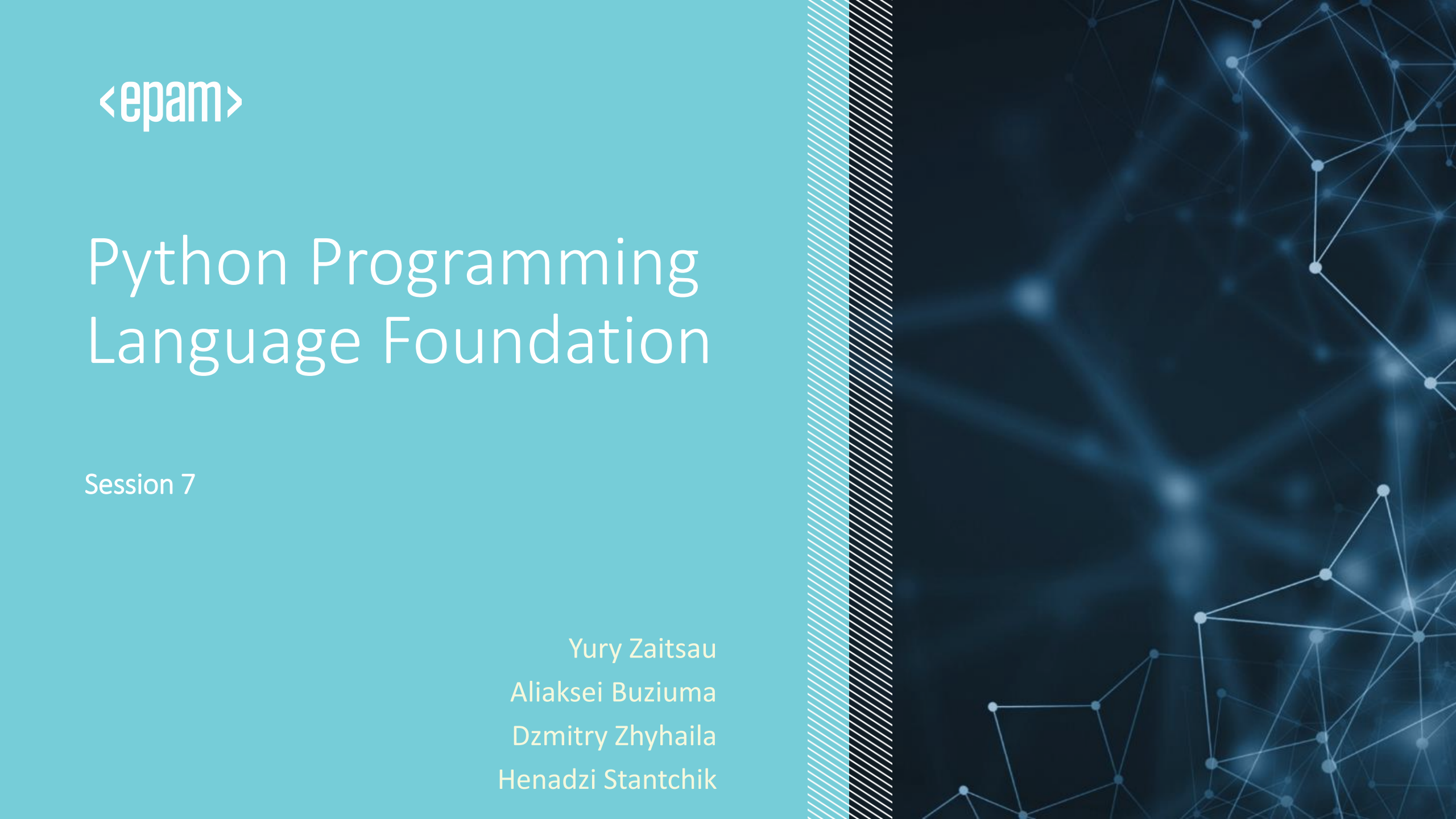


Python Programming Language Foundation

Session 7

Yury Zaitsau
Aliaksei Buziuma
Dzmitry Zhyhaila
Henadzi Stantchik

Attendance check

https://epa.ms/attendance-bsu-24-10-19

Session overview

- Iterators and generators
- venv
- Profiling
- Python code style

Iterators and generators in Python

What is iterator?
Iterator is an object which defines order of walking through a collection.

What is collection?
Collection is an object that allows you to look over its elements one by one.

What is *iterable object*?
Iterable object is object that iterator walks through.

