabskiego, hebrajskiego,...
- znak pism sylabicznych: indyjskich, etiopskiego,...
- znak pism CJK: chińskiego, japońskiego, koreańskiego
- cyfra, znak przestankowy, symbol matematyczny, symbol waluty...
- odstęp, znak diakrytyczny,...
- emoji

- Wersja 1.0.0 (1991):
7161 punktów kodowych,
24 systemy pisma,
miejsce na 65 536 punktów kodowych
- Wersja 15.1 (2023):
149 813 punktów kodowych,
161 systemów pisma,
miejsce na $17 \times 65\,536 = 1\,114\,112$ punktów kodowych

17 **obszarów**. Każdy obszar ma 65 536 punktów kodowych

- obszar 0 (od **U+0000** do **U+FFFF**): **Basic Multilingual Plane**
- obszar 1 (od **U+010000** do **U+01FFFF**): wymarłe i bardzo egzotyczne systemy pisma
- obszar 2: (od **U+020000** do **U+02FFFF**) dodatkowe znaki CJK
- obszary 3–13 (od **U+030000** do **U+0DFFFF**): nieużywane
- obszar 14: (od **U+0E0000** do **U+0EFFFF**) dodatkowy obszar specjalnego przeznaczenia
- obszary 15–16 (od **U+0F0000** do **U+10FFFF**): obszary do użytku prywatnego

Punkty kodowe a glify: relacja „wiele do wielu” (1)

Glif: to, co widać

Tym 5 punktom kodowym odpowiada 1(?) glif:

- U+006F o LATIN SMALL LETTER O
- U+03BF o GREEK SMALL LETTER OMICRON
- U+043E o CYRILLIC SMALL LETTER O
- U+0585 o ARMENIAN SMALL LETTER OH
- U+1D0F o LATIN LETTER SMALL CAPITAL O

Homograph spoofing to oszustwo, które polega na użyciu glifów, które wyglądają tak samo jak inne glify, a odpowiadają innym punktom kodowym

Punkty kodowe a glify: relacja „wiele do wielu” (2)

1 punktowi kodowemu: **U+0628** ب **ARABIC LETTER BEH**
odpowiadają 4 glify:

- jako izolowana litera: ب
- na końcu wyrazu: ـب
- w środku wyrazu: ـبـ
- na początku wyrazu: بـ

Punkty kodowe a glify: relacja „wiele do wielu” (3)

Tym 2 punktom kodowym odpowiadają 2 glify:

- U+03C2 ς GREEK SMALL LETTER FINAL SIGMA
- U+03C3 σ GREEK SMALL LETTER SIGMA



Σίσυφος

- U+019B λ LATIN SMALL LETTER LAMBDA WITH STROKE
- U+039B Λ GREEK CAPITAL LETTER LAMDA
- U+03BB λ GREEK SMALL LETTER LAMDA

- U+00DF ß LATIN SMALL LETTER SHARP S
- U+1E9E ß LATIN CAPITAL LETTER SHARP S (od 2008)

W języku niemieckim: **Straße**

`'Straße'.upper()` == `'STRASSE'` lub (od 2017) `'STRAßE'`

- U+0049 | LATIN CAPITAL LETTER I
- U+0069 | LATIN SMALL LETTER I
- U+0130 | LATIN CAPITAL LETTER I WITH DOT ABOVE
- U+0131 | LATIN SMALL LETTER DOTLESS I

Balıkesir

(miasto w Turcji)

- 'Balıkesir'.upper() == 'BALIKESIR'
(oprócz lokalizacji tureckiej i azerskiej)
- 'Balıkesir'.upper() == 'BALIKESİR'
(w lokalizacji tureckiej i azerskiej)

- U+01C4 DŽ LATIN CAPITAL LETTER DZ WITH CARON
- U+01C5 Dž LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON
- U+01C6 dž LATIN SMALL LETTER DZ WITH CARON

W języku chorwackim: džungla

- 'džungla'.upper() == 'DŽUNGLA'
- 'džungla'.title() == 'Džungla'

Normalizacja i dekompozycja punktów kodowych

- mogę **normalizować** punkty kodowe, czyli składać je w złożone punkty kodowe:

