EDGE ЦЕНТР

Когда обновляться?

Что полезного в Python
после версии 3.7

Sk Сколково

АРПП Отечественный софт

О спикере

Алексей Марашов

- Руководитель отдела разработки продуктов Cloud в EdgeЦентр
- 6+ лет в разработке на Python (ex Aviasales, ex Technokratos)
- BCS Software Engineering (СПбГЭТУ «ЛЭТИ», 2016)
- MCS Data Science (Innopolis, 2018)

План доклада

1

Обзор языка Python

2

История развития

3

Python Enhancement Proposal (PEP)

4

Этапы запуска Python программы

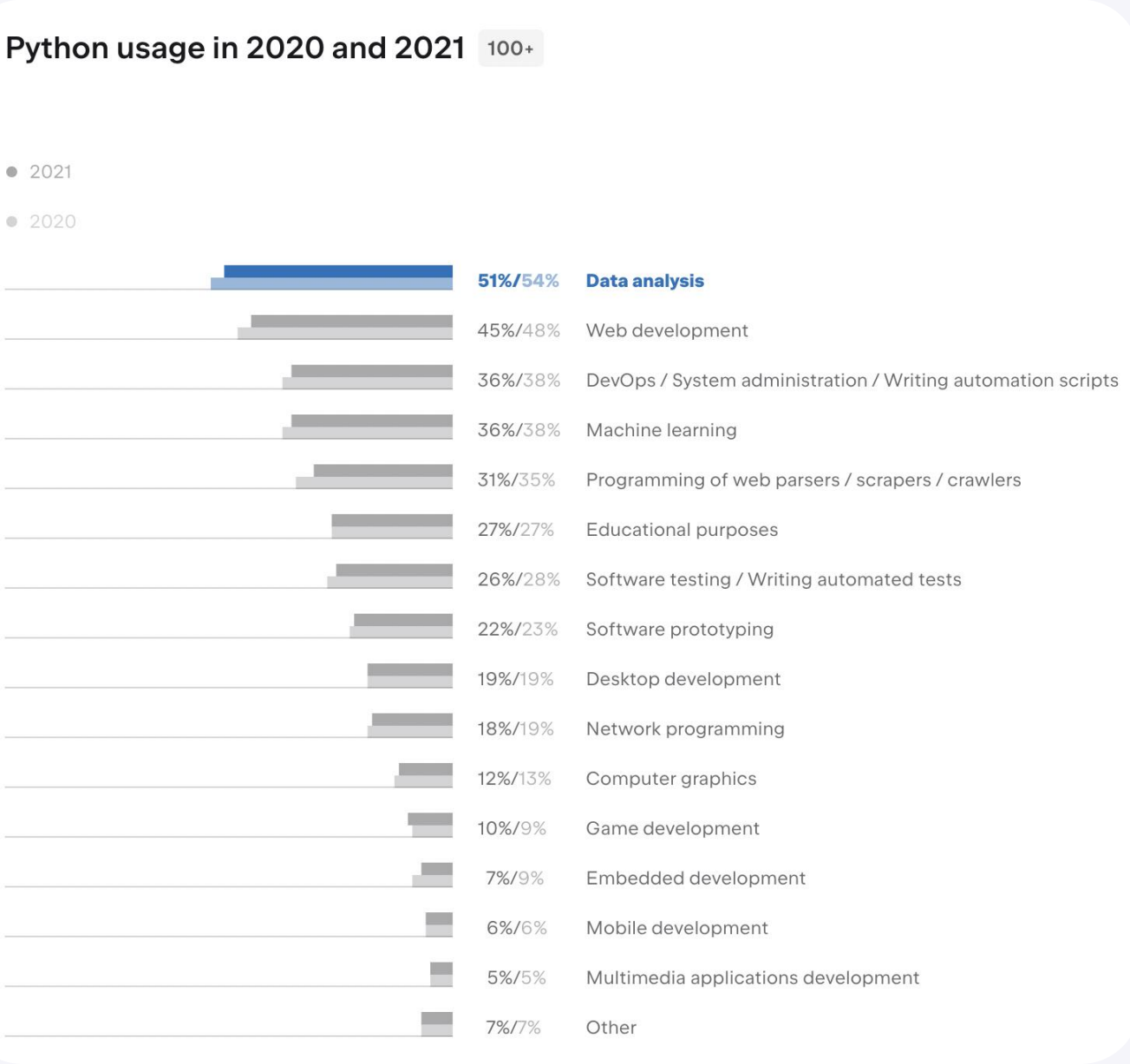
5

Из чего состоит интерпретатор

6

Синтаксис новых версий

В каких областях применяется



Источник: JetBrains

Ключевые этапы разработки Python

1

Версия 0.9.0

Модули, классы с наследованием, основные типы данных, обработка исключений.