Iterations protocol consists of two magic methods: __iter__ and __next__

	Iterable object	Iterator
__iter__	Returns iterator object	Returns ‘self’
__next__	-	Returns next element or if there are no more raises StopIteration

```
>>> a = [1, 2, 3]
>>> type(a)
<class 'list'>
>>> it = iter(a)
>>> type(it)
<class 'list_iterator'>
>>> number = next(a)
TypeError: 'list' object is not an iterator
```

Iterations protocol consists of two magic methods: __iter__ and __next__

	Iterable object	Iterator
__iter__	Returns iterator object	Returns ‘self’
__next__	-	Returns next element or if there are no more raises StopIteration

```
>>> a = [1, 2, 3]
>>> it = iter(a)
>>> number = next(it)
>>> print(number)
1
```

Iterators

```
>>> x = [1, 2, 3]
>>> y = iter(x)
>>> z = iter(x)
>>> next(y)
1
>>> next(y)
2
>>> next(z)
1
>>> type(x)
<class 'list'>
>>> type(y)
<class 'list_iterator'>
```

‘for’ cycle in python

```
a = [1,2,3]
for item in a:
    print(item)

1 2 3
```

```
a = [1,2,3]
it = iter(a)
while True:
    try:
        item = next(a)
        print(item)
    except StopIteration:
        break

1 2 3
```

```
class MyEvenIterator:

    def __init__(self, iterable):
        self.iterable_object = iterable
        self.current_item_number = 0

    def __iter__(self):
        return self

    def __next__(self):
        try:
            current_number = self.current_item_number
            self.current_item_number += 2
            return self.iterable_object[current_number]
        except IndexError:
            raise StopIteration
```

```
a = [0,1,2,3,4,5,6,7,8,9]
it = MyEvenIterator(a)

for item in it:
    print(item)

0 2 4 6 8
```

```
class Fib:
    def __init__(self):
        self.prev = 0
        self.curr = 1

    def __iter__(self):
        return self

    def __next__(self):
        value = self.curr
        self.curr += self.prev
        self.prev = value
        return value
```

```
f = Fib()

for _ in range(6):
    print(next(f))
```

```
1 1 2 3 5 8
```

```
def fib():
    prev, curr = 0, 1
    while True:
        yield curr
        prev, curr = curr, prev + curr


f = fib()

for _ in range(6):
    print(next(f))
```

```
1 1 2 3 5 8
```

```
def power_input():
    x = 1
    while True:
        value = yield x ** 2
        x = value if value else x + 1


gen1 = power_input()
gen2 = power_input()
gen1 is gen2 # False
print(next(gen1))  # 1
print(next(gen1))  # 4
print(gen1.send(10))  # 100
print(next(gen1))  # 121
gen1.trow(ValueError)  # VE with tb to yield
```

