glob and pathlib

Shell patterns

In addition to re syntax there is a shell patterns which is used to match paths in shell

- `*` - matches any symbol in any quantity except leading `.` (like `.bashrc`) - almost analogous to `.*` in re
- `?` - matches any one symbol
- `[a-z]` - matches range of characters
- `[[:class:]]` - matches some predefined class of characters - alnum, alpha, ascii, blank, cntrl, digit, graph, lower, print, punct, space, upper, word and xdigit

glob

Library for matching paths

- `glob(pattern, recursive=False)` - main method, returns list of paths, which match the pattern. If `recursive` is `True` `**` in the end of pattern will match everything recursively
- `iglob(pattern, recursive=False)` - same stuff, returns an iterator instead of a list
- `escape(pattern)` - returns string where special characters in a pathname were escaped

```
from glob import glob


glob('/home/arleg/PycharmProjects/[t-z]*')
['/home/arleg/PycharmProjects/victory',
 '/home/arleg/PycharmProjects/useful_scripts',
 '/home/arleg/PycharmProjects/tolya']

glob('/home/arleg/PycharmProjects/c?at')
['/home/arleg/PycharmProjects/caat',
 '/home/arleg/PycharmProjects/coat']
```

```
glob('/home/arleg/PycharmProjects/useful_scripts/**')
['/home/arleg/PycharmProjects/useful_scripts/scripts',
 '/home/arleg/PycharmProjects/useful_scripts/LICENSE',
 '/home/arleg/PycharmProjects/useful_scripts/README.md',
'/home/arleg/PycharmProjects/useful_scripts/jupyter_notebooks']


glob('/home/arleg/PycharmProjects/useful_scripts/**',
     recursive=True)
['/home/arleg/PycharmProjects/useful_scripts/',
 '/home/arleg/PycharmProjects/useful_scripts/scripts',
 '/home/arleg/PycharmProjects/useful_scripts/scripts/colored_ar
gparse.py',
 '/home/arleg/PycharmProjects/useful_scripts/scripts/ftp_parse.
py',
 ...]
```

```
from glob import iglob


base_path = '/home/arleg/PycharmProjects/simulations/**/*.py'

iglob(base_path, recursive=True)
<generator object _iglob at 0x7feafee51468>
```

```
for path in iglob(base_path, recursive=True):
    print(path)
```

```
/home/arleg/PycharmProjects/simulations/predators/demo1_1.py
/home/arleg/PycharmProjects/simulations/predators/demo1_1_aw
 are.py
/home/arleg/PycharmProjects/simulations/predators/trial.py
/home/arleg/PycharmProjects/simulations/predators/utils.py
/home/arleg/PycharmProjects/simulations/predators/prey.py
/home/arleg/PycharmProjects/simulations/predators/predator.py
/home/arleg/PycharmProjects/simulations/predators/cart.py
/home/arleg/PycharmProjects/simulations/predators/steps/tr.py
```

os module

Library for work with Operating System. Have diverse functionality, we will talk about paths today

Finding where are you and directories content

- `getcwd()` - infer directory where are you now, returns path as a string, corresponds to `pwd`
- `listdir(path='.')` - get list of paths to files and dirs on specified path, corresponds to `ls`
- `scandir(path='.')` - get iterator of paths to files and dirs on specified path, here paths are special objects (later about it)

```
import os


# Where am I?
os.getcwd()
'/home/arleg'

# Get content of the 16.file_management directory
os.listdir('/home/arleg/PycharmProjects/bf_course/
            16.file_management')
['Lord_of_a_Thousand_Suns', 'README.md']
```

What should you do now to get access for a file?

Path joining

Joining paths is kinda easy - naive way is too concat path strings

```
'/home/arleg/PycharmProjects/bf_course/16.file_manag
ement' + '/' + 'Lord_of_a_Thousand_Suns'
```

'/home/arleg/PycharmProjects/bf_course/16.file_management/Lord_of
_a_Thousand_Suns'

You'd better use strings template

```
basepath =
'/home/arleg/PycharmProjects/bf_course/16.file_manag
 ement'
filename = 'Lord_of_a_Thousand_Suns'
f'{basepath}/{filename}'
```

What's wrong with the previous approach?