2

Версия 1.0

Элементы функционального программирования.

3

Версия 2.X

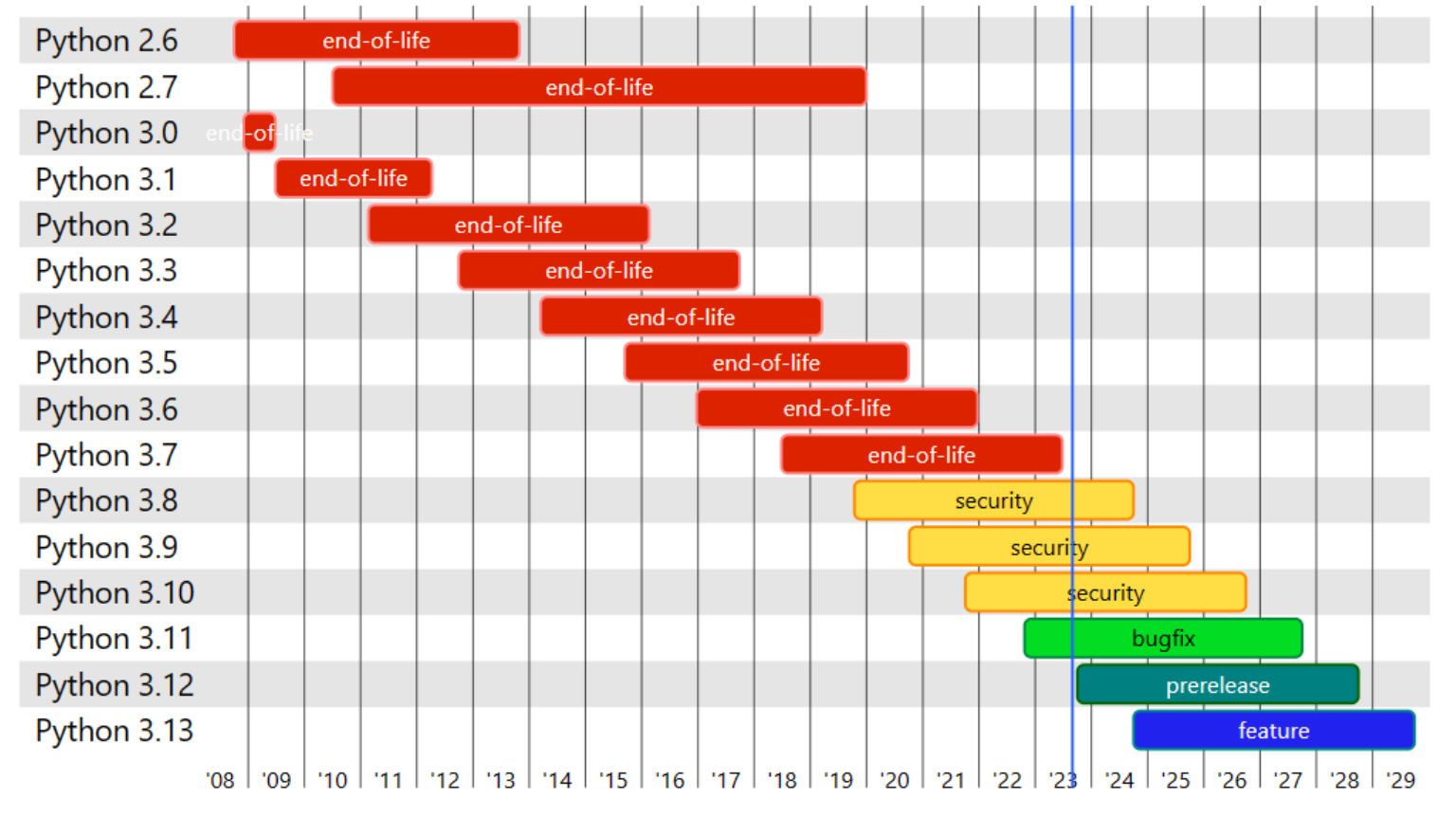
Сборщик мусора с поддержкой циклических ссылок, генераторы, элементы ООП.

