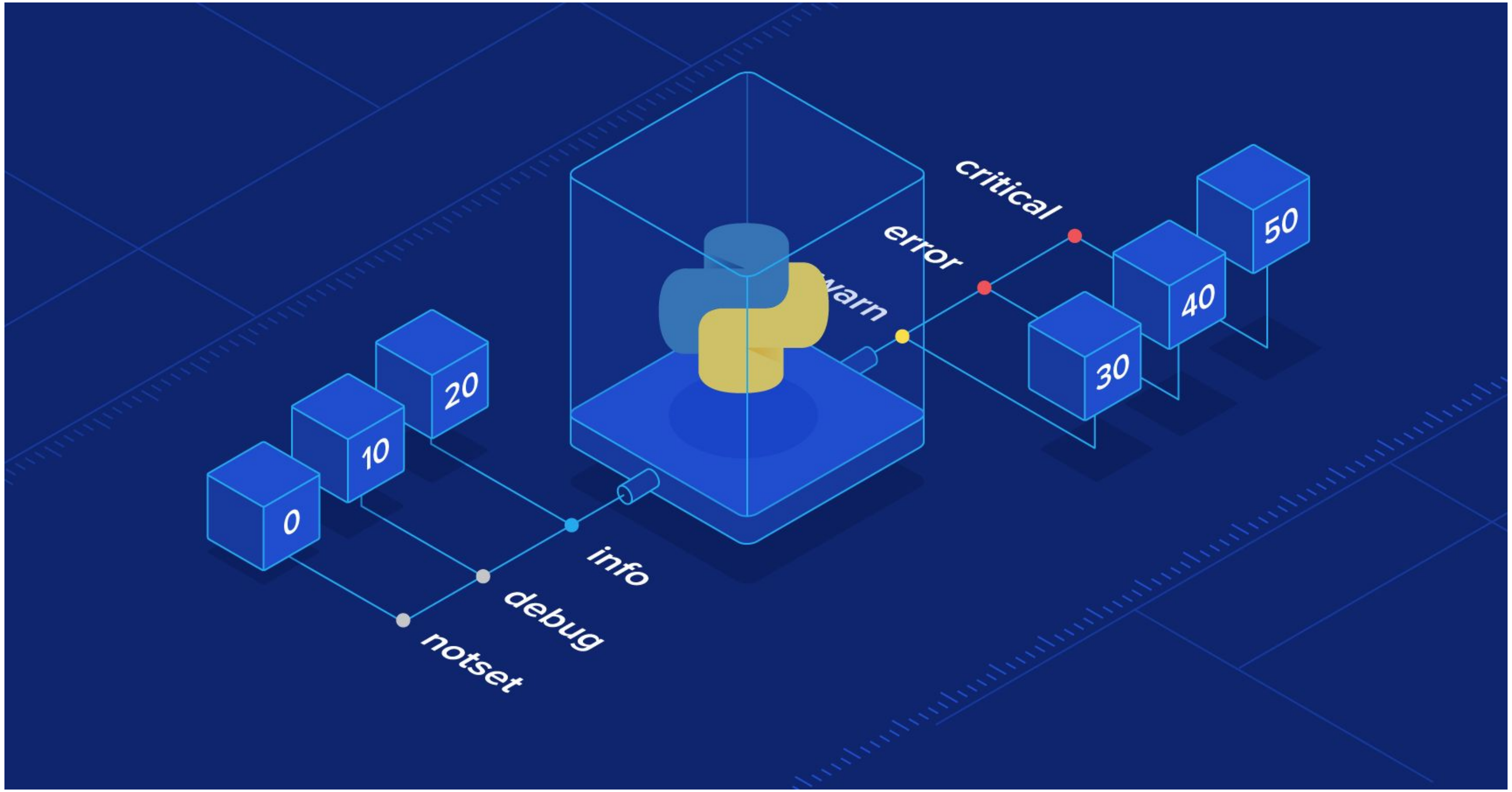
logging



What is it?