Generator Expressions vs List Comprehension

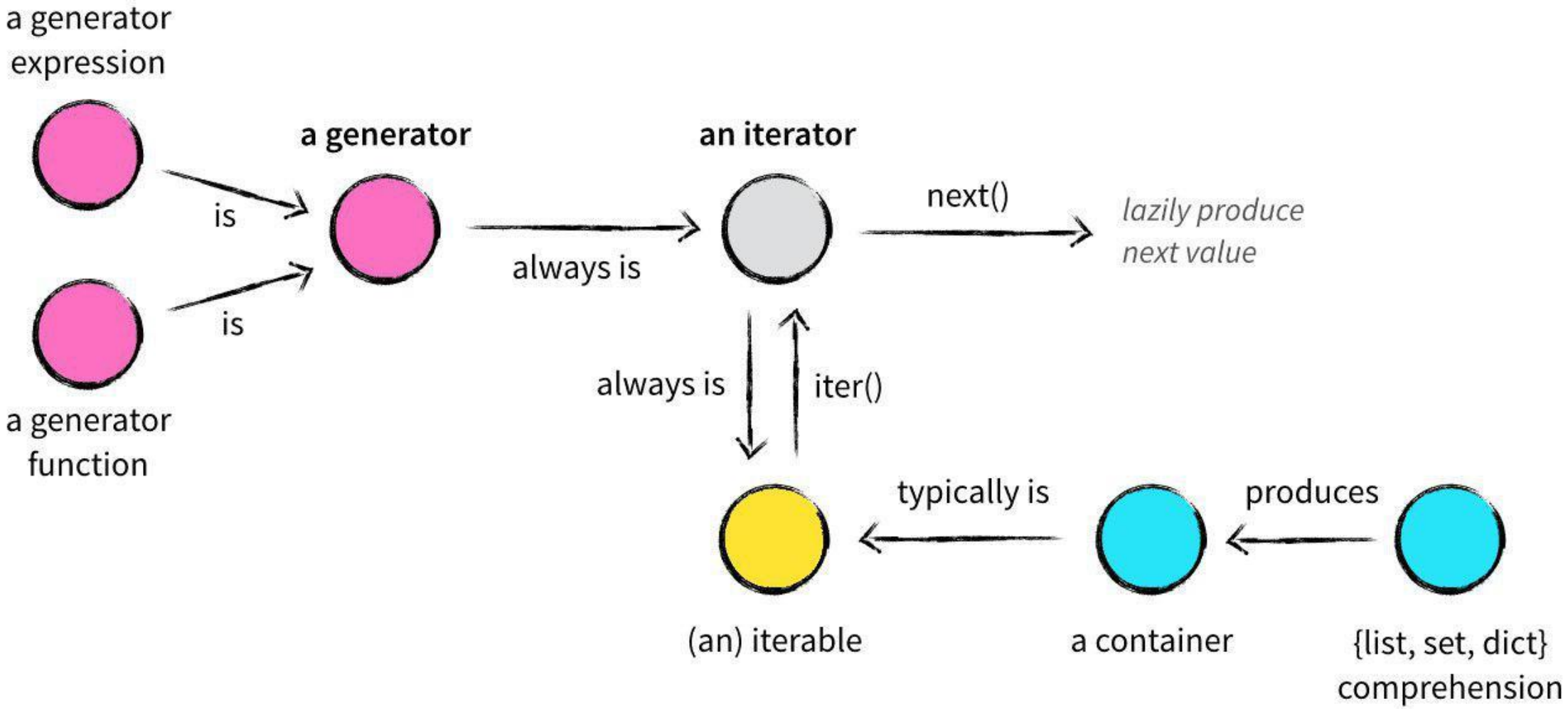
Topic	Generator Expression	List Comprehension
Memory usage	+	-
Ability to stop at any time	+	-
len()?	-	+
Serializable	-	+
speed	???	???

```
a = [1,2,3,4……,999999]
```

```
gen = (x ** 2 for x in a)
```

```
lc = [x ** 2 for x in a]
```

Generators and Iterators



Creation of virtual environments with **venv**

The venv module provides support for creating lightweight “virtual environments” with their own site directories, optionally isolated from system site directories. Each virtual environment has its own Python binary (which matches the version of the binary that was used to create this environment) and can have its own independent set of installed Python packages in its site directories.

Documentation: https://docs.python.org/3/tutorial/venv.html

How to create venv?

```
python3 -m venv tutorial-env
```

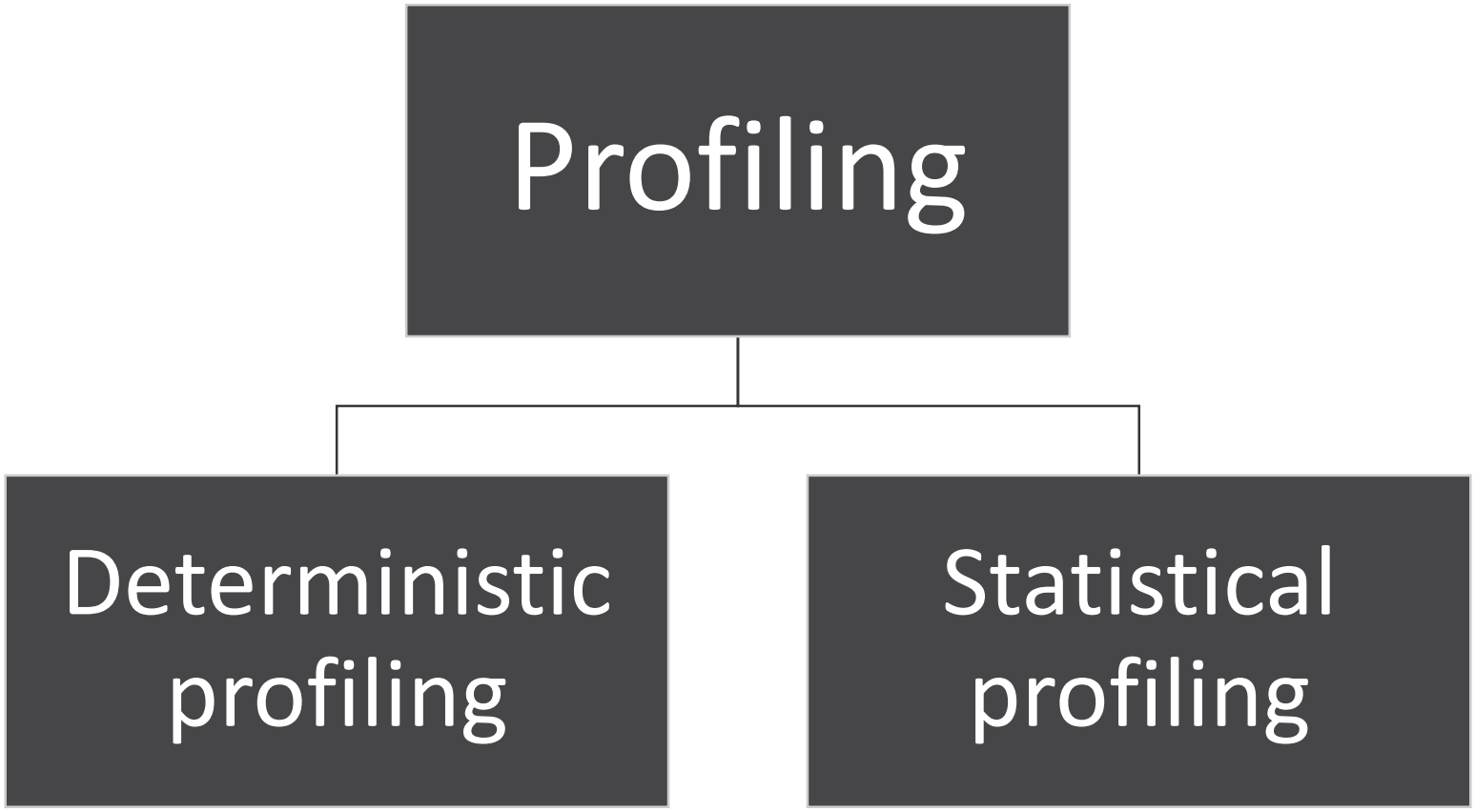
How to activate environment?

```
source tutorial-env/bin/activate
```