4

Версия 3.0

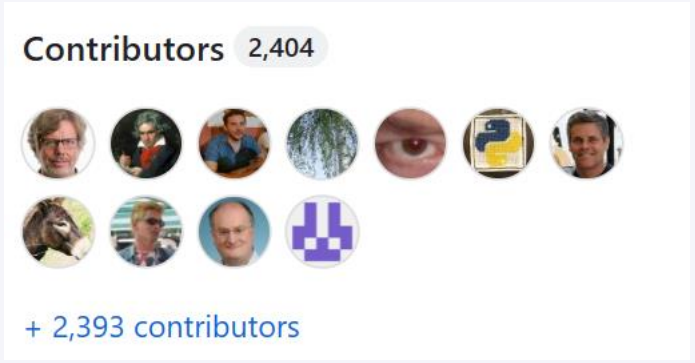
Unicode для строк, аннотация типов, функция print, новый синтаксис.

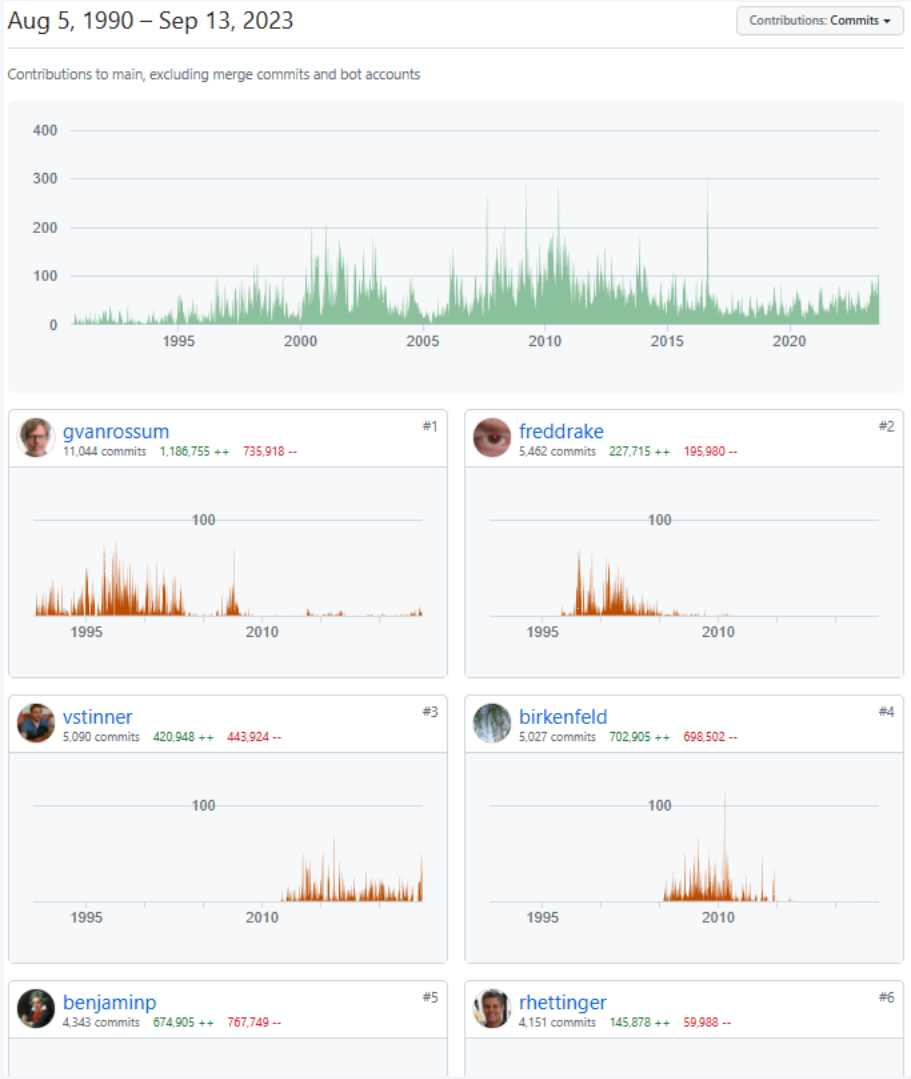
График выхода новых версий



Источник: devguide.python.org

Кто развивает Python сегодня





Python Software Foundation FAQ

Тренды развития Python

- Увеличение производительности в 5 раз и более (faster-cpython)
- Уход от GIL (PEP-554, Multiple Interpreters)
- Улучшение типизации (PEP-695 and many others)
- ... и многое другое!