Well, all of us probably know that logging is the way to get information about program functioning

It is similar to print statements in code, but much better because of

- writing to specified location
- tweakability

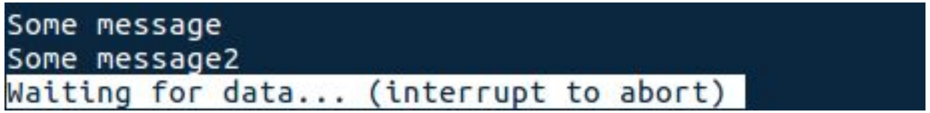
Why is it important?

- messages from program to display are ok when you testing small program, but it is difficult to work with them when your program is complex
- after debugging such prints should be removed - it takes time; and what if you will need to debug again?
- communication with users in case of program errors can be really painful if they have to provide you with all printed messages

Offtopic

First of all, small reminder about simple way to follow logs in real time

```
less +F your.log
```



```
Some message
Some message2
Waiting for data... (interrupt to abort)
```

Ctrl + C - stop following (you are just in less after this)
Shift + F - continue following

Logging libraries

There are several variants

- logging - built-in library
- loguru - popular alternative
- logbook - another one

Start logging

```
import logging


# To turn logging on     - change logging level to DEBUG
# To turn logging off    - change logging level to something
 higher, e.g. INFO
logging.basicConfig(filename='graph.log',
                    filemode='w',
                    format='%(message)s',
                    level=logging.DEBUG)
logger = logging.getLogger(__name__)
```

logging.basicConfig parameters

```
logging.basicConfig(filename, filemode, format, datefmt, level)
```

- `filename` - name of file where log will be stored
- `filemode` - mode of opening specified file (write or append)
- `format` - string denoting log records' format
- `datefmt` - string denoting log records' time format
- `level` - level of logging (soon about it)

Logging levels

From most important to less important

- CRITICAL
- ERROR
- WARNING
- INFO
- DEBUG
- NOTSET

These levels are used to specify whether each of your messages will be printed

```python
logger.info('Starting work...')
summa = 0
for i in range(10):
    summa += i
    logger.debug('i is %s', i)
    logger.debug('summa is %s', summa)

print(summa)
logger.info("Job's done!")
```

```
Starting work...
i is 0
summa is 0
i is 1
summa is 1
i is 2
summa is 3

...

i is 9
summa is 45
Job's done!
```

Now let's set logging level to INFO at the beginning of the script, other parts stay the same

```python
import logging

# To turn logging on    - change logging level to DEBUG
# To turn logging off   - change logging level to something
 higher, e.g. INFO
logging.basicConfig(filename='graph.log',
                    filemode='w',
                    format='%(message)s',
                    level=logging.INFO)
logger = logging.getLogger(__name__)
```

```python
logger.info('Starting work...')
summa = 0
for i in range(10):
    summa += i
    logger.debug('i is %s', i)
    logger.debug('summa is %s', summa)

print(summa)
logger.info("Job's done!")
```

```
Starting work...
Job's done!
```

That's all - only messages with logging level info or higher get written

Time formatting

Here is an example of adding time to your logs

```
# Turn logging on
logging.basicConfig(filename='my_log.log',
                    filemode='w',
                    format='%(asctime)s: %(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S',
                    level=logging.DEBUG)
logger = logging.getLogger(__name__)
```

```python
logger.info('Starting work...')
summa = 0
for i in range(10):
    summa += i
    logger.debug('i is %s', i)
    logger.debug('summa is %s', summa)

print(summa)
logger.info("Job's done!")
```

```
2020-04-17 00:14:59: Starting work...
2020-04-17 00:14:59: i is 0
2020-04-17 00:14:59: summa is 0
2020-04-17 00:14:59: i is 1
2020-04-17 00:14:59: summa is 1
2020-04-17 00:14:59: i is 2

...

2020-04-17 00:14:59: i is 7
2020-04-17 00:14:59: summa is 28
2020-04-17 00:14:59: i is 8
2020-04-17 00:14:59: summa is 36
2020-04-17 00:14:59: i is 9
2020-04-17 00:14:59: summa is 45
2020-04-17 00:14:59: Job's done!
```

Some format parts

These options are available inside `logging.basicConfig format` argument

- `%(message)s` - for passed message
- `%(asctime)s` - for log record time
- `%(levelname)s` - for level of log record
- `%(funcName)s` - for name of function from which log record was created
- `%(pathname)s` - for path to file from which log record was created