It won't work on all os. But we have methods in os, so everything is ok

```
os.path.join(basepath, filename)
'/home/arleg/PycharmProjects/bf_course/16.file_management/Lord_
 of_a_Thousand_Suns'

os.path.join('/home/', 'arleg', 'Downloads')
'/home/arleg/Downloads'
```

Check existance

- `os.path.exists(path)` - check whether the `path` is present
- `os.path.isfile(path)` - check whether the `path` is present and it is a file
- `os.path.isdir(path)` - check whether the `path` is present and it is a directory

There are other functions for checking links and so on

```
os.path.exists('Wrong path')
False

os.path.exists('/etc')
True

os.path.isfile('/bin')
False

os.path.isdir('/bin')
True
```

Back to scandir

```
# As expected
os.scandir('/sys')
<posix.ScandirIterator object at 0x7f2dc9f43c30>

for path in os.scandir('/sys'):
    print(path)
<DirEntry 'fs'>
<DirEntry 'bus'>
<DirEntry 'dev'>
<DirEntry 'devices'>
<DirEntry 'block'>
<DirEntry 'class'>
<DirEntry 'power'>
...
```

It is advised to use context manager here

```
with os.scandir('/sys') as paths:
    for path in paths:
        print(path)
```

How to extract paths here?

DirEntry has a number of useful attributes and methods

- `name` - name of the file/directory
- `path` - full path to the object from the specified path (same as joining path after `listdir`)
- `is_dir(follow_symlinks=True)` - whether the object is directory, if `follow_symlinks` will follow link and returns whether the linked object is directory
- `is_file(follow_symlinks=True)` - everything the same, just for a file
- `stat(follow_symlinks=True)` - returns detailed info about the object (permissions, size, etc)

```
with os.scandir('/sys') as paths:
    for path in paths:
        print(path.name, path.path, path.is_dir(),
              path.is_file(), path.stat(), sep='\n')
        break
```

```
fs
/sys/fs
True
False
os.stat_result(st_mode=16877, st_ino=2, st_dev=18, st_nlink=8,
 st_uid=0, st_gid=0, st_size=0, st_atime=1584027875,
 st_mtime=1584027875, st_ctime=1584027875)
```

Another useful function

`os.walk(top, followlinks=False)` - iterate over directories inside `top` directory and obtain tuples with name of directory, directories in it and files in it. If `followlinks` is True will visit linked objects too

```
for i, triple in enumerate(os.walk('bf_course')):
    if i > 2:
        break
    print(triple, end='\n\n')
```

```
('/home/arleg/PycharmProjects/bf_course',
['.ipynb_checkpoints', '19.development_metodologies',
 '12.functional_programming', '6.functions', ...],
['draft.py', 'test_after_10', 'README.md', ...])

('/home/arleg/PycharmProjects/bf_course/.ipynb_checkpoints',
[], ['Untitled-checkpoint.ipynb',
 'serpinsky_triangle-checkpoint.ipynb'])

('/home/arleg/PycharmProjects/bf_course/19.development_meto
  dologies',
[],
['README.md'])
```

Directories

- `makedirs(name, mode=0o777, exist_ok=False)` - method to create directories, `name` takes desired path of new dir, `mode` - defines permissions of it, `exist_ok` - one of the coolest name of parameter -
  - if False will raise an Error if such directory exists
  - if True will go next
- `mkdir(name, mode=0o777)` - sucks because can't create new nested directories like /existed/new_idir/new_nested_dir

```
# Create new dir
os.makedirs('new')

# new was created on previous step
os.makedirs('new')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/arleg/anaconda3/lib/python3.6/os.py",
 line 220, in makedirs
    mkdir(name, mode)
FileExistsError: [Errno 17] File exists: 'new'

# Ignore directory
os.makedirs('new', exist_ok=True)
```

Removing directories

There is function rmdir for removing directory, but is inconvenient because can delete only empty dirs. Thus we have another module shutil

`rmtree(path)` - remove directory on `path` and all its content, accept only a directory!

