Introduction to

– with Application to Bioinformatics

NBIS

Gimme an example of

literal
variable
builtin type
mutable sequence
immutable sequence
iterable (but not list)
range
int
float
str

list
bool
open
encoding
conditional
for-loop
comment
in (membership)
comparator
builtin function (stdlib)

Types

Builtin types	Builtin functions
int, float, str bool, list, ...	min(), max(), type(), list(), int(), str(), id(), help(), open(),...

Loops

```
fruits = ['Apple','Orange']

for fruit in fruits:
    print(fruit)
```

```
x = 0
while x < 100:
    print(x)
    x += 1
```

Conditionals

```
if condition:
    print('This will be executed')
else:
    print('Otherwise, it is this one')
```

I/O Files

```
open('filename', 'r', encoding='utf-8')
```

I/O Files

```
open('filename', 'r', encoding='utf-8')
```

```
with open('filename', 'r', encoding='utf-8') as the_file:
    for line in the_file:      # the_file is iterable, yeii !
        print(line.rstrip())   # Removing the trailing \n
                               # since print() adds one.
```

Identity

Logical	Physical
==	is
!=	is not

Some Operations on Sequences

min()

max()

➥ Notebook 3

sum()

abs()

New type

str	tuple	list
" "	(, , , ,)	[, , , ,]
str( )	tuple( )	list( )

iterable

Some Operations on Strings

rstrip()

lstrip()

strip()

Some Operations on Strings

str.**lstrip**([*chars*])

Return a copy of the string with leading characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.lstrip()                                    >>>
'spacious   '
>>> 'www.example.com'.lstrip('cmowz.')
'example.com'
```

Some Operations on Strings

str.**rstrip**([*chars*])

Return a copy of the string with trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a suffix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.rstrip()
'   spacious'
>>> 'mississippi'.rstrip('ipz')
'mississ'
```

Some Operations on Strings

str.**strip**([*chars*])

Return a copy of the string with the leading and trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix or suffix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.strip()
'spacious'
>>> 'www.example.com'.strip('cmowz.')
'example'
```

The outermost leading and trailing *chars* argument values are stripped from the string. Characters are removed from the leading end until reaching a string character that is not contained in the set of characters in *chars*. A similar action takes place on the trailing end. For example:

```
>>> comment_string = '#....... Section 3.2.1 Issue #32 .......'
>>> comment_string.strip('.#! ')
'Section 3.2.1 Issue #32'
```

Some Operations on Strings

split()

join()

Some Operations on Strings

`str.`**`split`**(*sep=None, maxsplit=-1*)

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done (thus, the list will have at most `maxsplit+1` elements). If *maxsplit* is not specified or `-1`, then there is no limit on the number of splits (all possible splits are made).

If *sep* is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, `'1,,2'.split(',')` returns `['1', '', '2']`). The *sep* argument may consist of multiple characters (for example, `'1<>2<>3'.split('<>')` returns `['1', '2', '3']`). Splitting an empty string with a specified separator returns `['']`.

For example:

```
>>> '1,2,3'.split(',')
['1', '2', '3']
>>> '1,2,3'.split(',', maxsplit=1)
['1', '2,3']
>>> '1,2,,3,'.split(',')
['1', '2', '', '3', '']
```

If *sep* is not specified or is `None`, a different splitting algorithm is applied: runs of consecutive whitespace are regarded as a single separator, and the result will contain no empty strings at the start or end if the string has leading or trailing whitespace. Consequently, splitting an empty string or a string consisting of just whitespace with a `None` separator returns `[]`.

For example:

```
>>> '1 2 3'.split()
['1', '2', '3']
>>> '1 2 3'.split(maxsplit=1)
['1', '2 3']
>>> '   1   2   3   '.split()
['1', '2', '3']
```

Some Operations on Strings

`str.`**`join`**(*iterable*)

Return a string which is the concatenation of the strings in the iterable *iterable*. A `TypeError` will be raised if there are any non-string values in *iterable*, including `bytes` objects. The separator between elements is the string providing this method.

Some Operations on Strings

`str.`**`startswith`**(*prefix*[, *start*[, *end*]])

Return `True` if string starts with the *prefix*, otherwise return `False`. *prefix* can also be a tuple of prefixes to look for. With optional *start*, test string beginning at that position. With optional *end*, stop comparing string at that position.

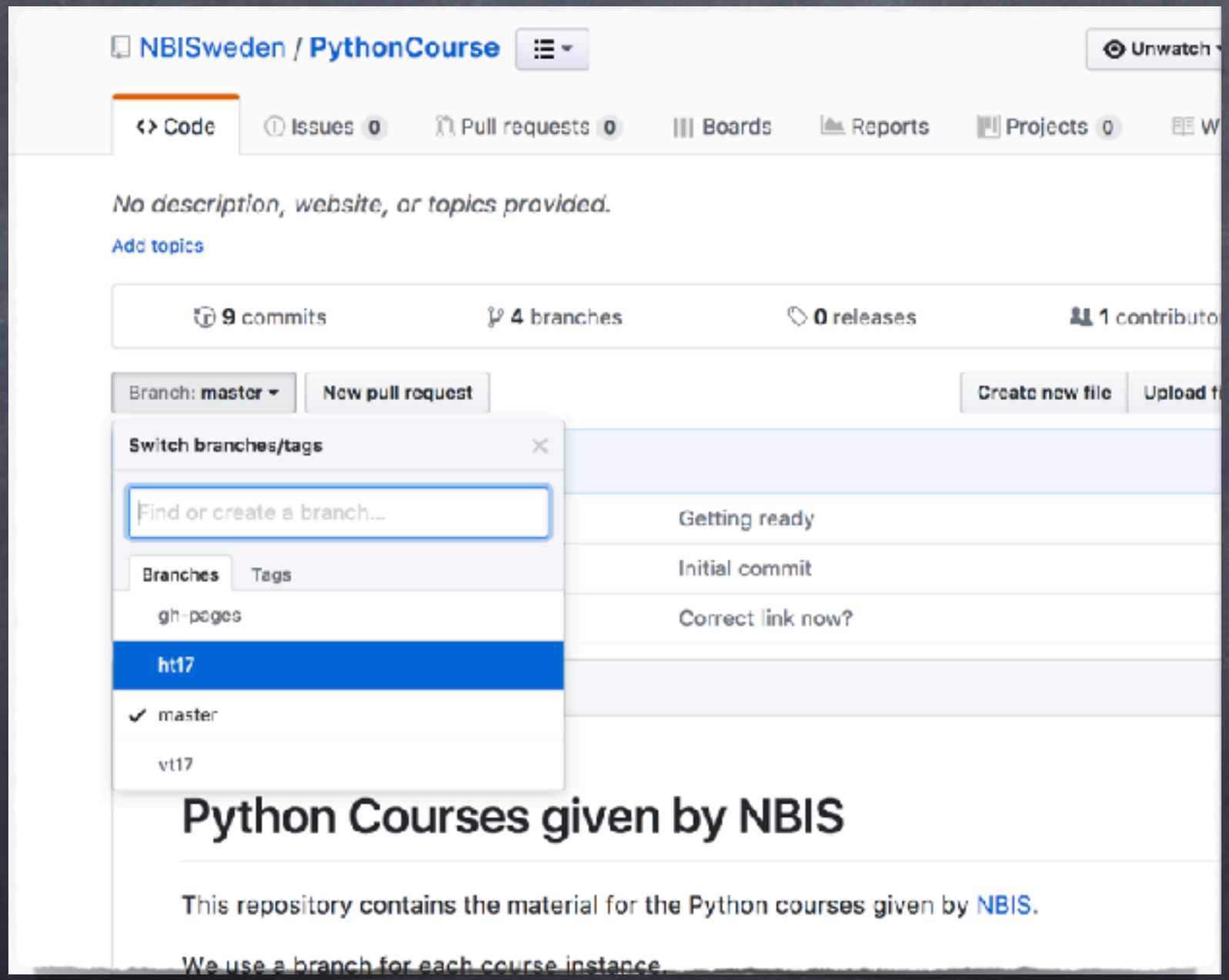
ImDB

Download the file 250.imdb

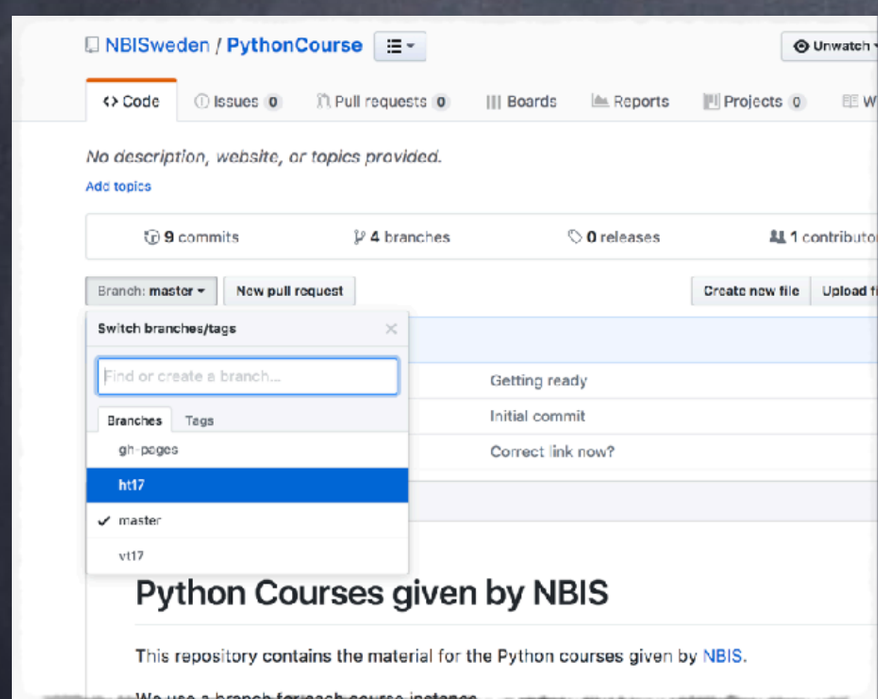
from

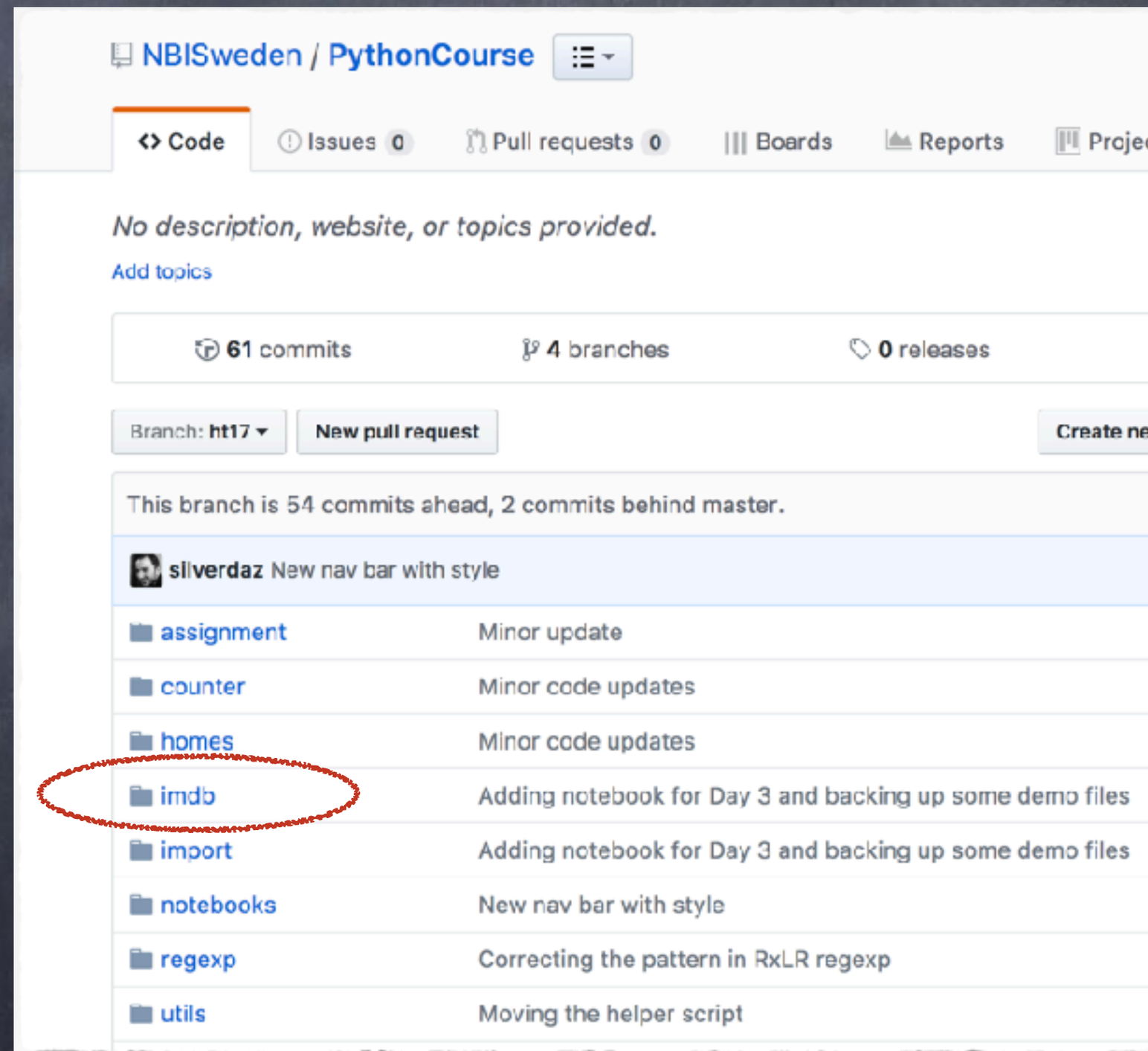
https://github.com/NBISweden/PythonCourse/

https://github.com/NBISweden/PythonCourse/



https://github.com/NBISweden/PythonCourse/





NBIS

Topics | Project | Preliminaries | Help

Introduction to Python - HT17

1 First programs

literals, variables, builtin types, int, float, str, list, bool, immutable, mutable, sequence (indexable), iterable, range, open, encoding, if/else, for, while, in (membership), comparators, stdlib.

Notebooks: [1] [2]

2 More Data Structures

set, tuple, dict, split, strip, min, max, sum, break, continue, and, or, not

250.imdb | Notebooks: [3] [4]

3 Using `dict`

