Dates, Documentation

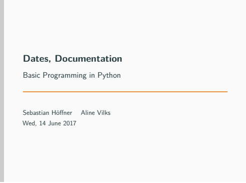
Basic Programming in Python

Sebastian Höffner Aline Vilks

Wed, 14 June 2017

2017-06-14

Dates, Documentation



Dates, Documentation

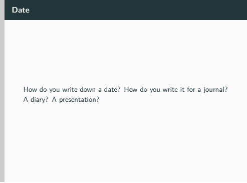
Basic Programming in Python

Sebastian Höffner Aline Vilks

Wed, 14 June 2017

Date

How do you write down a date? How do you write it for a journal? A diary? A presentation?


Date

How do you write down a date? How do you write it for a journal? A diary? A presentation?

Some date examples

(roughly “now”, give or take a few minutes)

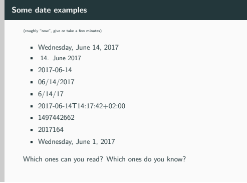
- Wednesday, June 14, 2017
- 14. June 2017
- 2017-06-14
- 06/14/2017
- 6/14/17
- 2017-06-14T14:17:42+02:00
- 1497442662
- 2017164
- Wednesday, June 1, 2017

Which ones can you read? Which ones do you know?

2017-06-14

Dates, Documentation

└─Some date examples


Some date examples

(roughly “now”, give or take a few minutes)

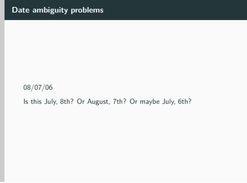
- Wednesday, June 14, 2017
- 14. June 2017
- 2017-06-14
- 06/14/2017
- 6/14/17
- 2017-06-14T14:17:42+02:00
- 1497442662
- 2017164
- Wednesday, June 1, 2017

Which ones can you read? Which ones do you know?

Date ambiguity problems

08/07/06

Is this July, 8th? Or August, 7th? Or maybe July, 6th?


Date ambiguity problems

08/07/06

Is this July, 8th? Or August, 7th? Or maybe July, 6th?

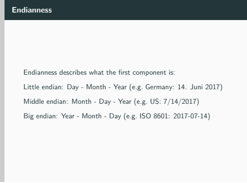
Endianness

Endianness describes what the first component is:

Little endian: Day - Month - Year (e.g. Germany: 14. Juni 2017)

Middle endian: Month - Day - Year (e.g. US: 7/14/2017)

Big endian: Year - Month - Day (e.g. ISO 8601: 2017-07-14)


Endianness

Endianness describes what the first component is:

Little endian: Day - Month - Year (e.g. Germany: 14. Juni 2017)

Middle endian: Month - Day - Year (e.g. US: 7/14/2017)

Big endian: Year - Month - Day (e.g. ISO 8601: 2017-07-14)

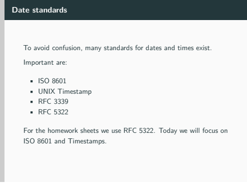
Date standards

To avoid confusion, many standards for dates and times exist.

Important are:

- ISO 8601
- UNIX Timestamp
- RFC 3339
- RFC 5322

For the homework sheets we use RFC 5322. Today we will focus on ISO 8601 and Timestamps.


Date standards

To avoid confusion, many standards for dates and times exist.

Important are:

- ISO 8601
- UNIX Timestamp
- RFC 3339
- RFC 5322

For the homework sheets we use RFC 5322. Today we will focus on ISO 8601 and Timestamps.

Date standards

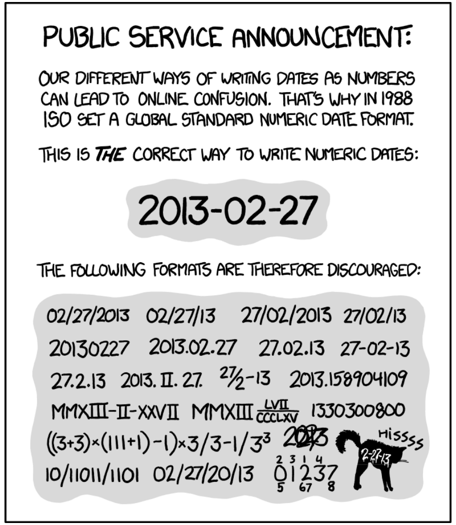

PUBLIC SERVICE ANNOUNCEMENT:
OUR DIFFERENT WAYS OF WRITING DATES AS NUMBERS CAN LEAD TO ONLINE CONFUSION. THAT'S WHY IN 1988 ISO SET A GLOBAL STANDARD NUMERIC DATE FORMAT.
THIS IS THE CORRECT WAY TO WRITE NUMERIC DATES:
2013-02-27
THE FOLLOWING FORMATS ARE THEREFORE DISCOURAGED:
02/27/2013 02/27/13 27/02/2013 27/02/13
20130227 2013.02.27 27.02.13 27-02-13
27.2.13 2013. II. 27. 27/2-13 2013.158904109
MMXIII-II-XXVII MMXIII LVII/CCCLXV 1330300800
((3+3)×(111+1)-1)×3/3-1/3³ 2013
HISSSSS
2-27-13
10/11011/1101 02/27/20/13 01237

