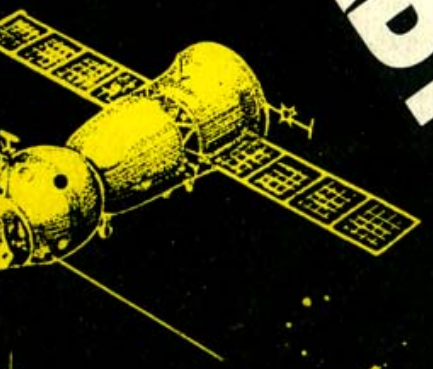
Н.А. Варваров

ПОПУЛЯРНАЯ КОСМОНАВТИКА



А. Романов
Л. Лебедев
Б. Лукьянов

Сыны голубой планеты

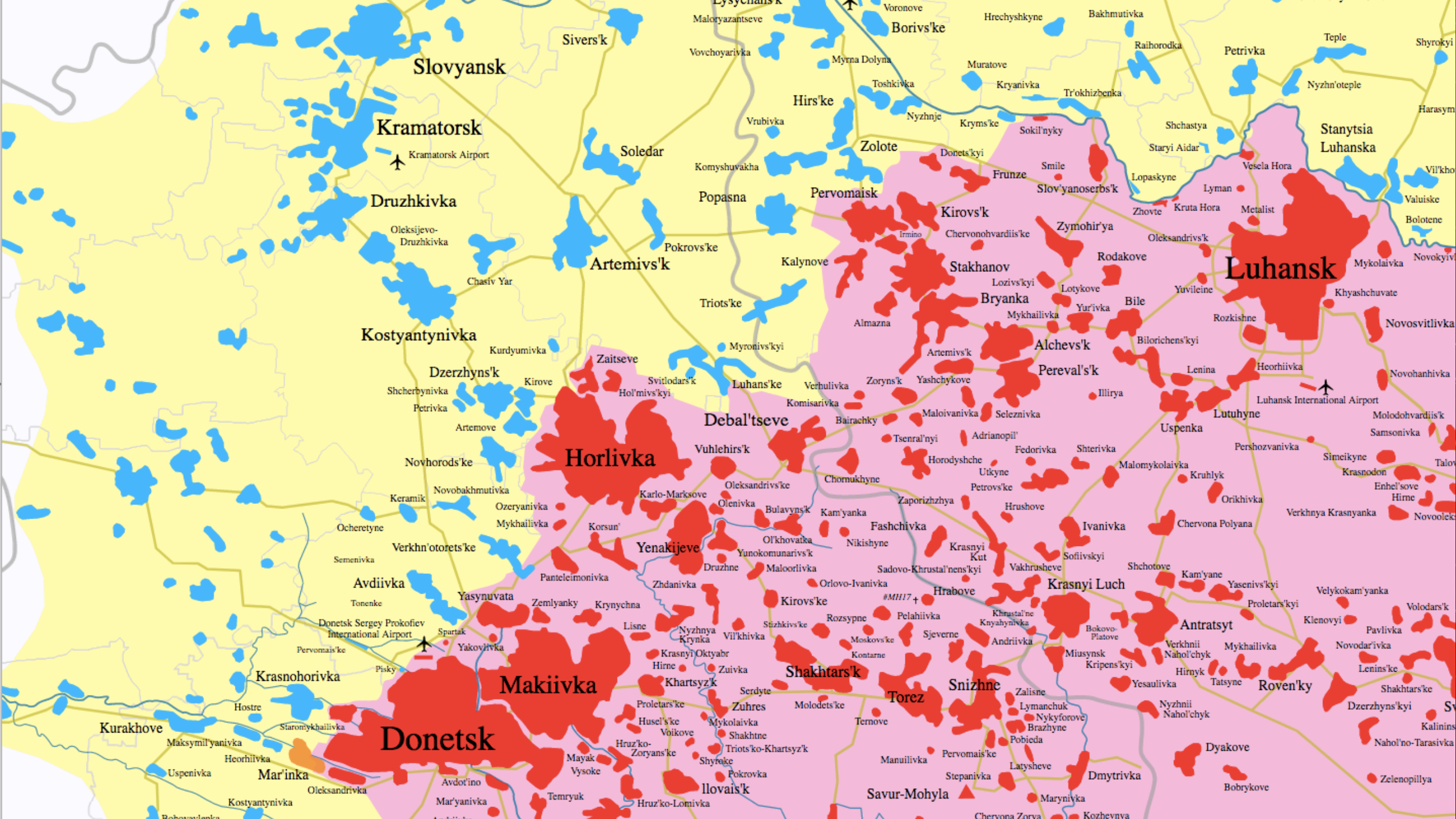
Нам, советским людям, строящим коммунизм, выпала честь первыми проникнуть в космос. Победы в освоении космоса мы считаем не только достижением нашего народа, но и всего человечества. Мы с радостью ставим их на службу всем народам, во имя прогресса, счастья и блага всех людей на Земле. Наши достижения и открытия мы ставим не на службу войне, а на службу миру и безопасности народов.

КОСМОС ДАЛЁКИЙ И БЛИЗКИ

РЕКОМЕ
БИБЛИО
УКАЗАТ

Облетев Землю в
корабле-спутнике, я увидел,
как прекрасна наша планета.
Люди, будем хранить и приумно-
жать эту красоту, а не разру-
шать её!