How to activate environment?

```
(tutorial-env) user@host:~$ python
>>> from pprint import pprint
>>> import sys
>>> pprint(sys.path)
['',
 '/usr/lib/python36.zip’,
 '/usr/lib/python3.6’,
 '/usr/lib/python3.6/lib-dynload’,
 '/home/user/tutorial-env/lib/python3.6/site-packages']
```

Python profiling



https://docs.python.org/3/library/profile.html#what-is-deterministic-profiling

```
import cProfile
import re
cProfile.run('re.compile("foo|bar")')
```

cProfile

```
243 function calls (236 primitive calls) in 0.000 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 <string>:1(<module>)
        2    0.000    0.000    0.000    0.000 enum.py:281(__call__)
        2    0.000    0.000    0.000    0.000 enum.py:537(__new__)
        9    0.000    0.000    0.000    0.000 enum.py:614(name)
        1    0.000    0.000    0.000    0.000 enum.py:784(_missing_)
        1    0.000    0.000    0.000    0.000 enum.py:791(_create_pseudo_member_)
        1    0.000    0.000    0.000    0.000 enum.py:827(__and__)
        1    0.000    0.000    0.000    0.000 enum.py:863(_decompose)
        1    0.000    0.000    0.000    0.000 enum.py:881(<listcomp>)
        1    0.000    0.000    0.000    0.000 re.py:232(compile)
        1    0.000    0.000    0.000    0.000 re.py:271(_compile)
        ...
```