Removing files

For this purpose `os.remove` function works well. Corresponds to `rm`

`remove(path)` - remove file on `path` and all its content, accept only a file!

```
import shutil


# Delete directory recursively
shutil.rmtree('new')

# Delete file
os.remove('not_important_file')
```

Copying

Entirely on shutil, correspond to `cp`

- `copytree(src, dst)` - copy directory `src` to `dst`
- `copy(src, dst)` - copy file `src` to `dst`... to be honest there are 3 functions to copy files with some differences

```
# Copy directory
shutil.copytree('bf_course/1.bash', 'Downloads/bash')
'Downloads/bash'

# Copy file
shutil.copy('new_file', 'Downloads/new_file')
'Downloads/new_file'
```

Move

`move(src, dst)` - move `src` object to `dst`

- if `dst` a directory - move `src` in this dir
- if `dst` a file - move src in the necessary directory and possibly change name

```
# Move file
shutil.move('new_file', '~/Downloads/new_file')
'~/Downloads/new_file'

# Rename file
shutil.move('~/Downloads/new_file', '~/Downloads/nf')
'~/Downloads/nf'
```

pathlib

pathlib is a new solution for work with paths and probably you should use it instead of other. You can pass path object to various place where we need a string with path

```
from pathlib import Path

# Specify path
my_path = Path('/home')
my_path
PosixPath('/home')
```

Joining paths

Looks nice, imho

```
my_path / 'arleg' / Path('Downloads')
PosixPath('/home/arleg/Downloads')

# Same stuff
my_path.joinpath('arleg', Path('Downloads'))
PosixPath('/home/arleg/Downloads')
```

Mutate path

```
# Elongate our path
my_path /= 'arleg/file'

# Change last part after the root
my_path.with_name('another_file')
PosixPath('/home/arleg/another_file')

# Change extension of the file
my_path = my_path.with_suffix('.py')
my_path
PosixPath('/home/arleg/file.py')

my_path.with_suffix('.hs')
PosixPath('/home/arleg/file.hs')
```

Utilities

```
Path.cwd()
PosixPath('/home')

Path.home()
PosixPath('/home/arleg')

Path('/etc').exists()
True

# Find all python scripts in PycharmProjects and all
 subdirectories
Path('/home/arleg/PycharmProjects/').glob('**/*.py')
<generator object Path.glob at 0x7f2dc9f53728>
```

```
# Whether it is directory or file
Path('.').is_dir()
True

Path('.').is_file()
False

# Directory content
Path('.').iterdir()
<generator object Path.iterdir at 0x7f2dc9f53728>

# Get rid of tilda
Path('~/../arleg/PycharmProjects/bf_course/15.re').expanduser()
PosixPath('/home/arleg/../arleg/PycharmProjects/bf_course/15.re')
# Get rid of . and ..
Path('~/../arleg/PycharmProjects/bf_course/15.re').expanduser() \
                                                  .resolve()
PosixPath('/home/arleg/PycharmProjects/bf_course/15.re')
```

Attributes

- `parts` - returns list if path components (root + other parts)
- `root` - returns root
- `drive` - windows thing (C://)
- `anchor` - returns drive + root
- `parents` - ordered sequence of parents
- `parent` - closest parent, if you have `..` in a path, use `resolve` method first
- `name` - returns name of the object (last path component)
- `suffix` - returns file extension
- `stem` - name without suffix

```
# Some path
path = Path('/home/arleg/PycharmProjects/bf_course/15.re')

# Components
path.parts
('/', 'home', 'arleg', 'PycharmProjects', 'bf_course', '15.re')

# Root and other stuff
path.root
'/'
path.drive
''
path.anchor
'/'
```

```
# Family things
path.parents
<PosixPath.parents>
path.parents[0]
PosixPath('/home/arleg/PycharmProjects/bf_course')

list(path.parents)
[PosixPath('/home/arleg/PycharmProjects/bf_course'),
 PosixPath('/home/arleg/PycharmProjects'),
 PosixPath('/home/arleg'), PosixPath('/home'), PosixPath('/')]

path.parent
PosixPath('/home/arleg/PycharmProjects/bf_course')

# Name of the object
path.name
'15.re'
```

```
path.suffix
'.re'

path.stem
'15'
```

Also pathlib can create files/directories, rename them, delete and do other stuff