

Przetwarzanie języka naturalnego

dr inż. Marcin Ciura

Wydział Informatyki i Telekomunikacji Politechniki Krakowskiej

Plan na dziś: 67 slajdów, w tym 5 dygresji

- Sprawy organizacyjne
- Python
- Unicode

Sprawy organizacyjne

Plan wykładów

1. Sprawy organizacyjne, Python i Unicode
2. Wyrażenia regularne i regexpy
3. Gramatyki
- 4.
- 5.
- 6.
- 7.
- 8.
- 9.
- 10.

- Treści zadań: <https://github.com/PK-PJN>
- Praca indywidualna
- Python 3.x
- Sprawozdania jako PDF-y

Warunki zaliczenia (1): punkty cząstkowe

- Oceny z 10 laboratoriów: 0-100
- Aktywność na jednych zajęciach: +2

Warunki zaliczenia (2): punkty końcowe

*średnia punktów z laboratoriów +
+ suma punktów za aktywność +
– 25 × liczba nieusprawiedliwionych nieobecności*

Warunki zaliczenia (3): ocena końcowa

5,0: od 90,01 do ∞

4,5: od 80,01 do 90,00

4,0: od 70,01 do 80,00

3,5: od 60,01 do 70,00

3,0: od 50,01 do 60,00

2,0: od 0,00 do 50,00

Pomysły, uwagi, pytania, sugestie, narzekania

`marcin.ciura@pk.edu.pl`

Dygresja 1: trochę rozrywki

- Turniej Elektrybałtów: Centonista
- 25th International Obfuscated C Code Contest
- PolEval 2019: Przetak

Python

Python w 9 slajdach (1): wcięcia

```
// Inne języki:  
for (warunki pętli)  
{  
    coś;  
    jeszcze_coś;  
}  
coś_poza_pętlą();
```

```
# Python:  
for warunki pętli:  
    coś  
    jeszcze_coś  
coś_poza_pętlą()
```

Python w 9 slajdach (2): funkcje

```
def nazwa_funkcji(argument1, argument2):  
    coś przed pętlą  
    for warunki_pętli:  
        wewnątrz_pętli  
        i to też  
    return wynik
```

Python w 9 slajdach (3): listy

```
prowiant = ['chleb', 'mleko', 'jaja']  
prowiant[0] = 'bułki'  
prowiant[-1] = 'ser'  
prowiant.append('masło')  
  
kup_z_listy(prowiant[1:-2])  
kup_z_listy(prowiant[-2:])
```

Python w 9 slajdach (4): zbiory

```
kwiaty = {'róże', 'fiołki'}  
kwiaty.add('róże')  
kwiaty.add('przebiśniegi')  
if 'fiołki' in kwiaty:  
    print('Mamy fiołki')
```

Python w 9 slajdach (5): słowniki

```
imiona_osób = {  
    'Kowalski': 'Jan',  
    'Nowak': 'Adam',  
    'Wójcik': 'Anna',  
}  
imiona_osób['Kowalczyk'] = 'Andrzej'  
imiona_osób['Nowak'] = 'Anna'  
if 'Nowak' in imiona_osób:  
    print('Nowak ma na imię',  
imiona_osób['Nowak'])
```


Python w 9 slajdach (6): pętle

```
for prowiant in ['bułki', 'mleko', 'ser']:  
    kup(prowiant)
```

```
for i in range(początek, koniec + 1):  
    print(i)
```

```
while True:  
    rzecz = input('Co kupić? ')  
    if rzecz == '':  
        break  
    kup(rzecz)
```

Python w 9 slajdach (7): generatory

```
def generuj_liczby_naturalne(n):  
    for i in range(n + 1):  
        yield i  
  
for i in generuj_liczby_naturalne(5):  
    print(i)
```

Python w 9 slajdach (8): instrukcja warunkowa

```
if x > 3:  
    print('Dużo')
```

```
if x > 3:  
    print('Dużo')  
else:  
    print('Mało')
```

```
if x > 3:  
    print('Dużo')  
elif x == 3:  
    print('Trzy')  
elif x == 2:  
    print('Dwa')  
else:  
    print('Mało')
```

Python w 9 slajdach (9): napisy

```
fragment = '...(Ała ma kota)...'  
zdanie = fragment.strip('().')  
wyrazy = zdanie.split()  
zdanie_z_kropką = ' '.join(wyrazy) + '.'  
pytanie = 'Czy {} ma {}?'.format(  
    'Ała', 'żółwia')
```

Unicode: prehistoria

(No Model.)

11 Sheets—Sheet 6.

J. M. E. BAUDOT.

PRINTING TELEGRAPH.