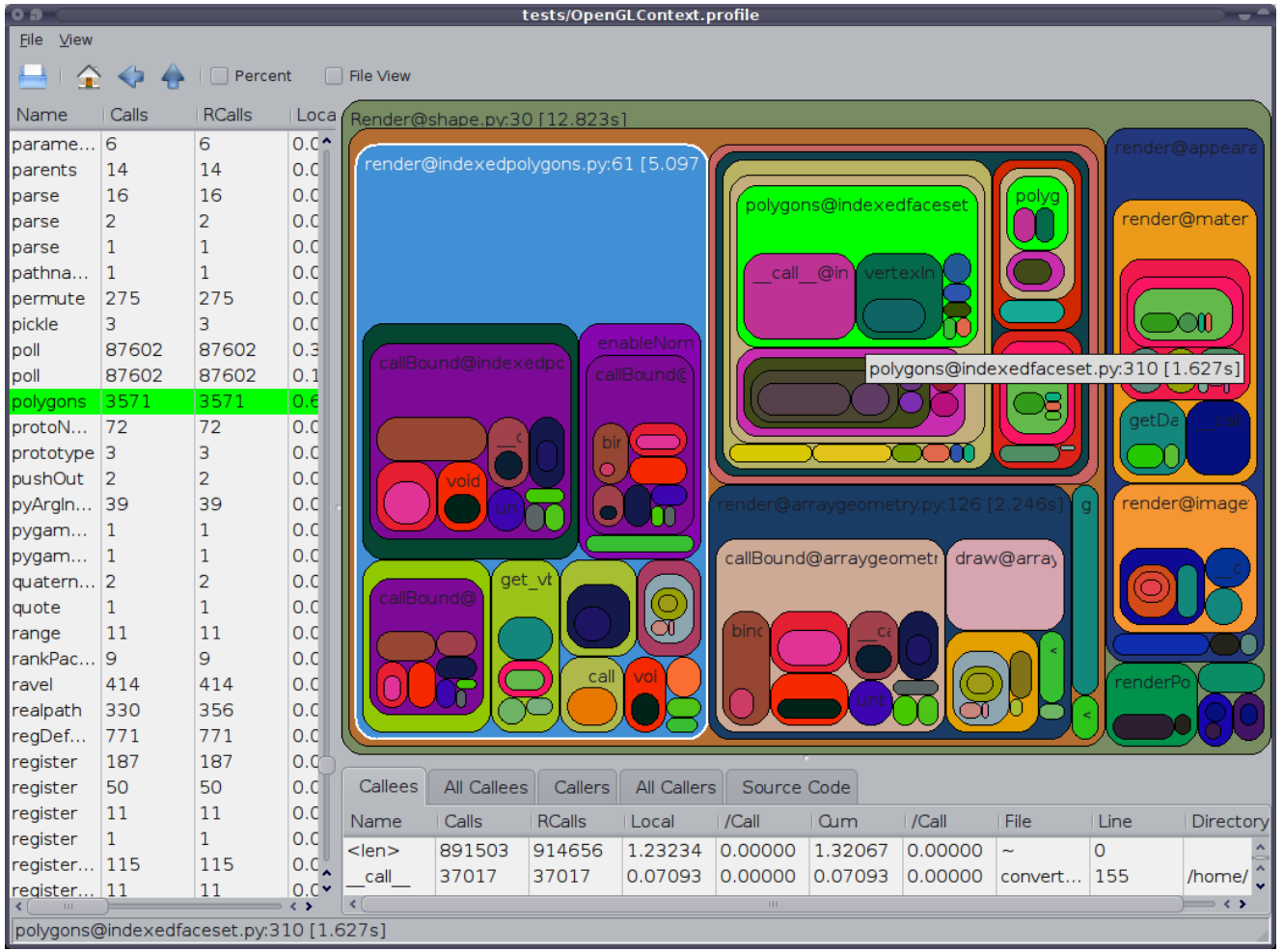
cProfile

```
import cProfile
prof = cProfile.Profile()
prof.enable()

# Do work here
re.compile("foo|bar")

prof.disable()
prof.dump_stats("/tmp/results.profile")
```

RunSnakeRun



http://www.vrplumber.com/programming/runsnakerun/

```
from memory_profiler import profile

@profile
def my_func():
    a = [1] * (10 ** 6)
    b = [2] * (2 * 10 ** 7)
    del b
    return a

if __name__ == '__main__':
    my_func()
```

```
Line #    Mem usage    Increment  Line Contents
================================================
     3                             @profile
     4      5.97 MB    0.00 MB   def my_func():
     5     13.61 MB    7.64 MB       a = [1] * (10 ** 6)
     6    166.20 MB  152.59 MB       b = [2] * (2 * 10 ** 7)
     7     13.61 MB -152.59 MB       del b
     8     13.61 MB    0.00 MB       return a
```