Figure 1: ISO 8601 was published on 06/05/88 and most recently amended on 12/01/04. (Munroe 2013)

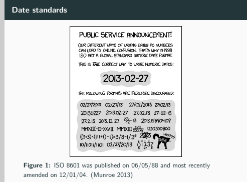
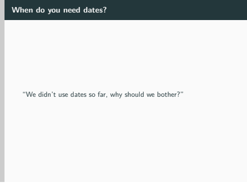

Date standards

Figure 1: ISO 8601 was published on 06/05/88 and most recently amended on 12/01/04. (Munroe 2013)

When do you need dates?

"We didn't use dates so far, why should we bother?"

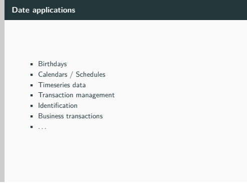


When do you need dates?

"We didn't use dates so far, why should we bother?"

Date applications

- Birthdays
- Calendars / Schedules
- Timeseries data
- Transaction management
- Identification
- Business transactions
- ...



Date applications

- Birthdays
- Calendars / Schedules
- Timeseries data
- Transaction management
- Identification
- Business transactions
- ...

Dates in Python

```python
import datetime

today = datetime.date.today()
print(today)
print(repr(today))
now = datetime.datetime.now()
print(now)
print(repr(now))
```

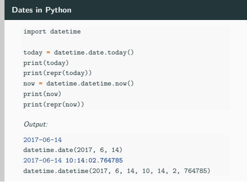
Output:

```
2017-06-14
datetime.date(2017, 6, 14)
2017-06-14 10:14:02.764785
datetime.datetime(2017, 6, 14, 10, 14, 2, 764785)
```

2017-06-14

Dates, Documentation

└─Dates in Python


Dates in Python

```python
import datetime

today = datetime.date.today()
print(today)
print(repr(today))
now = datetime.datetime.now()
print(now)
print(repr(now))
```

Output:

```
2017-06-14
datetime.date(2017, 6, 14)
2017-06-14 10:14:02.764785
datetime.datetime(2017, 6, 14, 10, 14, 2, 764785)
```

Specific date

```python
from datetime import date


bday = date(1991, 8, 21)
print(bday)
```

Output:

```
1991-08-21
```

2017-06-14

Dates, Documentation

└Specific date

Specific date

```python
from datetime import date

bday = date(1991, 8, 21)
print(bday)
```

Output:

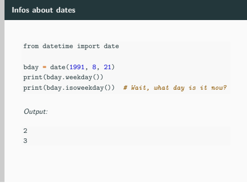
```
1991-08-21
```

Infos about dates

```python
from datetime import date

bday = date(1991, 8, 21)
print(bday.weekday())
print(bday.isoweekday())  # Wait, what day is it now?
```

Output:

```
2
3
```


Infos about dates

```python
from datetime import date

bday = date(1991, 8, 21)
print(bday.weekday())
print(bday.isoweekday())  # Wait, what day is it now?
```

Output:

```
2
3
```

`weekday()` starts with Monday as 0, the ISO standard (`isoweekday()`) with Monday as 1. So this is Wednesday.

Formatting outputs

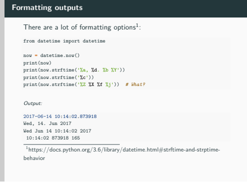
There are a lot of formatting options[1]:

```
from datetime import datetime

now = datetime.now()
print(now)
print(now.strftime('%a, %d. %b %Y'))
print(now.strftime('%c'))
print(now.strftime('%Z %X %f %j'))  # What?
```

Output:

```
2017-06-14 10:14:02.873918
Wed, 14. Jun 2017
Wed Jun 14 10:14:02 2017
 10:14:02 873918 165
```

[1]https://docs.python.org/3.6/library/datetime.html#strftime-and-strptime-behavior


Formatting outputs

There are a lot of formatting options[1]:

```
from datetime import datetime

now = datetime.now()
print(now)
print(now.strftime('%a, %d. %b %Y'))
print(now.strftime('%c'))
print(now.strftime('%Z %X %f %j'))  # What?
```

Output:

```
2017-06-14 10:14:02.873918
Wed, 14. Jun 2017
Wed Jun 14 10:14:02 2017
 10:14:02 873918 165
```

[1]https://docs.python.org/3.6/library/datetime.html#strftime-and-strptime-behavior

`strftime` can be remembered as "**str**ing **f**ormat of **time**".

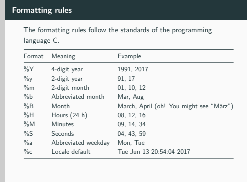
The weird ones are:

- %Z: Timezone. Not present here.
- %X: The current time.
- %f: The current milliseconds.
- %j: The current day of the year.