Python Enhancement Proposal (PEP)

Contents

Index by Category

Meta-PEPs (PEPs about PEPs or Processes)

	PEP	Title	Authors
PA	1	PEP Purpose and Guidelines	Barry Warsaw, Jeremy Hylton, David Goodger, Nick Coghlan
PA	2	Procedure for Adding New Modules	Brett Cannon, Martijn Faassen
PA	4	Deprecation of Standard Modules	Brett Cannon, Martin von Löwis
PA	5	Guidelines for Language Evolution	Paul Prescod
PA	6	Bug Fix Releases	Aahz, Anthony Baxter
PA	7	Style Guide for C Code	Guido van Rossum, Barry Warsaw
PA	8	Style Guide for Python Code	Guido van Rossum, Barry Warsaw, Nick Coghlan
PA	10	Voting Guidelines	Barry Warsaw
PA	11	CPython platform support	Martin von Löwis, Brett Cannon
PA	12	Sample reStructuredText PEP Template	David Goodger, Barry Warsaw, Brett Cannon
PA	13	Python Language Governance	The Python core team and community
PA	387	Backwards Compatibility Policy	Benjamin Peterson
PA	581	Using GitHub Issues for CPython	Mariatta
PA	609	Python Packaging Authority (PyPA) Governance	Dustin Ingram, Pradyun Gedam, Sumana Harihareswara
PA	676	PEP Infrastructure Process	Adam Turner

Источник: Index of Python Enhancement Proposals (PEPs) | peps.python.org

Пример PEP: Type Parameter Syntax

- PEP-695 Предлагает новый синтаксис для Generic Types
- Упростит использование typing.TypeVar
- Изменения приняты к реализации
- Новый синтаксис будет добавлен в версии python 3.12

Пример PEP: Type Parameter Syntax

Старый синтаксис:

```
from typing import Generic, TypeVar

_T_co = TypeVar("_T_co", covariant=True, bound=str)

class ClassA(Generic[_T_co]):
    def method1(self) -> _T_co:
        ...
```

Новый синтаксис:

```
class ClassA[T: str]:
    def method1(self) -> T:
        ...
```

Этапы запуска программы

> python main.py

1. Перевод исходного кода в лексемы (токенизация).
2. Построение AST.
3. Создания байт-кода по AST.
4. Выполнение байт-кода

Пример токенизации

```
from tokenize import tokenize
from io import BytesIO
from token import tok_name

code_string = 'print(222*555)'

tokens = tokenize(BytesIO(code_string.encode(
'utf-8')).readline)  # tokenize the string

pprint([( token.string, tok_name[token.type])
 for token in tokens])
```

```
[
('utf-8', 'ENCODING'),
('print', 'NAME'),
('(', 'OP'),
('222', 'NUMBER'),
('*', 'OP'),
('555', 'NUMBER'),
(')', 'OP'),
('', 'NEWLINE'),
('', 'ENDMARKER')
]
```

Abstract Syntax Trees | docs.python.org

Пример построения AST

```python
import ast

tree = ast.parse("x, y = y, x")
print(ast.dump(tree, indent=4))
```

```
Module(
    body=[
        Assign(
            targets=[
                Tuple(
                    elts=[
                        Name(id='x', ctx=Store()),
                        Name(id='y', ctx=Store())],
                    ctx=Store())],
            value=Tuple(
                elts=[
                    Name(id='y', ctx=Load()),
                    Name(id='x', ctx=Load())],
                ctx=Load()))],
    type_ignores=[])
```

Abstract Syntax Trees | docs.python.org

Пример компиляции в байткод

```python
import dis

def modulus(x, y):
    result = x % y
    return result

dis.dis(modulus)
```