No. 388,244.

Patented Aug. 21, 1888.

Fig. 2A.

	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	+	+	—	—	+
E	+	—	—	—	+
S	—	—	—	+	+
N	—	—	—	—	—
—	—	—	—	—	+
—	—	—	—	—	—

INVENTOR:

Jean Maurice Emile Baudot

Taśma perforowana



Alfabet cyfrowy c.k. sieci telefonicznych

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				

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

Escape sequences

NUL \0: null character
HT \t: horizontal tab
CR \r: carriage return
LF \n: line feed
ESC \033: escape
\\: backslash

Końce wierszy

\r\n: VAX/VMS > CP/M > DOS > Windows
także HTTP, FTP, email...

\n: Unix
tr -d '\r' < *wejście* > *wyjście*

Dygresja 2 (*off-topic*): sekwencje specjalne ANSI

\033[30;47m czarno na białym

\033[91;44m czerwono na niebieskim

\033[0m znów normalnie

Dygresja 2 (*off-topic*): kod zakończenia programu

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

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

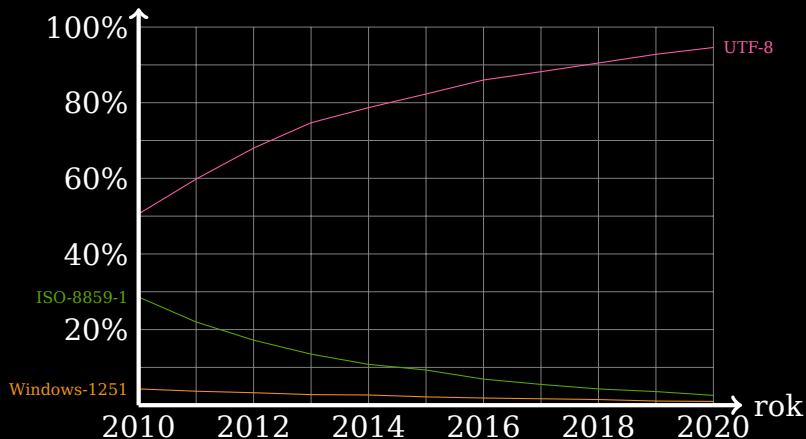
```
wyklad1.pdf    wyklad2.pdf
```

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

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

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

Strony kodowe



Udział w Internecie stron z danym kodowaniem

źródło: <https://w3techs.com/>

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

Unicode

Czym jest Unicode?

Międzynarodowym standardem kodowania, reprezentowania i przetwarzania tekstów, w którego skład wchodzi między innymi:

- zbiór niezmiennych *code points*, oznaczanych jako U+*hhhh* lub U+*hhhhhh*
- odwzorowanie między *code points* małych i wielkich liter
- reguły składania ciągów *code points* i dekompozycji złożonych *code points*
- algorytm sortowania napisów, który można dostosować do każdego języka

Code points

Code point to liczba, której odpowiada znak:

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

Krótką historia Unicode'u

- Wersja 1.0.0 (1991):
7161 *code points*,
24 systemy pisma,
„65 536 *code points* powinno wystarczyć każdemu”
- Wersja 12.1 (2019):
137 994 *code points*,
150 systemów pisma,
miejsce na $17 \times 65\,536 = 1\,114\,112$ *code points*

Płaszczyzny Unicode'u

17 płaszczyzn po 65 536 *code points* każda:

- płaszczyzna 0: *Basic Multilingual Plane*
- płaszczyzna 1: wymarłe i bardzo egzotyczne systemy pisma
- płaszczyzna 2: dodatkowe znaki CJK
- płaszczyzny 3–13: nieużywane
- płaszczyzna 14: *symbole wskaźników regionalnych*
- płaszczyzny 15–16: obszar do użytku prywatnego

Code points a glify: relacja „wiele do wielu” (1)

Glif: to, co widać

5 *code points*, 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: oszustwo za pomocą jednakowo wyglądających glifów, które odpowiadają różnym *code points*

Code points a glify: relacja „wiele do wielu” (2)

1 *code point*:

U+0628 ب ARABIC LETTER BEH

4 glify:

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

Code points a glify: są równi i równiejsi

2 code points i 2 glify:

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

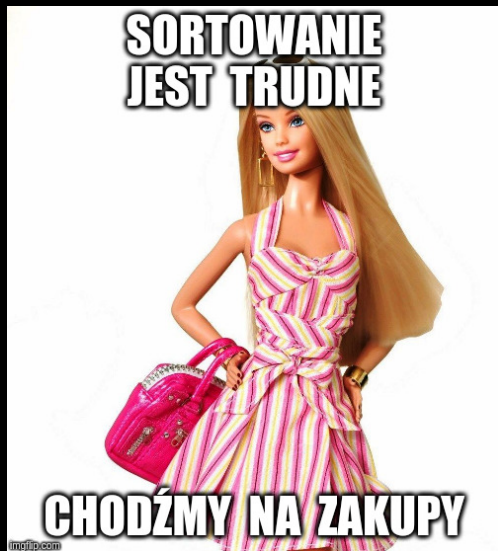
Σίσυφος

Code points a glify: combining character sequences

- U+006F o LATIN SMALL LETTER O +
U+0301 ◌ COMBINING ACUTE ACCENT = ó
- U+00F3 ó LATIN SMALL LETTER O WITH ACUTE
- dekompozycja: ó \rightarrow o + ◌
- normalizacja: o + ◌ \rightarrow ó

Kolejność sortowania napisów jest własnością ustawień regionalnych (*locale*), niezależną od napisów

Dygresja 3: sortowanie



Dygresja 3: sortowanie (1)

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

Dygresja 3: sortowanie (2)

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

Dygresja 3: sortowanie (3)

- W języku czeskim: $h < ch < i$
- W języku słowackim: $d < d' < dz < d\check{z}, h < ch < i$
- W języku hiszpańskim do 1994: $c < ch < d, l < ll < m$

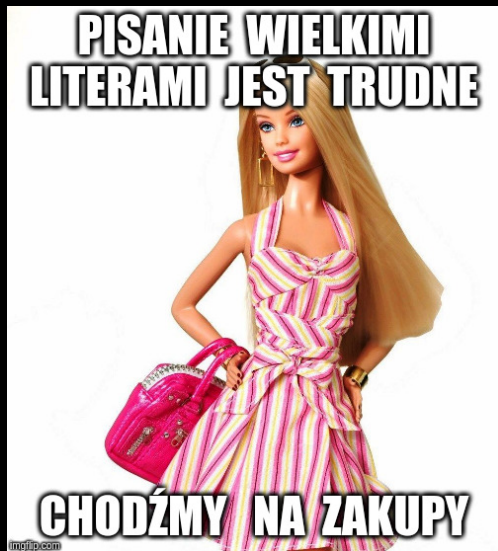
Þingvellir *(dolina na Islandii)*

- Þingvellir = Pingvellir?
- Þingvellir = Thingvellir?
- żżonymi < Þingvellir?

Jan Žižka (*czeski bohater narodowy*)

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

Dygresja 4: wielkie i małe litery



Dygresja 4: wielkie i małe litery (1)

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

W języku niemieckim: **Straße**

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

Dygresja 4: wielkie i małe litery (2)

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

Balıkesir *(miasto w Turcji)*

- 'Balıkesir'.upper() == 'BALIKESIR'
(oprócz *locale* tureckiego i azerskiego)
- 'Balıkesir'.upper() == 'BALIKESİR'
(w *locale* tureckim i azerskim)

Dygresja 4: wielkie i małe litery (3)

- 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
- U+01C7 LJ LATIN CAPITAL LETTER LJ
- U+01C8 Lj LATIN CAPITAL LETTER L WITH SMALL LETTER J
- U+01C9 lj LATIN SMALL LETTER LJ
- U+01CA NJ LATIN CAPITAL LETTER NJ
- U+01CB Nj LATIN CAPITAL LETTER N WITH SMALL LETTER J
- U+01CC nj LATIN SMALL LETTER NJ

W języku chorwackim: džungla

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

Kodowanie znaków

Standard Unicode nie zajmuje się zapisywaniem i odczytywaniem *code points*

Dopiero *kodowanie znaków* określa, jak zamieniać *code points* na bajty i bajty na *code points*

UTF = *Unicode Transformation Format*

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

Kodowanie znaków: UTF-8

Od 1 do 4 bajtów na znak

- od U+0000 do U+007F (ASCII): 1 bajt
- od U+0080 do U+07FF (Europa i okolice): 2 bajty
- od U+0800 do U+FFFF (prawie wszystko): 3 bajty
- od U+010000 do U+10FFFF: 4 bajty

Kod samosynchronizujący się: ani końcówka kodu jednego code point, ani fragment zlepionych kodów dwóch code points nie są poprawnymi kodami

Kodowanie znaków: UTF-16

2 lub 4 bajty na znak

- od U+0000 do U+D7FF: 2 bajty
- od U+D800 do U+DFFF: te *code points* nie mają przypisanych znaków
- od U+E000 do U+FFFF: 2 bajty
- od U+010000 do U+10FFFF: 4 bajty

Kodowanie znaków: UTF-32

4 bajty na znak

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

Little-endianness i big-endianness

U+0041 A LATIN LETTER CAPITAL A

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

Jak poprawnie dekodować pliki/strumienie bajtów
wymieniane między systemami?