Formatting rules

The formatting rules follow the standards of the programming language C.

Format	Meaning	Example
%Y	4-digit year	1991, 2017
%y	2-digit year	91, 17
%m	2-digit month	01, 10, 12
%b	Abbreviated month	Mar, Aug
%B	Month	March, April (oh! You might see “März”)
%H	Hours (24 h)	08, 12, 16
%M	Minutes	09, 14, 34
%S	Seconds	04, 43, 59
%a	Abbreviated weekday	Mon, Tue
%c	Locale default	Tue Jun 13 20:54:04 2017



Formatting rules

The formatting rules follow the standards of the programming language C.

Format	Meaning	Example
%Y	4-digit year	1991, 2017
%y	2-digit year	91, 17
%m	2-digit month	01, 10, 12
%b	Abbreviated month	Mar, Aug
%B	Month	March, April (oh! You might see “März”)
%H	Hours (24 h)	08, 12, 16
%M	Minutes	09, 14, 34
%S	Seconds	04, 43, 59
%a	Abbreviated weekday	Mon, Tue
%c	Locale default	Tue Jun 13 20:54:04 2017

This list is not exhaustive, it just contains some important ones.

Locale can be roughly seen as you computers language and location settings.

Formatting rules example: Locale

Try it out!

```
from datetime import datetime

print(datetime.now().strftime('%c'))
```

Output:

```
Wed Jun 14 10:14:02 2017
```

Use `strftime(...)` to create the same output as %c did here.
(You can try your own at home, if it differs)

Formatting rules example: Locale

Try it out!

```
from datetime import datetime

print(datetime.now().strftime('%c'))
```

Output:

```
Wed Jun 14 10:14:02 2017
```

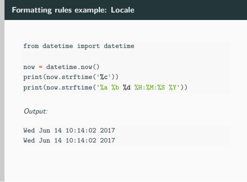
Use `strftime(...)` to create the same output as %c did here.
(You can try your own at home, if it differs)

Formatting rules example: Locale

```python
from datetime import datetime

now = datetime.now()
print(now.strftime('%c'))
print(now.strftime('%a %b %d %H:%M:%S %Y'))
```

Output:

```
Wed Jun 14 10:14:02 2017
Wed Jun 14 10:14:02 2017
```



Formatting rules example: Locale

```python
from datetime import datetime

now = datetime.now()
print(now.strftime('%c'))
print(now.strftime('%a %b %d %H:%M:%S %Y'))
```

Output:

```
Wed Jun 14 10:14:02 2017
Wed Jun 14 10:14:02 2017
```

Formatting rules example: Locale

```python
from datetime import datetime
import locale

locale.setlocale(locale.LC_ALL, 'de_DE')

now = datetime.now()
print(now.strftime('%c'))
print(now.strftime('%a %b %d %H:%M:%S %Y'))
```

Output:

```
Mi 14 Jun 10:14:02 2017
Mi Jun 14 10:14:02 2017
```

Formatting rules example: Locale

```python
from datetime import datetime
import locale

locale.setlocale(locale.LC_ALL, 'de_DE')

now = datetime.now()
print(now.strftime('%c'))
print(now.strftime('%a %b %d %H:%M:%S %Y'))
```

Output:

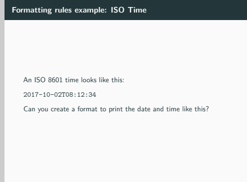
```
Mi 14 Jun 10:14:02 2017
Mi Jun 14 10:14:02 2017
```

Formatting rules example: ISO Time

An ISO 8601 time looks like this:

```
2017-10-02T08:12:34
```

Can you create a format to print the date and time like this?



Formatting rules example: ISO Time

An ISO 8601 time looks like this:

2017-10-02T08:12:34

Can you create a format to print the date and time like this?

Formatting rules example: ISO Time

```
from datetime import datetime

print(datetime.now().strftime('%Y-%m-%dT%H:%M:%S'))
```

Output:

```
2017-06-14T10:14:03
```

Formatting rules example: ISO Time

```
from datetime import datetime

print(datetime.now().strftime('%Y-%m-%dT%H:%M:%S'))
```

Output:

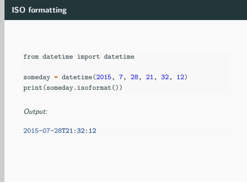
```
2017-06-14T10:14:03
```

ISO formatting

```python
from datetime import datetime

someday = datetime(2015, 7, 28, 21, 32, 12)
print(someday.isoformat())
```

Output:

```
2015-07-28T21:32:12
```



ISO formatting

```python
from datetime import datetime

someday = datetime(2015, 7, 28, 21, 32, 12)
print(someday.isoformat())
```