Номер строки	Номер байта	Название инструкции	Индекс аргумента	Название аргумента
4	0	LOAD_FAST	0	(x)
	2	LOAD_FAST	1	(y)
	4	BINARY_MODULO		
	6	STORE_FAST	2	(result)
5	8	LOAD_FAST	2	(result)
	10	RETURN_VALUE		

Disassembler for Python bytecode | docs.python.org

Компоненты CPython

1. **Runtime** (GIL, Mem. Alloc.)
2. **Interpreter** (A group of threads and their data)
3. **Thread** (data specific to a single OS thread; includes the call stack)
4. **Frame** (an element of a call stack, contains a code object, provides a state to execute it)
5. **Evaluation loop** (place where a frame is executed)

Python: behind the scene | tenthousandmeters.com

Пример: структура CodeObject

Содержит код без контекста

Некоторые поля:

- `co_argcount`: количество аргументов к блоку
- `co_code`: байткод
- `co_consts`: константы (берутся через LOAD_CONST)
- `co_names`: не-локальные переменные
- `co_varnames`: локальные переменные

```
typedef struct {
 PyObject_HEAD
 /* ... */
 PyObject *co_code;      /* instruction opcodes */

 PyObject *co_consts;    /* list (constants used) */
 PyObject *co_names;     /* list of strings(names used)*/
 PyObject *co_varnames; /* local variable names */
 /* ... */
} PyCodeObject;
```

Пример: структура FrameObject

Определяет контекст выполнения

Некоторые поля:

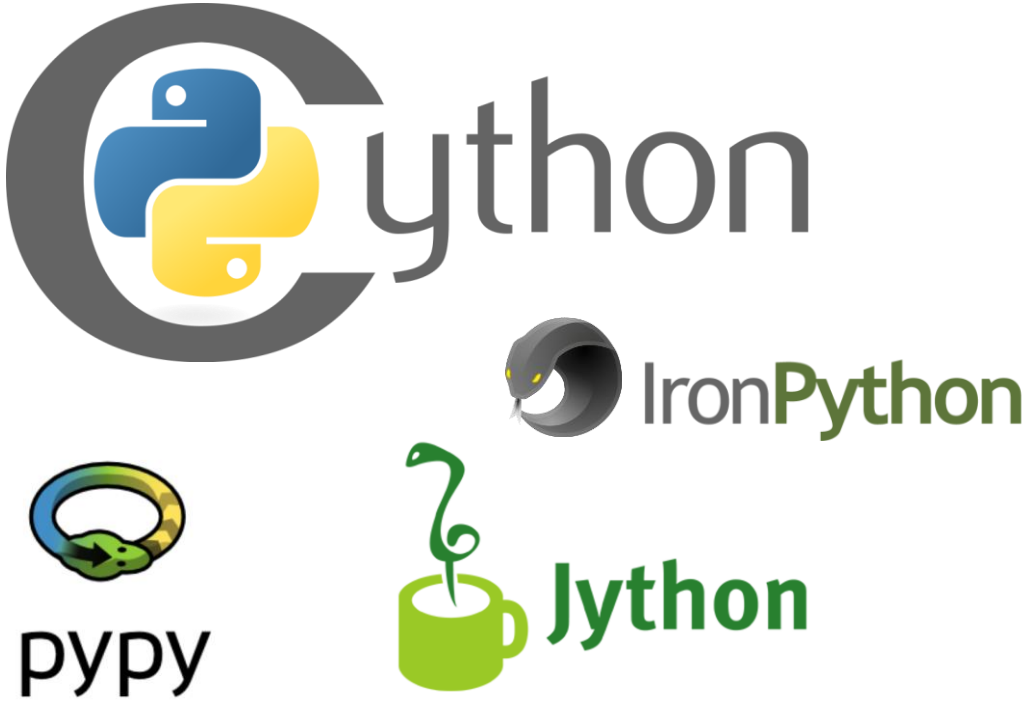
- `f_back`: ссылка на предыдущий фрейм (используется, например, для построения стека вызовов).
- `f_code`: ссылка на code object
- `f_builtins`: ссылка на встроенный неймспейс
- `f_globals`: ссылка на глобальные переменные
- `f_locals`: локальные переменные
- `f_valuestack`: ссылка на стек выполнения

```
typedef struct _frame {
    PyObject_VAR_HEAD           /* тоже PyObject! */

    PyCodeObject *f_code;       /* code segment */
    PyObject *f_builtins;       /* builtin symbol table (PyDictObject) */
    PyObject *f_globals;        /* global symbol table (PyDictObject) */
    PyObject *f_locals;         /* local symbol table (any mapping) */
    PyObject **f_valuestack;    /* points after the last local */

    /* ... */
    int f_lasti;                /* Last instruction if called */
} PyFrameObject;
```