dict, dict, dict, dict... function, positional argument, keyword argument, default value, import, sys.argv

4 Using someone else's code

import, documentation, function, default value, sys.argv, arguments

5 Formatting String and Pattern matching

format, positional argument, keyword argument, regular expression

NBISweden / Pythor

Code | Issues 0

No description, website,

Add topics

9 commits

Branch: master | New pul

Switch branches/tags

Find or create a branch...

Branches | Tags

gh-pages

ht17

master

vt17

Python Co

This repository cor

We use a branch fo

orts | Projec

ses

Create new

some demo files

some demo files

ImDB

The format of this file is

* line by line
* Column separated by the | character

The meaning of each column
is described on the first line:

Votes | Rating | Year | Runtime | URL | Genres | Title

ImDB

The format of this file is

* line by line
* Column separated by the | character

Votes | Rating | Year | Runtime | URL | Genres | Title

~~Find the best movie~~

Find the movie with most votes

ImDB

The format of this file is

* line by line
* Column separated by the | character

Votes | Rating | Year | Runtime | URL | Genres | Title

~~Find the best movie~~

~~Find the movie with most votes~~

For the category "Adventure",
find both:

* the top movie and print them
* the flop movie to the terminal

Find the number of categories.

Print them all.

Note: the answer is 21.
Think again if you get 24.

new data type: a `set`

Find the number of movies per category.

Print them all.

Dictionary

A mapping
key => value

Syntax: curly braces and colons