Output:

```
2015-07-28T21:32:12
```

Switching sides – date parsing

Last week's homework discussed string parsing. For dates we can do the same:

```python
from datetime import datetime

parsed = datetime.strptime('Wed Jun 14 14:47:12 2017',
                           '%a %b %d %H:%M:%S %Y')

print(parsed.isoformat())
```

Output:

```
2017-06-14T14:47:12
```

Switching sides – date parsing

Last week's homework discussed string parsing. For dates we can do the same:

```python
from datetime import datetime

parsed = datetime.strptime('Wed Jun 14 14:47:12 2017',
                           '%a %b %d %H:%M:%S %Y')

print(parsed.isoformat())
```

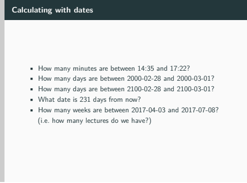
Output:

```
2017-06-14T14:47:12
```

Analogue to `strftime`, `strptime` stands for **str**ing **p**arse **time**.

Calculating with dates

- How many minutes are between 14:35 and 17:22?
- How many days are between 2000-02-28 and 2000-03-01?
- How many days are between 2100-02-28 and 2100-03-01?
- What date is 231 days from now?
- How many weeks are between 2017-04-03 and 2017-07-08? (i.e. how many lectures do we have?)


Calculating with dates

- How many minutes are between 14:35 and 17:22?
- How many days are between 2000-02-28 and 2000-03-01?
- How many days are between 2100-02-28 and 2100-03-01?
- What date is 231 days from now?
- How many weeks are between 2017-04-03 and 2017-07-08? (i.e. how many lectures do we have?)

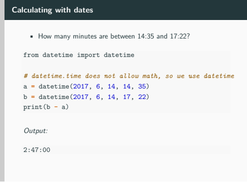
Calculating with dates

- How many minutes are between 14:35 and 17:22?

```python
from datetime import datetime

# datetime.time does not allow math, so we use datetime
a = datetime(2017, 6, 14, 14, 35)
b = datetime(2017, 6, 14, 17, 22)
print(b - a)
```

Output:

```
2:47:00
```



Calculating with dates

- How many minutes are between 14:35 and 17:22?

```python
from datetime import datetime

# datetime.time does not allow math, so we use datetime
a = datetime(2017, 6, 14, 14, 35)
b = datetime(2017, 6, 14, 17, 22)
print(b - a)
```

Output:

```
2:47:00
```

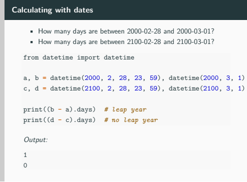
Calculating with dates

- How many days are between 2000-02-28 and 2000-03-01?
- How many days are between 2100-02-28 and 2100-03-01?

```
from datetime import datetime

a, b = datetime(2000, 2, 28, 23, 59), datetime(2000, 3, 1)
c, d = datetime(2100, 2, 28, 23, 59), datetime(2100, 3, 1)

print((b - a).days)  # leap year
print((d - c).days)  # no leap year
```

Output:

```
1
0
```


Calculating with dates

- How many days are between 2000-02-28 and 2000-03-01?
- How many days are between 2100-02-28 and 2100-03-01?

```
from datetime import datetime

a, b = datetime(2000, 2, 28, 23, 59), datetime(2000, 3, 1)
c, d = datetime(2100, 2, 28, 23, 59), datetime(2100, 3, 1)

print((b - a).days)  # leap year
print((d - c).days)  # no leap year
```

Output:

```
1
0
```

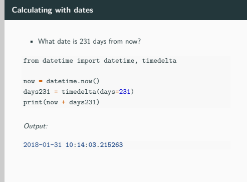
Calculating with dates

- What date is 231 days from now?

```
from datetime import datetime, timedelta

now = datetime.now()
days231 = timedelta(days=231)
print(now + days231)
```

Output:

```
2018-01-31 10:14:03.215263
```


Calculating with dates

- What date is 231 days from now?

```
from datetime import datetime, timedelta

now = datetime.now()
days231 = timedelta(days=231)
print(now + days231)
```

Output:

```
2018-01-31 10:14:03.215263
```

Calculating with dates

- How many weeks are between 2017-04-03 and 2017-07-08? (i.e. how many lectures do we have?)

```python
import math
from datetime import datetime, timedelta

begin = datetime(2017, 4, 3)
end = datetime(2017, 7, 8)

print(math.ceil((end - begin) / timedelta(weeks=1)))
```

Output:

```
14
```

Calculating with dates

- How many weeks are between 2017-04-03 and 2017-07-08? (i.e. how many lectures do we have?)

```python
import math
from datetime import datetime, timedelta

begin = datetime(2017, 4, 3)
end = datetime(2017, 7, 8)

print(math.ceil((end - begin) / timedelta(weeks=1)))
```

Output:

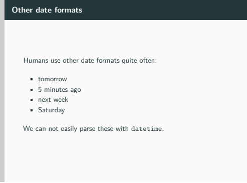
```
14
```

Other date formats

Humans use other date formats quite often:

- tomorrow
- 5 minutes ago
- next week
- Saturday

We can not easily parse these with `datetime`.