U+006F o LATIN SMALL LETTER O +

+ U+0301 ◌ COMBINING ACUTE ACCENT →

→ U+00F3 ó LATIN SMALL LETTER O WITH ACUTE

- mogę **dekomponować** złożone punkty kodowe, czyli rozkładać je na punkty kodowe:

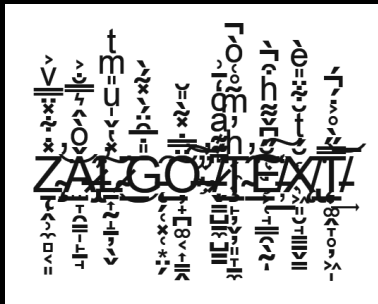
ó → o + ◌

Normalizacja i dekompozycja punktów kodowych

Złożone punkty kodowe mogą się składać z wielu punktów kodowych

Việt Nam

Tekst **zalgo** to zniekształcony tekst, który zawiera dziwne symbole i znaki diakrytyczne









Osobliwości Unicode'u (2)

- U+1F3FB 🍷 EMOJI MODIFIER FITZPATRICK TYPE-1-2
- U+1F3FC 🍷 EMOJI MODIFIER FITZPATRICK TYPE-3
- U+1F3FD 🍷 EMOJI MODIFIER FITZPATRICK TYPE-4
- U+1F3FE 🍷 EMOJI MODIFIER FITZPATRICK TYPE-5
- U+1F3FF 🍷 EMOJI MODIFIER FITZPATRICK TYPE-6

U+263A 🤪 WHITE SMILING FACE

- 🤪 + 🟡 = 🤪
- 🤪 + 🟠 = 🤪
- 🤪 + 🟢 = 🤪
- 🤪 + 🔴 = 🤪
- 🤪 + ⬛ = 🤪

Osobliwości Unicode'u (4)

- U+1F1E6  REGIONAL INDICATOR SYMBOL LETTER A
- U+1F1F1  REGIONAL INDICATOR SYMBOL LETTER L
- U+1F1F5  REGIONAL INDICATOR SYMBOL LETTER P
- U+1F1F8  REGIONAL INDICATOR SYMBOL LETTER S
- U+1F1FA  REGIONAL INDICATOR SYMBOL LETTER U
- U+1F1FF  REGIONAL INDICATOR SYMBOL LETTER Z

Osobliwości Unicode'u (5)

• $\boxed{\text{P}} + \boxed{\text{L}} = \text{🇵🇱}$

• $\boxed{\text{U}} + \boxed{\text{S}} = \text{🇺🇸}$

Kolejność sortowania napisów należy do ustawień lokalizacji

- Naturalne sortowanie liter akcentowanych,
od lewej do prawej:
cote < coté < côte < côté
- W słownikach języka francuskiego,
od prawej do lewej:
cote < côte < coté < côté

- W języku czeskim: $h < ch < i$
- W języku słowackim: $d < d' < dz < dž, h < ch < i$
- W języku hiszpańskim do 1994: $c < ch < d, l < ll < m$

Sortowanie jest trudne

- W niemieckich słownikach: ä = a
- W niemieckich książkach telefonicznych: ä = ae
- W austriackich książkach telefonicznych: ä > az
- W językach szwedzkim i fińskim: z < å < ä < ö

Þingvellir (dolina na Islandii)

- Þingvellir = Pingvellir?
- Þingvellir = Thingvellir?
- żyźnie < Þingvellir?

Jan Žižka (czeski bohater narodowy)

- Žižka = Zizka?
- Žižka = Žizka?
- zżęłyśmy < Žižka < ž?

Kodowanie znaków określa to, jak zamieniać punkty kodowe na ciągi bajtów i to, jak zamieniać ciągi bajtów na punkty kodowe

Kodowanie znaków nie jest częścią standardu Unicode

Skrót UTF oznacza **Unicode Transformation Format**

Oto kodowania znaków, które mają w nazwie litery UTF

- UTF-8
- UTF-16
- UTF-32

Kodowanie znaków: UTF-8

1 znak Unicode'u w kodowaniu UTF-8 zajmuje od 1 do 4 bajtów

Number of bytes	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	U+0000	U+007F	0xxxxxxx			
2	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	U+10000	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Cześć 😊