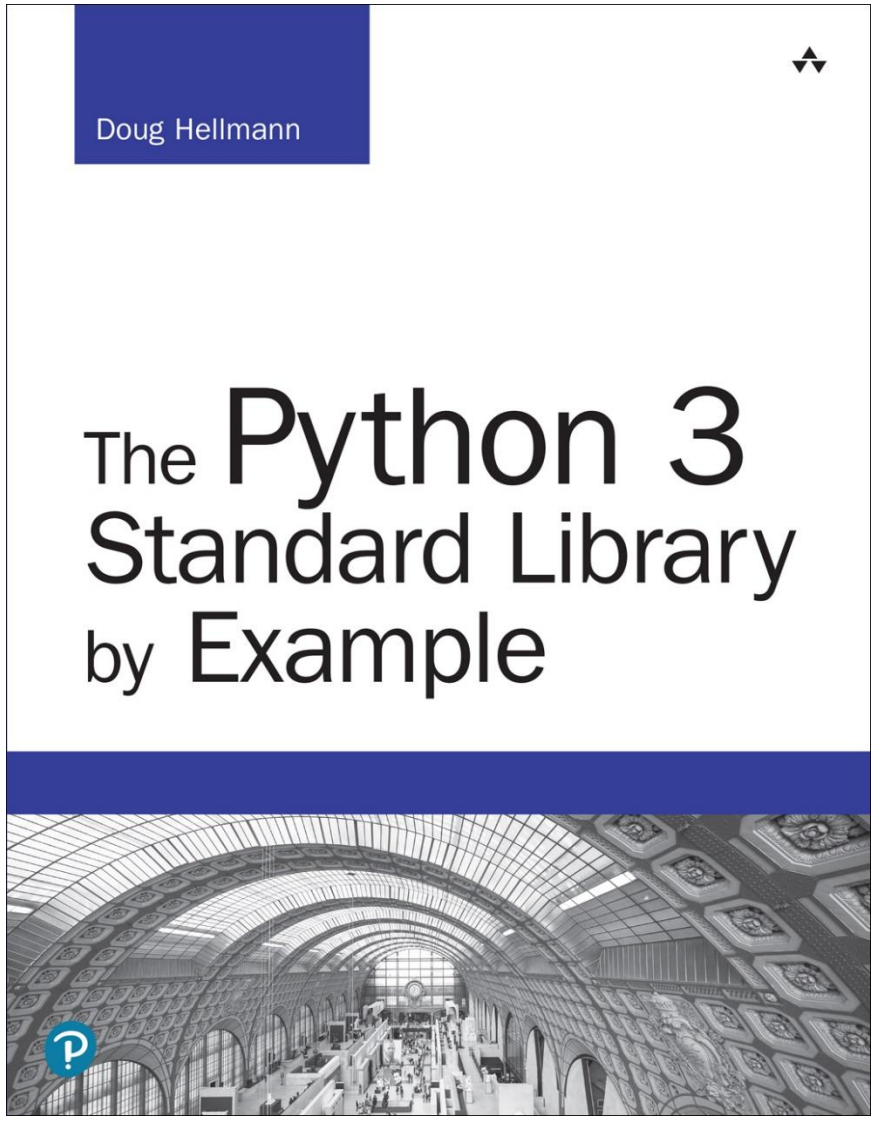
https://pypi.org/project/memory_profiler/