Other date formats

Humans use other date formats quite often:

- tomorrow
- 5 minutes ago
- next week
- Saturday

We can not easily parse these with `datetime`.

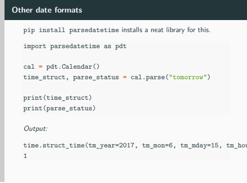
Other date formats

`pip install parsedatetime` installs a neat library for this.

```
import parsedatetime as pdt

cal = pdt.Calendar()
time_struct, parse_status = cal.parse("tomorrow")

print(time_struct)
print(parse_status)
```

Output:

```
time.struct_time(tm_year=2017, tm_mon=6, tm_mday=15, tm_hou
1
```



Other date formats

pip install parsedatetime installs a neat library for this.

```
import parsedatetime as pdt

cal = pdt.Calendar()
time_struct, parse_status = cal.parse("tomorrow")

print(time_struct)
print(parse_status)
```

Output:

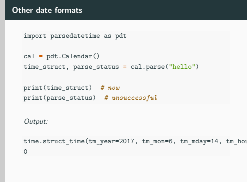
```
time.struct_time(tm_year=2017, tm_mon=6, tm_mday=15, tm_hou
1
```

Other date formats

```python
import parsedatetime as pdt

cal = pdt.Calendar()
time_struct, parse_status = cal.parse("hello")

print(time_struct)  # now
print(parse_status)  # unsuccessful
```

Output:

```
time.struct_time(tm_year=2017, tm_mon=6, tm_mday=14, tm_hou
0
```


Other date formats

```python
import parsedatetime as pdt

cal = pdt.Calendar()
time_struct, parse_status = cal.parse("hello")

print(time_struct)  # now
print(parse_status)  # unsuccessful
```

Output:

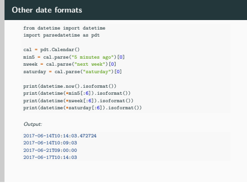
```
time.struct_time(tm_year=2017, tm_mon=6, tm_mday=14, tm_hou
0
```

Other date formats

```python
from datetime import datetime
import parsedatetime as pdt

cal = pdt.Calendar()
min5 = cal.parse("5 minutes ago")[0]
nweek = cal.parse("next week")[0]
saturday = cal.parse("saturday")[0]

print(datetime.now().isoformat())
print(datetime(*min5[:6]).isoformat())
print(datetime(*nweek[:6]).isoformat())
print(datetime(*saturday[:6]).isoformat())
```

Output:

```
2017-06-14T10:14:03.472724
2017-06-14T10:09:03
2017-06-21T09:00:00
2017-06-17T10:14:03
```


Other date formats

```python
from datetime import datetime
import parsedatetime as pdt

cal = pdt.Calendar()
min5 = cal.parse("5 minutes ago")[0]
nweek = cal.parse("next week")[0]
saturday = cal.parse("saturday")[0]

print(datetime.now().isoformat())
print(datetime(*min5[:6]).isoformat())
print(datetime(*nweek[:6]).isoformat())
print(datetime(*saturday[:6]).isoformat())
```

Output:

```
2017-06-14T10:14:03.472724
2017-06-14T10:09:03
2017-06-21T09:00:00
2017-06-17T10:14:03
```

Measuring times

In many cases we don't need full dates:

- Program execution times
- Download times
- Racing times
- . . .

Measuring times

In many cases we don't need full dates:

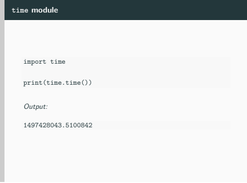
- Program execution times
- Download times
- Racing times
- . . .

`time` module

```
import time

print(time.time())
```

Output:

```
1497428043.5100842
```


time module

import time

print(time.time())

Output:

1497428043.5100842

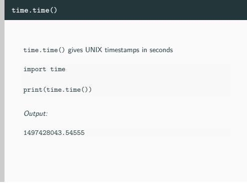
`time.time()`

`time.time()` gives UNIX timestamps in seconds

```
import time

print(time.time())
```

Output:

```
1497428043.54555
```



time.time()

time.time() gives UNIX timestamps in seconds

import time

print(time.time())

Output:

1497428043.54555

The seconds are exact, everything in between depends on the system. However, for most things that's enough.

UNIX timestamp

The UNIX time (or POSIX time) starts at

January 1st, 1970, 00:00:00 UTC

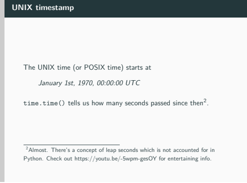
`time.time()` tells us how many seconds passed since then[2].

[2] Almost. There's a concept of leap seconds which is not accounted for in Python. Check out https://youtu.be/-5wpm-gesOY for entertaining info.

2017-06-14

Dates, Documentation

└UNIX timestamp



UNIX timestamp

The UNIX time (or POSIX time) starts at

January 1st, 1970, 00:00:00 UTC

`time.time()` tells us how many seconds passed since then[2].