```
d = {
    'key1': 'value1',
    'key2': 'value2',
    'key3': 'value3',
    'key4': 'value4',
    # oh look, an extra comma.
    # Python can handle it
}
```

hashable

Not a good idea to use a mutable object

v[2]	str	'hello'
v[2]	list	[1,2,3,4,]
v[2]	tuple	(1,2,3,4,)
2 in v	set	{1,2,3,4,}
v['fr']	dict	{'en': 'hi', 'sv': 'hej', 'fr': 'salut', 'norrland': ' ', }

dict

`len(d)`	Number of items
`d[key]`	Returns the item `value` for key `key`
`d[key] = value`	Updating the mapping for `key` with `value`
`del d[key]`	Delete key from d
`key in d` `key not in d`	Membership tests
`d.keys()`	Returns an iterator on the keys
`d.values()`	Returns an iterator on the values
`d.items()`	Returns an iterator on the pair (key,value)
`d.update(other)`	Add (or overwrite) the mappings with the ones from `other`

➥ Notebook 4

I've got a little time on my hands
and I'm not too picky:

Find the first drama movie
over 8.7
under 2h

Operations on `bool`

These are the Boolean operations, ordered by ascending priority:

Operation	Result	Notes
`x or y`	if *x* is false, then *y*, else *x*	(1)
`x and y`	if *x* is false, then *x*, else *y*	(2)
`not x`	if *x* is false, then `True`, else `False`	(3)

Notes:

1. This is a short-circuit operator, so it only evaluates the second argument if the first one is `False`.
2. This is a short-circuit operator, so it only evaluates the second argument if the first one is `True`.
3. `not` has a lower priority than non-Boolean operators, so `not a == b` is interpreted as `not (a == b)`, and `a == not b` is a syntax error.

```
if x == y and a >= b and e < f:
    print('Gotcha')
else:
    print('Not you')
```

Recall: else is optional

No a > b or c > d and e > f or not g and x != y

Parenthesis !!

First something...

Stop the loop if item found

```
someList = [1,7,2,4,14,11,9,3]
for item in someList:
    if item > 10:
        print(item)
        break
```

```
someList = [1,7,2,4,14,11,9,3]
for item in someList:
    if item <= 10:
        continue
    else:
        print(item)
```

```
someList = [1,7,2,4,14,11,9,3]
for item in someList:
    if item <= 10:
        continue
    else:
        print(item)
```

```
someList = [1,7,2,4,14,11,9,3]
for item in someList:
    if item <= 10:
        continue

    print(item)
```

I've got a little time on my hands
and I'm not too picky:

Find the first drama movie
over 8.7
under 2h