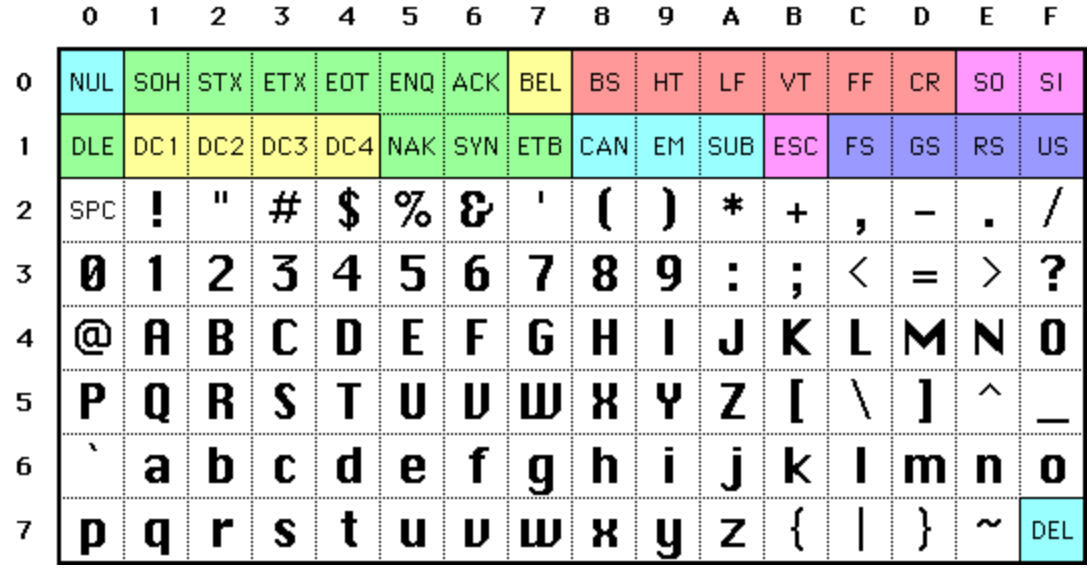
Строки, кодировки. Ввод-вывод, работа с файлами

Кодировки, байты, файлы

Этот слайд специально оставлен пустым.

Кодировки. ASCII

7 бит = один символ, т.е. каждому символу соответствует одно из значений от 0 до 127



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

ASCII появился в конце 70-х, большинство компьютеров уже использовало 8-битный байт, поэтому значения на самом деле от 0 до 255.

Каким символам соответствовали значения > 127?

Кодировки. ASCII

Стандарт ANSI описывал только символы меньше 128.

Как обрабатывать символы больше 128 описывалось кодовой страницей, зависящей от места, где вы проживаете.

Например, Израильская версия DOS использовала страницу с названием 862, а греческая — 737.

Получить при этом иврит и греческий язык на одном компьютере было совершенно невозможно, если вы не написали свою собственную программу, которая отображала бы все с использованием растровой графики, потому что для иврита и греческого языка требовались разные кодовые страницы.

Кодировки. Unicode

Unicode — не кодировка.

Unicode — это стандарт, сопоставляющей каждому символу или букве код в виде неотрицательного целого числа (code point), которое обычно представляется в шестнадцатеричной форме.

Область с кодами от `U+0000` до `U+007F` соответствует ASCII.

Символы unicode доступны здесь: [unicode]

Unicode 1.0 / UCS-2

1 символ = 2 байта

Особенности:

- легко найти n-ый символ
- 2 байта на любой символ :(

Unicode 2.0 / UTF-16

Каждый символ кодируется двумя или четырьмя байтами.

Особенности:

- расширяет UCS-2
- нельзя найти n-ый символ или
- минимум 2 байта даже для английских букв

UTF-8

ASCII-символы кодируются одним байтом. *Почему это хорошо?*

Все символы с кодом больше `U+007F` кодируются переменным числом байт, вплоть до 6.