[2] Almost. There's a concept of leap seconds which is not accounted for in Python. Check out https://youtu.be/-5wpm-gesOY for entertaining info.

Execution time

Most commonly we use `time.time()` to measure execution times.

```
import time

start = time.time()
time.sleep(.3)  # do something (here: nothing)
end = time.time()

print(end - start)
```

Output:

```
0.30513620376586914
```

Execution time

Most commonly we use time.time() to measure execution times.

```
import time

start = time.time()
time.sleep(.3)  # do something (here: nothing)
end = time.time()

print(end - start)
```

Output:

```
0.30513620376586914
```

Important applications are: download times, complex computations, simulations, computer games, . . .

`time.sleep(...)` lets your program sleep for roughly the number of seconds passed to it.

Benchmarking functions

```python
import timeit

print(timeit.timeit("123 + 456"))
```

Output:

```
0.013090435997582972
```

Benchmarking functions

```python
import timeit

print(timeit.timeit("123 + 456"))
```

Output:

```
0.013090435997582972
```

timeit runs your function multiple times and calculates some statistics about it.

This can help you figure out which functions are fast, which ones are slow, etc.

Benchmarking functions

```python
import timeit

def add(a, b):
    return a + b

print(timeit.timeit("add(123, 456)",
                    setup="from __main__ import add"))
```

Output:

```
0.28266442695166916
```

Benchmarking functions

```python
import timeit

def add(a, b):
    return a + b

print(timeit.timeit("add(123, 456)",
                    setup="from __main__ import add"))
```

Output:

```
0.28266442695166916
```

It requires a little bit more work to test your own functions: You need some *setup* to import them.

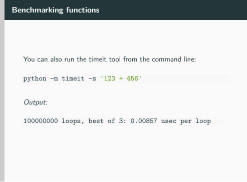
Benchmarking functions

You can also run the timeit tool from the command line:

```
python -m timeit -s '123 + 456'
```

Output:

```
100000000 loops, best of 3: 0.00857 usec per loop
```



Benchmarking functions

You can also run the timeit tool from the command line:

python -m timeit -s '123 + 456'

Output:

100000000 loops, best of 3: 0.00857 usec per loop

Measure the time 100,000,000 times (sometimes fewer, it makes assumptions about how many iterations are reasonable) and returns the average of the best three runs.

Enough of dates and times

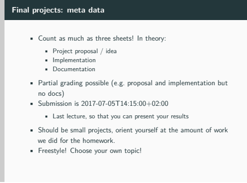
Let's talk a little bit about the final projects!

Enough of dates and times

Let's talk a little bit about the final projects!

Final projects: meta data

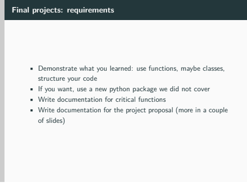
- Count as much as three sheets! In theory:
  - Project proposal / idea
  - Implementation
  - Documentation
- Partial grading possible (e.g. proposal and implementation but no docs)
- Submission is 2017-07-05T14:15:00+02:00
  - Last lecture, so that you can present your results
- Should be small projects, orient yourself at the amount of work we did for the homework.
- Freestyle! Choose your own topic!


Final projects: meta data

- Count as much as three sheets! In theory:
  - Project proposal / idea
  - Implementation
  - Documentation
- Partial grading possible (e.g. proposal and implementation but no docs)
- Submission is 2017-07-05T14:15:00+02:00
  - Last lecture, so that you can present your results
- Should be small projects, orient yourself at the amount of work we did for the homework.
- Freestyle! Choose your own topic!

Final projects: requirements

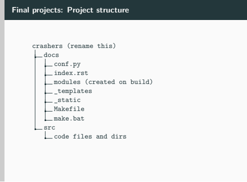
- Demonstrate what you learned: use functions, maybe classes, structure your code
- If you want, use a new python package we did not cover
- Write documentation for critical functions
- Write documentation for the project proposal (more in a couple of slides)


Final projects: requirements

- Demonstrate what you learned: use functions, maybe classes, structure your code
- If you want, use a new python package we did not cover
- Write documentation for critical functions
- Write documentation for the project proposal (more in a couple of slides)

Final projects: Project structure

```
crashers (rename this)
 └─ docs
     └─ conf.py
     └─ index.rst
     └─ modules (created on build)
     └─ _templates
     └─ _static
     └─ Makefile
     └─ make.bat
 └─ src
     └─ code files and dirs
```



Final projects: Project structure

```
crashers (rename this)
 └─ docs
     └─ conf.py
     └─ index.rst
     └─ modules (created on build)
     └─ _templates
     └─ _static
     └─ Makefile
     └─ make.bat
 └─ src
     └─ code files and dirs
```

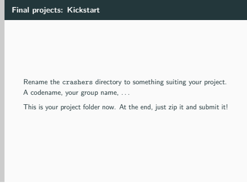
The `src` directory is the heart of your project. Here will all your modules, packages, etc. be.