Рекомендую доклад “Устройство CPython“ от Яндекса

Альтернативные реализации Python

- IronPython (C#)
- Jython (Java)
- PyPy (Python)
- Cython (Си)



Python 3.6

f-strings (PEP 498)

```
import math

radius = 1.2
length = 2 * math.pi * radius
f"Length of the circle with the radius {radius} = {length:.2f}"
```

Typing module

```
primes: List[int] = []
captain: str  # Note: no initial value!
class Starship:
    stats: Dict[str, int] = {}
```

Python 3.6

Path module

```
from pathlib import Path
path = str(Path("/") / Path("tmp") / Path("demo.tmp"))
with open(path, "w") as f:
    f.write("Hello, Path!")
```

Python 3.7

Dataclasses (PEP 557)

```python
from dataclasses import dataclass



@dataclass(order=True)
class User:
    name: str
    age: int
```

Python 3.8

Assignment expression “the walrus operator” (PEP 572)

```
# Loop over fixed length blocks
while (block := f.read(256)) != '':
    process(block)
```

```
if (n := len(a)) > 10:
    print(f"List is too long ({n} elements, expected <= 10)")
```

Python 3.8

Positional only parameters

```
def strlen(obj: str, /):
    c = 0
    for _ in obj:
        c += 1
    return c

strlen("hello")
strlen(obj='hello')  # The "obj" keyword argument impairs readability
```