43 | 7a | 65 | c5 9b | c4 87 | 20 | f0 9f 98 8a

Kodowanie znaków: UTF-16

1 znak Unicode'u w kodowaniu UTF-16 zajmuje 2 lub 4 bajty

- od **U+0000** do **U+D7FF**: 2 bajty
- od **U+D800** do **U+DFFF**: tym punktom kodowym nie odpowiadają żadne znaki Unicode'u, dlatego te punkty kodowe kodują znaki Unicode z obszarów 1–17
- od **U+E000** do **U+FFFF**: 2 bajty
- od **U+010000** do **U+10FFFF**: 4 bajty

Cześć 😊

0043 | 007a | 0065 | c59b | c487 | 0020 | d83d de04

1 znak Unicode'u w kodowaniu UTF-32 zajmuje 4 bajty

- od **U+0000** do **U+10FFFF**: 4 bajty

Little-endian i big-endian

Termin **little-endian** oznacza taki sposób przechowywania wielobajtowych danych w pamięci komputera, w którym pierwsza komórka pamięci przeznaczona na dane przechowuje najmniej znaczący bajt danych, a kolejne komórki pamięci przechowują kolejne, coraz bardziej znaczące bajty

Oto liczba szesnastkowa **0x1234** zapisana w systemie little-endian

bajt 1 bajt 2

0x34 **0x12**

W trybie little-endian pracują między innymi procesory Intel x86 i RISC-V, które są stosowane w komputerach osobistych, i procesory ARM, które są stosowane w urządzeniach mobilnych

Little-endian i big-endian

Termin **big-endian** oznacza taki sposób przechowywania wielobajtowych danych w pamięci komputera, w którym pierwsza komórka pamięci przeznaczona na dane przechowuje najbardziej znaczący bajt danych, a kolejne komórki pamięci przechowują kolejne, coraz mniej znaczące bajty

Oto liczba szesnastkowa **0x1234** zapisana w systemie big-endian

bajt 1	bajt 2
--------	--------

0x12	0x34
-------------	-------------

W trybie big-endian pracują między innymi procesory Motorola 68000, które były stosowane między innymi w komputerach Amiga

Little-endian i big-endian

Kodowania UTF-16LE i UTF-32LE to takie kodowania, w których ciągi bajtów są zapisywane w kolejności little-endian

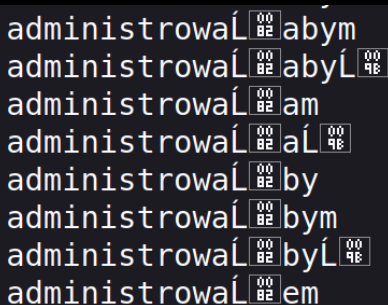
Kodowania UTF-16BE i UTF-32BE to takie kodowania, w których ciągi bajtów są zapisywane w kolejności big-endian

Tak zamieniam znak Unicode'u **U+0041** A **LATIN LETTER CAPITAL A**
na ciąg bajtów w różnych kodowaniach

- UTF-8: **0x41**
- UTF-16LE: **0x41 0x00**
- UTF-16BE: **0x00 0x41**
- UTF-32LE: **0x41 0x00 0x00 0x00**
- UTF-32BE: **0x00 0x00 0x00 0x41**

Mojibake czyli krzaczk

Mojibake czyli **krzaczk** to zniekształcony tekst, który powstaje, gdy tekst jest dekodowany z użyciem innego kodowania znaków niż to kodowanie, w którym został zakodowany



The image shows a terminal window with a dark background. It displays eight lines of text, each representing a different encoding of the word 'administrowałem'. Each character is shown inside a small box that also displays its hexadecimal byte value. The boxes for the 'ł' character in each line show different byte values, illustrating how the same character is represented differently in various encodings, leading to mojibake when decoded incorrectly.

```
administrowaŁ 00 82 abým  
administrowaŁ 00 82 abyÍ 00 9B  
administrowaŁ 00 82 am  
administrowaŁ 00 82 aÍ 00 9B  
administrowaŁ 00 82 by  
administrowaŁ 00 82 bym  
administrowaŁ 00 82 byÍ 00 9B  
administrowaŁ 00 82 em
```

Jak mam wysłać łańcuchy znaków Unicode'u do innych systemów, żeby te inne systemy mogły je poprawnie zdekodować?

Znacznik kolejności bajtów

Jak mam wysyłać łańcuchy znaków Unicode'u do innych systemów, żeby te inne systemy mogły je poprawnie zdekodować?