The `docs` directory is reserved for the documentation. We will have to do some minor adjustments here.

Final projects: Kickstart

Rename the `crashers` directory to something suiting your project. A codename, your group name, . . .

This is your project folder now. At the end, just zip it and submit it!

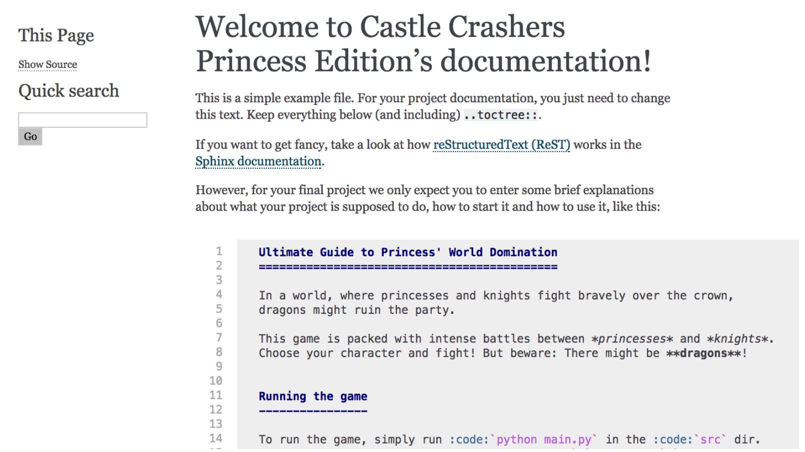


Final projects: Kickstart

Rename the `crashers` directory to something suiting your project. A codename, your group name, . . .

This is your project folder now. At the end, just zip it and submit it!

Final projects: Documentation


This Page

Show Source

Quick search

Go

Welcome to Castle Crashers Princess Edition's documentation!

This is a simple example file. For your project documentation, you just need to change this text. Keep everything below (and including) `..toctree::`.

If you want to get fancy, take a look at how reStructuredText (ReST) works in the Sphinx documentation.

However, for your final project we only expect you to enter some brief explanations about what your project is supposed to do, how to start it and how to use it, like this:

```
Ultimate Guide to Princess' World Domination
============================================

In a world, where princesses and knights fight bravely over the crown,
dragons might ruin the party.

This game is packed with intense battles between *princesses* and *knights*.
Choose your character and fight! But beware: There might be **dragons**!


Running the game
----------------

To run the game, simply run :code:`python main.py` in the :code:`src` dir.
```

Figure 2: Example docs

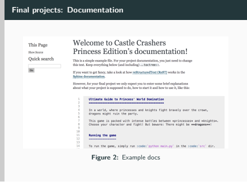

Final projects: Documentation

Figure 2: Example docs

We will use Sphinx for the documentation.

```
pip install sphinx
```

Final projects: Documentation

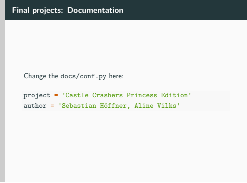
Change the `docs/conf.py` here:

```
project = 'Castle Crashers Princess Edition'
author = 'Sebastian Höffner, Aline Vilks'
```

2017-06-14

Dates, Documentation

└Final projects: Documentation



Final projects: Documentation

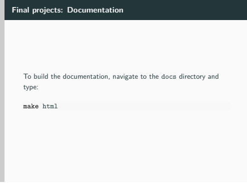
Change the docs/conf.py here:

```
project = 'Castle Crashers Princess Edition'
author = 'Sebastian Höffner, Aline Vilks'
```

Final projects: Documentation

To build the documentation, navigate to the `docs` directory and type:

```
make html
```



Final projects: Documentation

To build the documentation, navigate to the docs directory and type:

```
make html
```

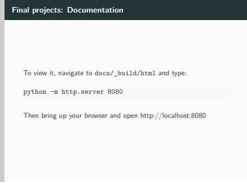
This may or may not work properly now. If you have any troubles you can't solve, talk to us!

Final projects: Documentation

To view it, navigate to `docs/_build/html` and type:

```
python -m http.server 8080
```

Then bring up your browser and open http://localhost:8080



Final projects: Documentation

To view it, navigate to docs/_build/html and type:

python -m http.server 8080

Then bring up your browser and open http://localhost:8080

This can differ for older Python versions. Come to us with any problems!

Final projects: Documentation

To change what you see, adjust the `index.rst` inside the `docs` directory. Then rebuild (`make html`) the documentation!