Materials

Python 3 Module of the Week

https://pymotw.com/3/

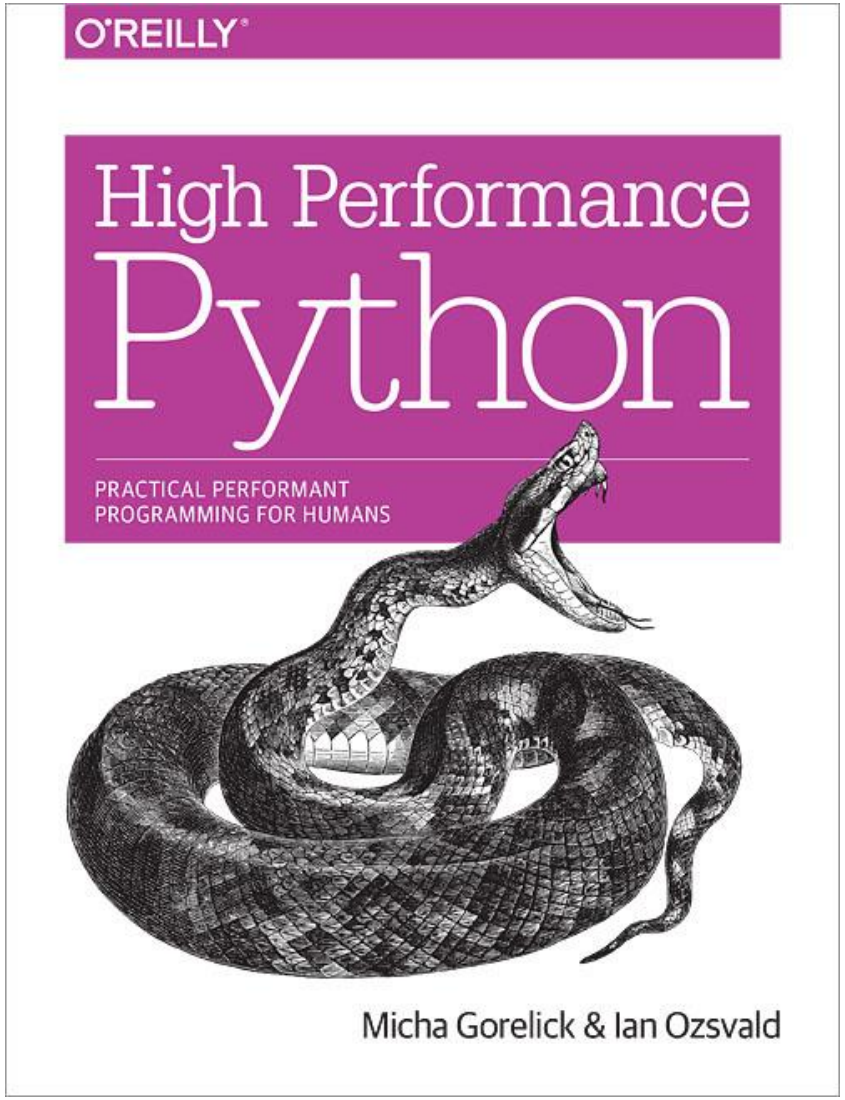
Python Standard Library by Example



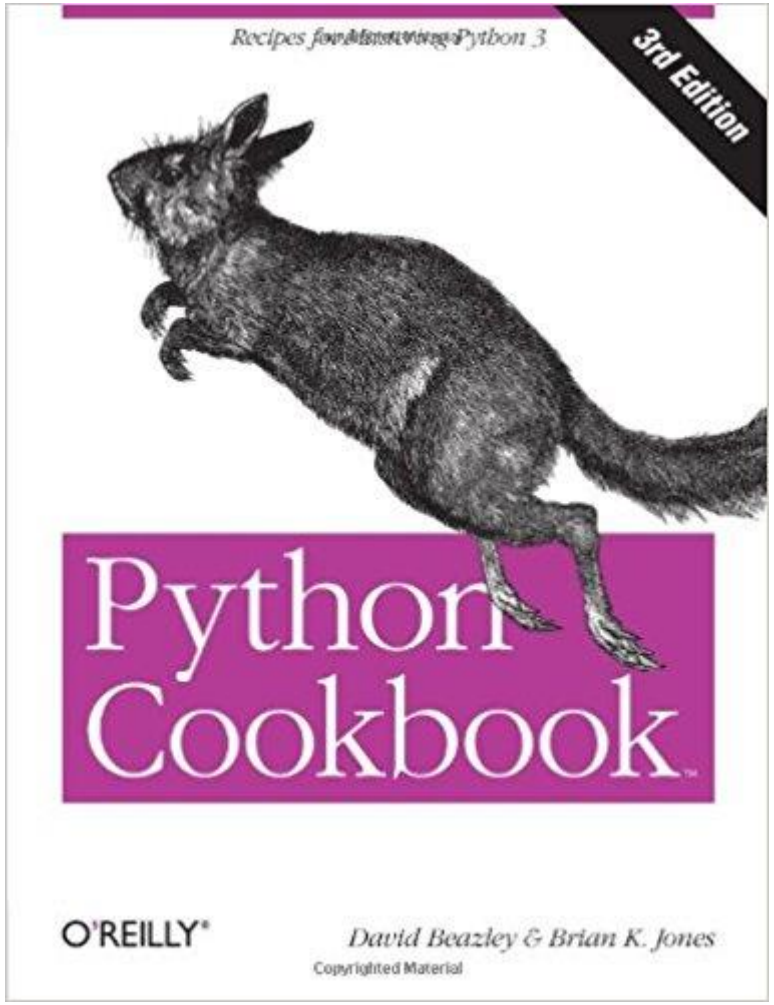
Awesome python

https://github.com/vinta/awesome-python

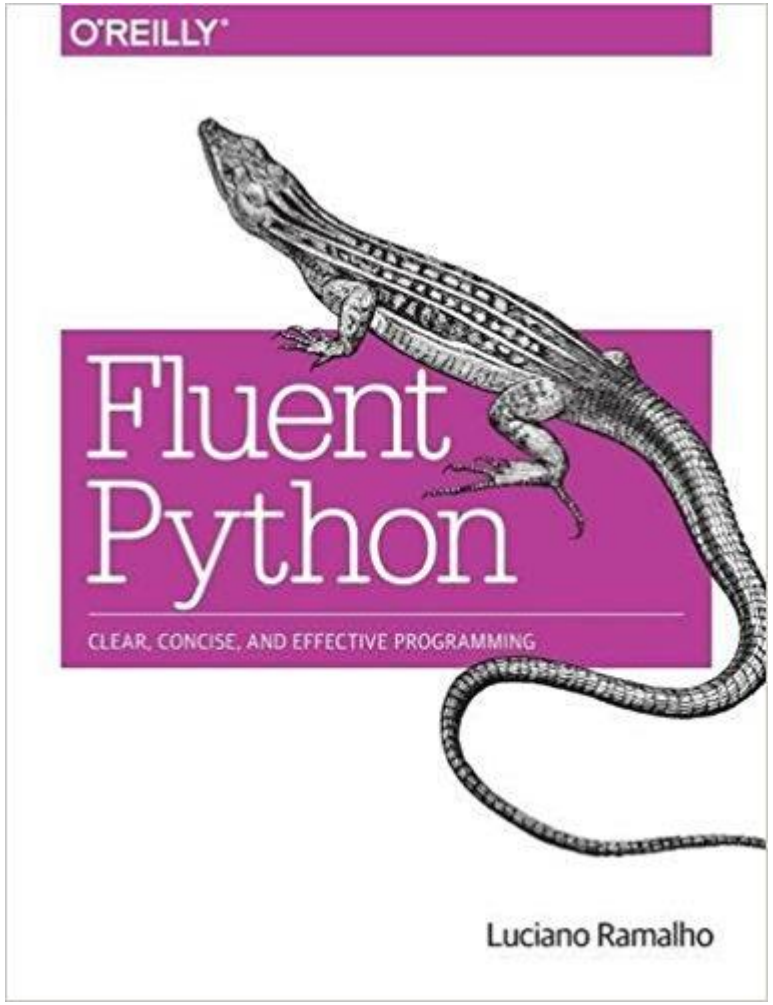