Де-факто является стандартом.

Ввод-вывод

`input` — считать строку из стандартного потока ввода. Символы перевода строки будут обрезаны.

`print` — (по-умолчанию) вывести значение в стандартный поток вывода

```
def hello_world():
    hello = input("Say hello to the world: ")
    return f"{hello} world"


print('<-','-' * 5, hello_world(), '-' * 5, '->',  sep='|',
end='\nc:\n') #?
```

Перенаправление потоков

Дежурное напоминание, что интерпретатор, исполняющий скрипт — это полноценный процесс со своим набором файловых дескрипторов.

```
$ echo 'Hi' > goodbye_file
$ python script.py < goodbye_file
Say hello to the world: <-|-----|Hi world|-----|->
c:

$ echo 'Hi' | python script.py
Say hello to the world: <-|-----|Hi world|-----|->
c:
```

Работа с файлами.

Открыть файл можно с помощью `open`:

```
open(file, mode='r', buffering=-1, encoding=None,
errors=None, newline=None, closefd=True, opener=None)
```

- `file` — путь к файлу,
- `mode` — режим, в котором нужно открыть файл (*см. след. слайд*),
- `encoding` — кодировка

Работа с файлами. Режимы

- 'r' — read (default),
- 'w' — write,
- 'a' — append,
- 'r+' — read and write,
- 'a+' — read and append;

Работа с файлами. Режимы

И есть два дополнительных режима:

- 'b' — binary mode,
- 't' — text mode (default)

binary mode нужно использовать, чтобы читать/писать файл как последовательность байтов.

text mode, если нужно декодировать текст из байтов и работать с текстом.

Файлы, которые вернет `open` будут оперировать с байтовыми объектами или строками соответственно.

Работа с файлами

```
f.read() # считать весь файл
f.read(size) # считать не более size символов/байт

f.readline() # считать строку из файла

f.write('This is a test\n') # записать строку в файл


# для работы с бинарными файлами
f.tell() # текущая позиция
f.seek(5) # перейти на 5 байт вперед от текущей позиции
```

Работа с файлами.

После работы с файлом его обязательно нужно закрывать!

```
f = open('file', 'wt')
print(type(f))
# _io.TextIOWrapper

f.write('Ahahaha\n')
f.close()

f = open('file', 'r')
print(f.read())
# Ahahaha
f.close()
```

Работа с файлами. with

with-блок позволяет автоматически открывать/закрывать ресурсы (и не только) перед/после выполнения кода, в том числе закрывать используемые файлы.

Настоятельно рекомендуется использовать `with` всегда при работе с файлами.

```
with open('file', r) as file:
    # по файлу тоже можно итерироваться
    for line in file:
        print(line)
# не нужно закрывать файл, with сам его закроет
```

Дескрипторы

`open` также может принимать номер дескриптора, а не имя файла.

```
def print_from_descriptor(fd):
    with open(fd) as file:
        for line in file:
            print(line, end='')

print_from_descriptor(4)
```

Дескрипторы

запустим скрипт с предыдущего слайда:

```
$ printf "1\n2\n\3n" > file
$ python script.sh 4<file
1
2
3
```

Дескрипторы

Для стандартных потоков файловые объекты уже есть, в модуле sys:

- sys.stdin,
- sys.stdout,
- sys.stderr.

`input` и `print` используют `sys.stdin` и `sys.stdout`.

Все ошибки интерпретатор пишет в `sys.stderr`.