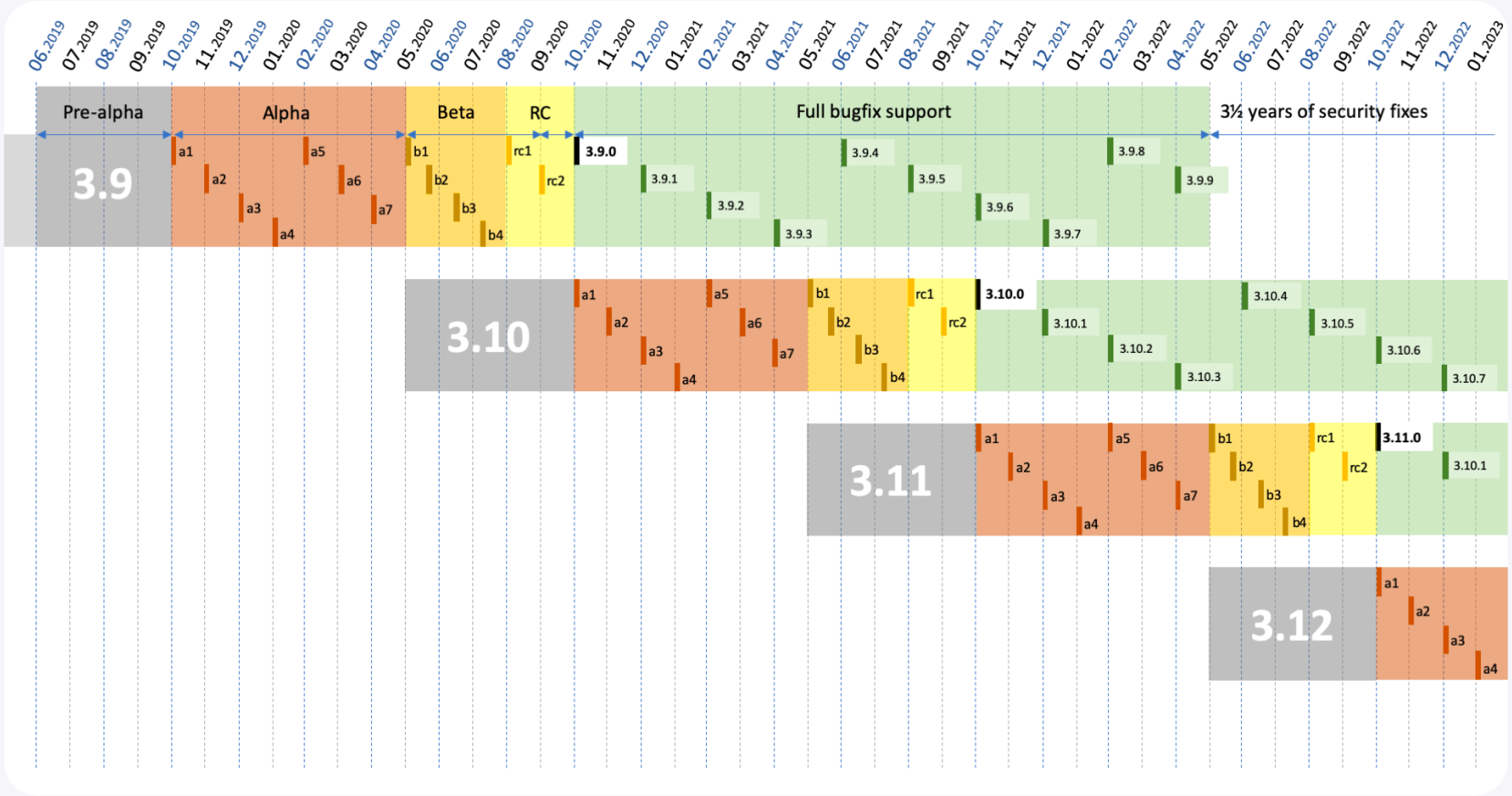
Python 3.8

An “=“ specifier in f-strings

```
import math

radius = 1.2
length = 2 * math.pi * radius
f"A circle with the {radius=} has a length {length=:.2f}"
```

Python 3.9

Регулярный график обновления версий (PEP 602)



Python 3.9

Dictionary Merge operator (PEP 584)

```
dict1 = {"x": "1", "y": "1"}
dict2 = {"y": "2", "z": "2"}

print(dict1 | dict2)    # {'x': '1', 'y': '2', 'z': '2'}
print(dict2 | dict1)    # {'y': '1', 'z': '2', 'x': '1'}
```

Python 3.9

Type hint generics from standard collection (PEP 585)

```
def greet_all(names: list[str]) -> None:
    for name in names:
        print("Hello", name)
```

Python 3.10

Structural pattern matching (PEP 634, 635, 636)

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something's wrong with the internet"
```

```
case 401 | 403 | 404:
    return "Not allowed"
```

Python 3.10

Writing Union types as X | Y

```
def add_5(value: int | float) -> int | float:
    return value + 5
```

Python 3.10

New Context manager syntax

```
from contextlib import redirect_stdout

with (open("file.txt", "w") as file, redirect_stdout(file)):
    ...
```

```
with (open("file1.txt", "r") as file1, open("file2.txt", "w") as file2):
    ...
```

Python 3.11

ExceptionGroup and a new exception syntax

```
try:
    raise ExceptionGroup("Data validations",(ValueError("problem 1"), ValueError("problem 2")))
except * (ValueError, ValueError) as exception_group_1:
    print("validations failed")
    raise ValueError from exception_group_1
```

Python 3.12

Type Parameter Syntax (PEP 695)

```
class Loadable(typing.Protocol):
    def load(self) -> None:
        ...


class Stack[T:Explainable]:
    ...


class Element:
    def load(self) -> str:
        ...

stack = Stack[Element]()
stack.push(Element())
```

Инструменты для отладки

- Debugging and Profiling
  - Audit events table
  - bdb — Debugger framework
  - faulthandler — Dump the Python traceback
  - pdb — The Python Debugger
  - The Python Profilers
  - timeit — Measure execution time of small code snippets
  - trace — Trace or track Python statement execution
  - tracemalloc — Trace memory allocations

Debugging and profiling | docs.python.org

Резюмируя

1. Разработчики уделяют все больше внимания оптимизации Cpython
2. Новые версии CPython в несколько раз быстрее и эффективнее
3. Обновления удобно планировать благодаря регулярному графику выхода новых версий
4. Обновления закрывают уязвимости
5. Появляется удобный и актуальный синтаксис, востребованный сообществом

EDGE ЦЕНТР

Спасибо за внимание!

Репозиторий с материалами лекции