High Performance Python



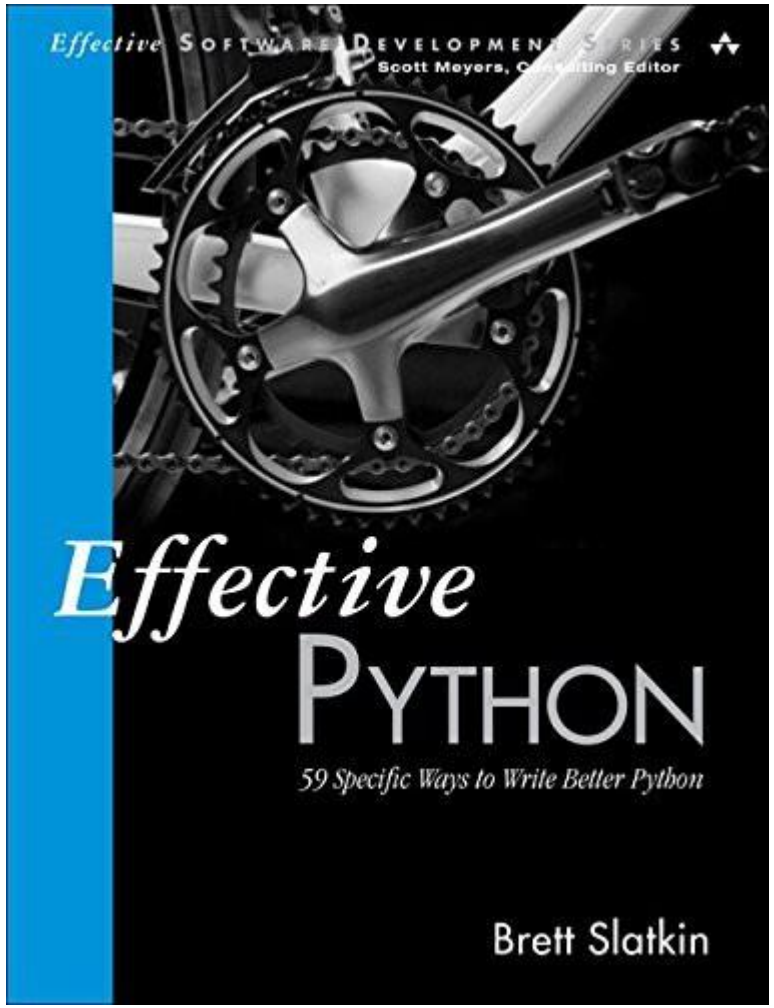
Python Cookbook



Fluent Python



Effective Python



Python weekly

https://www.pythonweekly.com/

Code style

Thanks for attention

Yury_Zaitsau1@epam.com
Aliaksei_Buziuma@epam.com
Dzmitry_Zhyhaila@epam.com
Henadzi_Stantchik@epam.com