Дескрипторы

Будьте осторожны и не используйте `with` со стандартными потоками:

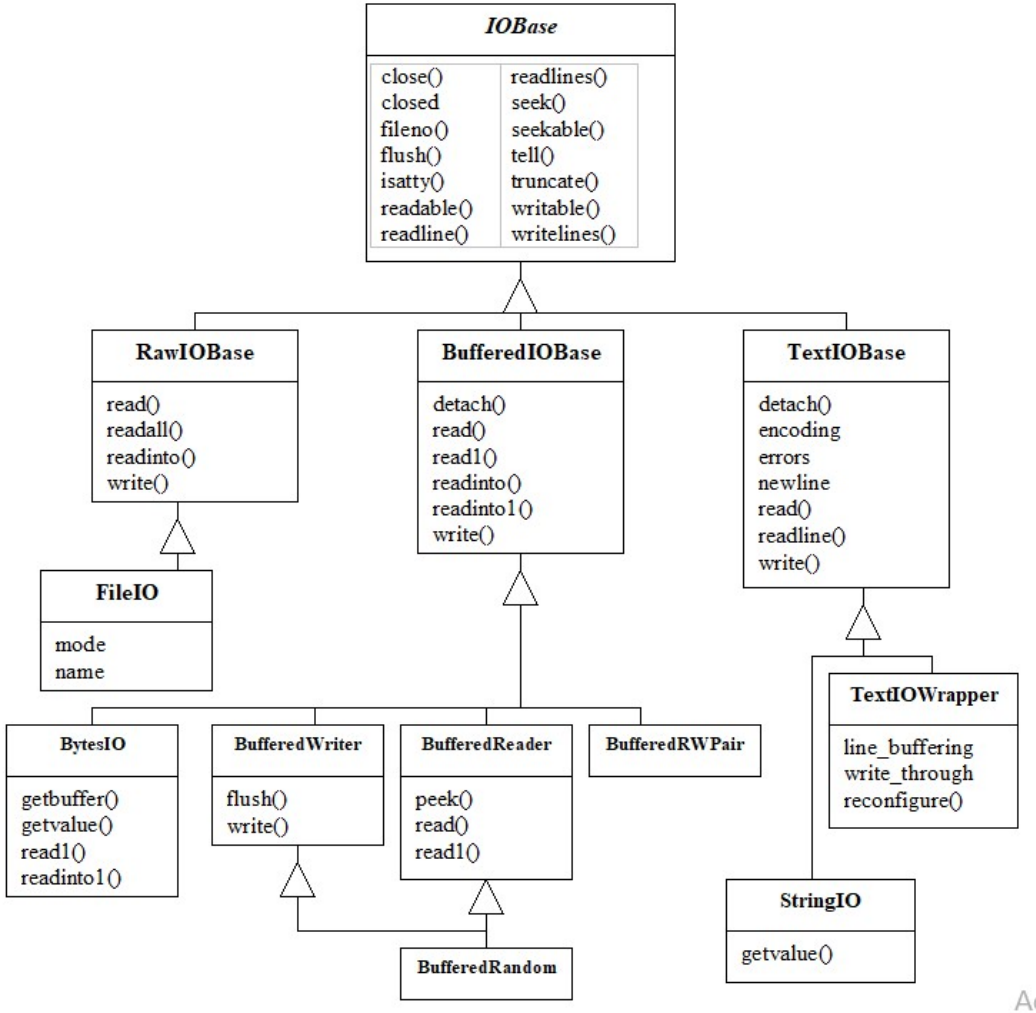
```
import sys

with sys.stdin as file:
     file.read()

print(input())
# Traceback (most recent call last):
#     print(input()) #?
# ValueError: I/O operation on closed file.
```

io

На самом деле файл — это какой-то из наследников IOBase



io

```
>>> c = open('tmp', 'wb')
>>> f = open('tmp', 'rb')
>>> e = open('tmp', 'wb', buffering=0)
>>> g = open('tmp', 'w')
>>> c, f, e, g
(<_io.BufferedWriter name='tmp'>,
<_io.BufferedReader name='tmp'>,
<_io.FileIO name='tmp' mode='wb' closefd=True>,
<_io.TextIOWrapper name='tmp' mode='w' encoding='UTF-8'>)
```