Textos

Caracteres

Agora sim vamos falar do último dos 4 tipos primários que abordaremos que é o tipo usado para armazenar texto.

Tudo o que você aprendeu até agora sobre protocolos e métodos especiais também se aplica aos textos, mas os textos tem uma pequena particularidade, eles são formados por caracteres.

```
>>> chr(65)
A
>>> chr(66)
B
>>> chr(67)
C
```

Portanto o texto `"ABC"` internamente contém um conjunto de 3 caracteres em suas respectivas posições na tabela de caracteres.

Existem várias tabelas de caracteres usadas na computação mas nesse treinamento vamos ficar em apenas duas `ascii` e `utf8`.

A tabela `ASCII` possui 128 posições, ou seja, vai do 0 ao 127 e em cada posição armazena apenas um caracter.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[ENG OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*

Estes são os carecteres básicos da lingua inglesa e como pode perceber ela não considera acentuação ou carecteres especiais de outros idiomas como Russo ou Mandarim.

Quando a computação globalizou foi preciso mudar de tabela e adotar uma maior que pudesse comportar uma quantidade universal de caracteres e também os `emojis` que se tornaram parte da comunicação moderna.

A tabela `unicode` de `8 bits` - **utf8** atualmente tem 120 mil caracteres.

https://unicode-table.com/en/

Nesta tabela além da tabela ASCII padrão, apartir da posição `128` temos acentuação e sub tabelas para simbolos e emojis.

Na tabela `ASCII` cada caracter ocupava menos de 1 byte (7 bits) e por isso que `A` é `65` que na tabela é `1000001` (7 digitos).

Já na tabela unicode cada caractere pode ser formado por mais de um byte, por exemplo, uma letra com acento `Ã` ocupa 2 bytes `11000011 10000011` na tabela.

E alguns emojis como o 🍉 ocupam 4 bytes `11110000 10011111 10001101 10001001`

Durante a programação com Python nós iremos considerar que nossos textos utilizam os caracteres disponíveis na tabela `utf8` e em alguns raros casos no Python3 teremos que explicitamente fazer operações de `encode` e `decode` a partir de um texto `ascii` para `utf-8`.

```
# variável
fruit = "🍉"

# para transmitir este texto ou gravar em um arquivo
# ou banco de dados pode ser necessário encodificar ele.
>>> fruit.encode("utf-8")
b'\xf0\x9f\x8d\x89'
```

Esse valor `b'\xf0\x9f\x8d\x89'` é um objeto do tipo `bytes` e repare que ele tem 4 elementos separados por `\` cada um deles é um dos bytes que formam a 🍉

A operação contrária, por exemplo quando lermos de um arquivo ou banco de dados que não suporta `utf8` será com o `decode`.

```
melancia_em_bytes = b'\xf0\x9f\x8d\x89'
>>>  melancia_em_bytes.decode("utf-8")
'🍉'
```

O objeto ali iniciado por `b''` é uma sequencia de bytes em formato hexadecimal a titulo de curiosidade

f0 = 11110000
9f = 10011111
8d = 10001101
89 = 10001001

Que são os 4 bytes que formam o carecte 🍉 e você pode verificar isso no Python com cada um dos valores da lista:

```
>>> hex(0b11110000)
'0xf0'
```

Em Python números começados com `0b` são binários e `0x` são hexadecimais.

Strings, ou cadeia de caracteres

Até aqui falamos de caracteres isolados como `A`, `B`, 🍉 mas ao programar também precisaremos juntar esses carecteres para formar palavras e frases, quando criamos uma variável do tipo texto em Python ele através da presença de aspas sejam elas simples `'` ou duplas `"` armazena esse valor em uma classe do tipo `str` e este tipo de dado pode armazenar um ou mais caracteres.

```
>>> nome = "Bruno"
type(nome)
```

E como você já deve ter imaginado aqui estamos armazenando cada uma das letras `B`, `r`, `u`, `n`, `o` com seus respectivos bytes e sequencia posicional em um único objeto. (a plavra string significa corda, cadeia ou corrente),

A palavra `"Bruno"` é uma lista contendo em cada posição um caractere da tabela `utf8`.

```
>>> list(bytes(nome, "utf-8"))
[66, 114, 117, 110, 111]

>>> chr(66)
'B'

>>> chr(114)
'r'

>>> chr(117)
'u'

>>> chr(110)
'n'

>>> chr(111)
'o'
```

Bem, para guardar o nome "Bruno" você mais uma vez não precisa se procupar com esses detalhes todos, basta fazer `nome = "Bruno"` e usar este texto para efetuar as operações que você desejar, porém é muito útil saber como o objeto está implementado pois isso te permite efetuar operações como a que fizemos em nosso script `hello.py`

```
current_language = os.getenv("LANG", "en_US")[:5]
```

Sabendo que `current_language` poderia ter o valor `en_US.utf8` nós usamos o protocolo `Sliceable` do objeto `str` para **fatiar** o texto e pegar somente os primeiros 5 caracteres.

```
>>> "en_US.utf8"[:5]
'en_US'

>>> "Bruno"[2]
'u'

>>> "Python"[0]
'P'
```

O tipo `str` possui a maioria das carecteristicas que já abordamos nos outros tipos de dados e uma grande quantidade de protocolos implementados, vamos ver alguns.

```
# Sliceable (pode ser fatiado)
>>> "Bruno"[1]
'r'
# que internamente invoca o método `__getitem__`
>>> "Bruno".__getitem__(1)
'r'

# Addible (pode ser adicionado a outro texto)
# Essa operação se chama "Concatenação"
>>> nome = Bruno"
>>> sobrenome = "Rocha"
```

```
>>> nome + " " + sobrenome
'Bruno Rocha'
# que internamente invoca o método `__add__`
>>> nome.__add__(" ".__add__(sobrenome))
'Bruno Rocha'

# Multipliable (que pode ser multiplicado)
>>> "Bruno" * 5
'BrunoBrunoBrunoBrunoBruno'

# Iterable (que pode ser iterado/percorrido)
>>> for letra in "Bruno":
...     print("-->" + letra.upper())
-->B
-->R
-->U
-->N
-->O
# Internamente o statement `for` invoca o método `__iter__`
>>> iterador = "Bruno".__iter__()
>>> next(iterador)
'B'
>>> next(iterador)
'r'
```

Além disso tudo, o tipo `str` também oferece muitos métodos públicos, que nós podemos usar explicitamente e que são muito úteis.

```
>>> "Bruno".upper()
'BRUNO'

>>> "BRUNO".lower()
'bruno'

>>> "bruno rocha".capitalize()
'Bruno rocha'

>>> "bruno rocha".title()
'Bruno Rocha'

>>> "bruno rocha".split(" ")
['bruno', 'rocha']

>>> "bruno".startswith("b")
True

>>> "bruno".endswith("b")
False

>>> "bruno rocha".count("o")
2

>>> "bruno rocha".index("c")
8
```

```
>>> "bruno rocha"[8]
'c'
```

E também algumas coisas que podemos fazer com qualquer objeto sequencial do Python:

```
>>> len("Bruno Rocha")
11

>>> sorted("Bruno Rocha")
[' ', 'B', 'R', 'a', 'c', 'h', 'n', 'o', 'o', 'r', 'u']

>>> list(reversed("Bruno Rocha"))
['a', 'h', 'c', 'o', 'R', ' ', 'o', 'n', 'u', 'r', 'B']
```