Mam najpierw przesłać **znacznik kolejności bajtów U+FEFF BYTE ORDER MARK**

- UTF-8: **0xEF 0xBB 0xBF**
- UTF-16LE: **0xFF 0xFE**
- UTF-16BE: **0xFE 0xFF**
- UTF-32LE: **0xFF 0xFE 0x00 0x00**
- UTF-32BE: **0x00 0x00 0xFE 0xFF**

Znacznikowi kolejności bajtów odpowiada w każdym kodowaniu inny ciąg bajtów

Porównanie kodowań: UTF-8

- Jak używać kodowania UTF-8?
 - Go: **string** i **[]byte**
 - C i C++: funkcje **mbstowcs** i **wcstombs** konwertują między multibyte strings (UTF-8) i wide-character strings (UTF-16 albo UTF-32)
- + łańcuch znaków w kodowaniu UTF-8 łatwo jest przeglądać w przód
- łańcuch znaków w kodowaniu UTF-8 trudniej jest przeglądać wstecz
- żeby znaleźć *n*-ty znak łańcucha, trzeba przejrzeć po kolei poprzednie znaki, czyli złożoność indeksowania łańcuchów znaków jest liniowa
- + łańcuch znaków w kodowaniu UTF-8 jest krótki, gdy jego znaki pochodzą z języków europejskich
- + łańcuch znaków w kodowaniu UTF-8 jest taki sam w trybie little-endian i w trybie big-endian

Porównanie kodowań: UTF-16

- Jak używać kodowania UTF-16?
 - C w Windows: `wchar_t*`
 - C++ w Windows: `std::wstring`
 - Java: `String`
 - JavaScript: `String`
- + łańcuch znaków w kodowaniu UTF-16 łatwo jest przeglądać w przód
- + łańcuch znaków w kodowaniu UTF-16 łatwo jest przeglądać wstecz, gdy punkty kodowe wszystkich jego znaków leżą w obszarze 0, czyli są $\leq U+FFFF$
- + łatwo znaleźć n -ty punkt kodowy łańcucha znaków w kodowaniu UTF-16, gdy punkty kodowe wszystkich znaków tego łańcucha leżą w obszarze 0

Porównanie kodowań: UTF-32

- Jak używać kodowania UTF-32?
 - Go: `[]rune`
 - C pod Unixami: `wchar_t*`
 - C++ pod Unixami: `std::wstring`
- + łańcuch znaków w kodowaniu UTF-32 łatwo jest przeglądać w przód
- + łańcuch znaków w kodowaniu UTF-32 łatwo jest przeglądać wstecz
- + łatwo znaleźć *n*-ty punkt kodowy łańcucha znaków w kodowaniu UTF-32, czyli złożoność indeksowania łańcuchów znaków jest stała

- U+A668 ⦿ CYRILLIC CAPITAL LETTER MONOCULAR O
- U+A669 ⦿ CYRILLIC SMALL LETTER MONOCULAR O

W niektórych rękopisach cerkiewnosłowiańskich:

ⲐⲚⲟ
(oko)

Osobliwości Unicode'u (8)

- U+A66A ☐ CYRILLIC CAPITAL LETTER BINOCULAR O
- U+A66B ☐ CYRILLIC SMALL LETTER BINOCULAR O
- U+A66C ☐ CYRILLIC CAPITAL LETTER DOUBLE MONOCULAR O
- U+A66D ☐ CYRILLIC SMALL LETTER DOUBLE MONOCULAR O

W niektórych rękopisach cerkiewnosłowiańskich:

ⲠⲚⲓ ⲞⲚⲓ

(oczy)

· U+A66E ❀ CYRILLIC LETTER MULTIocular 0

W jednym rękopisie cerkiewnośłowiańskim:

серафими многоꙋчитїй
(wieloocy serafini)

Unicode w Go

Unicode w Go

W Go typ **string** reprezentuje **niemodyfikowalne** łańcuchy znaków Unicode'u w kodowaniu UTF-8

Tak mogę odczytywać znaki z wartości, które mają typ **string**

```
s := "Cześć, piękny świecie :-)"  
c := s[3]
```

Tak mogę **konkatenować** wartości, które mają typ **string**

```
s := s + " " + s
```

Tak mogę przeglądać znaki w wartościach, które mają typ **string**

```
for _, c := range s
```

Zmienna **c** ma type **rune**

Typ **rune** reprezentuje 1 znak Unicode'u

Typ **rune** to alias typu **int32**, który oznacza 32-bitowe liczby całkowite ze znakiem

Zmienne typu **rune** mogą przyjmować wartości od -2147483648 do $+2147483647$

Typ `[]byte` reprezentuje **modyfikowalne** ciągi bajtów.