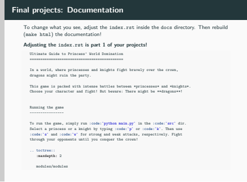
Adjusting the `index.rst` is part 1 of your projects!

```
Ultimate Guide to Princess' World Domination
============================================

In a world, where princesses and knights fight bravely over the crown,
dragons might ruin the party.

This game is packed with intense battles between *princesses* and *knights*.
Choose your character and fight! But beware: There might be **dragons**!


Running the game
----------------

To run the game, simply run :code:`python main.py` in the :code:`src` dir.
Select a princess or a knight by typing :code:`p` or :code:`k`. Then use
:code:`s` and :code:`w` for strong and weak attacks, respectively. Fight
through your opponents until you conquer the crown!

.. toctree::
   :maxdepth: 2

   modules/modules
```

2017-06-14

Dates, Documentation

└Final projects: Documentation


Final projects: Documentation

To change what you see, adjust the index.rst inside the docs directory. Then rebuild (make html) the documentation!

Adjusting the index.rst is part 1 of your projects!

Final projects: Documentation – ReST

```
Titles are underlined
=====================

**Bold fonts**
*italic fonts*

Subtitle
--------

:code:`inline code`

.. code-block:: python

   print('Hello World!')

This is `a link`_ in a sentence.

.. _a link: http://localhost:8080
```

Final projects: Documentation – ReST

```
Titles are underlined
=====================

**Bold fonts**
*italic fonts*

Subtitle
--------

:code:`inline code`

.. code-block:: python

   print('Hello World!')

This is `a link`_ in a sentence.

.. _a link: http://localhost:8080
```

There's much much more to ReST, but these are the most important things you will need.

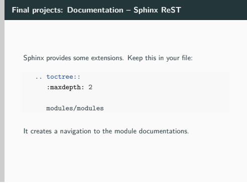
You can try out (some) things at http://rst.ninjs.org/ .

Final projects: Documentation – Sphinx ReST

Sphinx provides some extensions. Keep this in your file:

```
.. toctree::
   :maxdepth: 2

   modules/modules
```

It creates a navigation to the module documentations.


Final projects: Documentation – Sphinx ReST

Sphinx provides some extensions. Keep this in your file:

```
.. toctree::
   :maxdepth: 2

   modules/modules
```

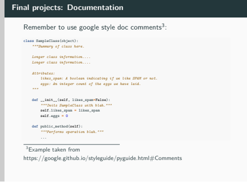
It creates a navigation to the module documentations.

Final projects: Documentation

Remember to use google style doc comments[3]:

```python
class SampleClass(object):
    """Summary of class here.

    Longer class information....
    Longer class information....

    Attributes:
        likes_spam: A boolean indicating if we like SPAM or not.
        eggs: An integer count of the eggs we have laid.
    """

    def __init__(self, likes_spam=False):
        """Inits SampleClass with blah."""
        self.likes_spam = likes_spam
        self.eggs = 0

    def public_method(self):
        """Performs operation blah."""
        ...
```

[3]Example taken from https://google.github.io/styleguide/pyguide.html#Comments


Final projects: Documentation

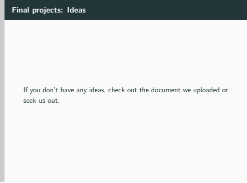
Remember to use google style doc comments[3]:

```python
class SampleClass(object):
    """Summary of class here.

    Longer class information....
    Longer class information....

    Attributes:
        likes_spam: A boolean indicating if we like SPAM or not.
        eggs: An integer count of the eggs we have laid.
    """

    def __init__(self, likes_spam=False):
        """Inits SampleClass with blah."""
        self.likes_spam = likes_spam
        self.eggs = 0

    def public_method(self):
        """Performs operation blah."""
        ...
```

[3]Example taken from https://google.github.io/styleguide/pyguide.html#Comments

Final projects: Ideas

If you don't have any ideas, check out the document we uploaded or seek us out.



Final projects: Ideas

If you don't have any ideas, check out the document we uploaded or seek us out.

Final projects: Main guideline

Have fun!

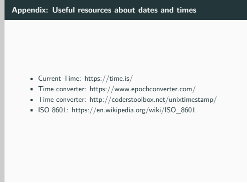
Final projects: Main guideline

Have fun!

Appendix: Useful resources about dates and times

- Current Time: https://time.is/
- Time converter: https://www.epochconverter.com/
- Time converter: http://coderstoolbox.net/unixtimestamp/
- ISO 8601: https://en.wikipedia.org/wiki/ISO_8601

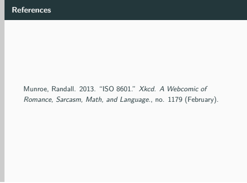
└─Appendix: Useful resources about dates and times


Appendix: Useful resources about dates and times

- Current Time: https://time.is/
- Time converter: https://www.epochconverter.com/
- Time converter: http://coderstoolbox.net/unixtimestamp/
- ISO 8601: https://en.wikipedia.org/wiki/ISO_8601

References

Munroe, Randall. 2013. “ISO 8601.” *Xkcd. A Webcomic of Romance, Sarcasm, Math, and Language.*, no. 1179 (February).



References

Munroe, Randall. 2013. “ISO 8601.” *Xkcd. A Webcomic of Romance, Sarcasm, Math, and Language.*, no. 1179 (February).