Znacznik kolejności bajtów

> Jak poprawnie dekodować pliki/strumienie bajtów wymieniane między systemami?

Zacząć od przesłania *znacznika 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

Porównanie kodowań: UTF-8

- Użycie:
 - wymiana danych: pliki tekstowe, HTML, JSON... (kodowanie UTF-8 samo się synchronizuje)
 - C i C++: funkcje `mbstowcs()` i `wcstombs()` konwertują między multibyte strings (UTF-8) i wide-character strings (UTF-16 albo UTF-32)
 - Go: `string`
 - Python 3.3+: `str` z *code points* $\leq U+007F$
- + oszczędza pamięć, jeśli napisy są w językach europejskich
- + iterowanie w przód napisu
- + `startswith()`
- + `endswith()`
- iterowanie po napisie wstecz
- dostęp do *i*-tego *code point*

Porównanie kodowań: UTF-16

- Użycie:
 - C pod Windows: `wchar_t*`
 - C++ pod Windows: `std::wstring`
 - Java: `String`
 - JavaScript: `String`
 - Python 3.3+: `str` z *code points* $\leq U+FFFF$
- więcej pamięci na napisy w językach europejskich
- + iterowanie w przód napisu
- + `startswith()`
- + `endswith()`
- + iterowanie po napisie wstecz,
jeśli *code points* $\leq U+FFFF$
- + dostęp do *i*-tego *code point*,
jeśli *code points* $\leq U+FFFF$

Porównanie kodowań: UTF-32

- Użycie:
 - C pod Unixami: `wchar_t*`
 - C++ pod Unixami: `std::wstring`
 - Python 3.3+: `str` z *code points* $\geq U+010000$
- zużywa dużo pamięci na napisy
- + iterowanie w przód napisu
- + `startswith()`
- + `endswith()`
- + iterowanie po napisie wstecz
- + dostęp do *i*-tego *code point*

Dygresja 5: osobliwości Unicode'u (1)

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

Dygresja 5: 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

Dygresja 5: osobliwości Unicode'u (3)







U+263A 😊 WHITE SMILING FACE

- 😊 + 🟡 = 😊
- 😊 + 🟠 = 😊
- 😊 + 🟤 = 😊
- 😊 + 🟡 = 😊
- 😊 + 🟤 = 😊

Dygresja 5: 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

Dygresja 5: osobliwości Unicode'u (5)

-  +  = 
-  +  = 

Dygresja 5: osobliwości Unicode'u (6)

- U+A668 О CYRILLIC CAPITAL LETTER MONOCULAR O
- U+A669 о CYRILLIC SMALL LETTER MONOCULAR O

Dygresja 5: osobliwości Unicode'u (7)

W niektórych rękopisach cerkiewnosłowiańskich:

ⲐⲚⲟ
(*oko*)

Dygresja 5: osobliwości Unicode'u (8)

- U+A66A О CYRILLIC CAPITAL LETTER BINOCULAR O
- U+A66B о CYRILLIC SMALL LETTER BINOCULAR O
- U+A66C ОО CYRILLIC CAPITAL LETTER DOUBLE MONOCULAR O
- U+A66D оо CYRILLIC SMALL LETTER DOUBLE MONOCULAR O

Dygresja 5: osobliwości Unicode’u (9)

W niektórych rękopisach cerkiewnosłowiańskich:

⦿ЧИ ⦿⦿ЧИ
(oczy)

Dygresja 5: osobliwości Unicode'u (10)

- U+A66E  CYRILLIC LETTER MULTIOCLULAR O

Dygresja 5: osobliwości Unicode’u (11)

W jednym rękopisie cerkiewnosłowiańskim:

серафими многоꙋчитіи
(*wieloocy serafini*)

Podsumowanie

- Wcięcia
- Funkcje i generatory
- Kolekcje: listy, zbiory, słowniki
- Konstrukcje sterujące: instrukcja warunkowa, pętle
- Napisy

- *Code point*: liczba
- *Glif*: to, co widać
- Kodowanie znaków:
 - UTF-8, UTF-16, UTF-32: zalety i wady
 - *little-endianness* i *big-endianness*
 - znacznik kolejności bajtów

**Do zobaczenia
na następnym wykładzie
o wyrażeniach regularnych i regexpach**