Tak mogę konwertować wyrażenia, które mają typ `string`, na typ `[]byte`

```
s := "cześć"
```

```
bs := []byte(s)
```

```
bs[0] = 'C'
```

```
fmt.Println(s, bs)
```

```
cześć [67 122 101 197 155 196 135]
```


Tak mogę odczytywać bajty z wartości, które mają typ `[]byte`

```
c := bs[2]
```

Tak mogę zapisywać bajty do zmiennych, które mają typ `[]byte`

```
bs[0] := ' '
```

Tak mogę przeglądać bajty wartości, które mają typ `[]byte`

```
for _, b := range bs
```

Zmienna `b` ma type `byte`

Typ `byte` reprezentuje 1 bajt

Typ `byte` to alias typu `uint8`, który oznacza 8-bitowe liczby całkowite bez znaku

Zmienne typu `byte` mogą przyjmować wartości od 0 do 255

Podsumowanie

1. Żeby zaliczyć algorytmy tekstowe, wykonam 7 zadań laboratoryjnych

1. Kiedy wykonuję zadanie i mam z tym jakiś problem, mówię prowadzącemu zajęcia o tym problemie
2. Jeśli nie skończę zadania podczas zajęć, wykonuję to zadanie po zajęciach, a potem wysyłam email do prowadzącego zajęcia

- `:=` / `=`
- `:=` / `var`
- najpierw nazwa, potem typ
- `map`
- tablice
- wycinki
- `...`
- `for`
- `if`
- jak testować programy

Unicode

- **Punkt kodowy**: liczba
- **Glif**: to, co widać
- Kodowanie znaków:
 - UTF-8, UTF-16, UTF-32
 - **little-endian** i **big-endian**
 - znacznik kolejności bajtów
- Unicode w Go: **string**, **[]byte**

Pomysły, uwagi, pytania, sugestie

Proszę wysyłać podpisane pomysły, uwagi, pytania, sugestie na temat wykładów lub laboratoriów na adres mgc@agh.edu.pl

lub wpisywać anonimowe pomysły, uwagi, pytania, sugestie pod adresem <https://tiny.cc/algorytmy-tekstowe>

Do zobaczenia na następnym wykładzie

Jego tematem będą regexpy i wyrażenia regularne

Źródła zdjęć

- Zdeněk Jírotka, Saturnin – zrzut ekranu
- [https://imslp.org/wiki/Violin_Sonata_in_E_minor,_K.304/300c_\(Mozart,_Wolfgang_Amadeus\)](https://imslp.org/wiki/Violin_Sonata_in_E_minor,_K.304/300c_(Mozart,_Wolfgang_Amadeus))
- https://commons.wikimedia.org/wiki/File:DNA123_rotated.png
- <https://openclipart.org/detail/344870/programmers-keyboard>
- <https://www.flickr.com/photos/jeepersmedia/14010924432/>
- <https://classic.csunplugged.org/activities/text-compression/>
- <https://go.dev/blog/gopher>
- https://en.wikipedia.org/wiki/Ken_Thompson
- https://en.wikipedia.org/wiki/Rob_Pike
- https://en.wikipedia.org/wiki/Robert_Griesemer
- <https://go.dev/blog/slices-intro>
- https://en.wikipedia.org/wiki/%C3%89mile_Baudot
- https://commons.wikimedia.org/wiki/File:Le_Mythe_de_Sisyphe.png
- https://en.wikipedia.org/wiki/Zalgo_text

Źródła zdjęć

Tabela kodowania UTF-8: Rob Pike, Creative Commons

Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

Tabela pochodzi ze strony

https://wiki.ubc.ca/File:FSS-UTF_1992_UTF-8_1993.png

Odciałem z tej tabeli 2 ostatnie kolumny i 2 ostatnie wiersze

Wygaszacz ekranu GLMatrix: Copyright © 1999-2003 by Jamie Zawinski.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. No representations are made about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.