Гагарин


Slovyansk
Kramatorsk
Kramatorsk Airport
Druzhkivka
Oleksijevo-Druzhkivka
Sivers'k
Soledar
Artemivs'k
Pokrovs'ke
Chasiv Yar
Kostyantynivka
Kurdyumivka
Dzerzhyns'k
Kirove
Shcherbynivka
Petrivka
Artemove
Novhorods'ke
Novobakhmutivka
Keramik
Ozeryanivka
Mykhailivka
Ocheretyne
Verkhn'otorets'ke
Semenivka
Avdiivka
Tonenke
Yasynuvata
Donetsk Sergey Prokofiev International Airport
Spartak
Yakovlivka
Pervomais'ke
Pisky
Krasnohorivka
Hostre
Kurakhove
Staromykhailivka
Maksymil'yanivka
Heorhiivka
Uspenivka
Mar'inka
Oleksandrivka
Kostyantynivka
Donetsk
Makiivka
Horlivka
Zaitseve
Svitlodars'k
Hol'mivs'kyi
Debal'tseve
Luhans'ke
Vuhlehirs'k
Yenakijeve
Korsun'
Karlo-Marksove
Oleksandrivs'ke
Olenivka
Bulavyns'k
Kam'yanka
Ol'khovatka
Yunokomunarivs'k
Druzhne
Maloorlivka
Panteleimonivka
Zhdanivka
Zemlyanky
Krynychna
Lisne
Nyzhnya Krynka
Vil'khivka
Krasnyi Oktyabr
Hirne
Zuivka
Khartsyz'k
Proletars'ke
Husel's'ke
Voikove
Mykolaivka
Shakhtne
Triots'ko-Khartsyz'k
Hruz'ko-Zoryans'ke
Shyroke
Pokrovka
Mayak
Vysoke
Temryuk
Avdot'ino
Mar'yanivka
Hruz'ko-Lomivka
Ilovais'k
Popasna
Komyshuvakha
Vrubivka
Pervomaisk
Kalynove
Triots'ke
Myronivs'kyi
Zolote
Hirs'ke
Borivs'ke
Voronove
Maloryazantseve
Vovchoyarivka
Myrna Dolyna
Toshkivka
Nyzhnje
Kryms'ke
Kirovs'k
Irmino
Chervonohvardiis'ke
Stakhanov
Almazna
Bryanka
Lozivs'kyi
Mykhailivka
Alchevs'k
Artemivs'k
Pereval's'k
Zoryns'k
Yashchykove
Verhulivka
Komisarivka
Bairachky
Maloivanivka
Selezniv'ka
Tsenral'nyi
Adrianopil'
Horodyshche
Utkyne
Petrovs'ke
Chornukhyne
Zaporizhzhya
Hrushove
Fashchivka
Nikishyne
Krasnyi Kut
Sadovo-Khrustal'nens'kyi
Orlovo-Ivanivka
Hrabove
Kirovs'ke
Rozsypne
Pelahiivka
Stizhkivs'ke
Moskovs'ke
Kontarne
Sjeverne
Andriivka
Shakhtars'k
Torez
Snizhne
Zalisne
Lymanchuk
Nykyforove
Brazhyne
Pobieda
Molodets'ke
Ternove
Zuhres
Serdyte
Manuilivka
Pervomais'ke
Stepanivka
Latysheve
Savur-Mohyla
Marynivka
Chervona Zorya
Kozhevnya
Dmytrivka
Dyakove
Bobrykove
Zelenopillya
Nahol'no-Tarasivka
Dzerzhyns'kyi
Kalinins
Shakhtars'ke
Roven'ky
Tatsyne
Hirnyk
Nyzhnii Nahol'chyk
Yesaulivka
Lenins'ke
Novodar'ivka
Mykhailivka
Verkhnii Nahol'chyk
Antratsyt
Bokovo-Platove
Miusynsk
Kripens'kyi
Velykokam'yanka
Volodars'k
Klenovyi
Pavlivka
Proletars'kyi
Yasenivs'kyi
Kam'yane
Shchotove
Krasnyi Luch
Vakhrusheve
Sofiivskyi
Ivanivka
Chervona Polyana
Verkhnya Krasnyanka
Novoolek
Krasnodon
Enhel'sove
Hirne
Orikhivka
Kruhlyk
Malomykolaivka
Shterivka
Fedorivka
Pershozvanivka
Simeikyne
Talov
Uspenka
Lutuhyne
Molodohvardiis'k
Samsonivka
Luhansk International Airport
Heorhiivka
Lenina
Novohanhivka
Illirya
Selezniv'ka
Luhansk
Mykolaivka
Novokyivi
Khyashchuvate
Novosvitlivka
Yuvileine
Bilorichens'kyi
Rozkishne
Bile
Yur'ivka
Lotykove
Rodakove
Zymohir'ya
Oleksandrivs'k
Zhovte
Kruta Hora
Metalist
Lyman
Vesela Hora
Lopaskyne
Slov'yanoserbs'k
Smile
Frunze
Donets'kyi
Sokil'nyky
Shchastya
Staryi Aidar
Stanytsia Luhanska
Valuiske
Bolotene
Vil'kho
Harasym
Nyzhn'oteple
Teple
Shyrokyi
Petrivka
Raihorodka
Bakhmutivka
Hrechyshkyne
Muratove
Kryanivka
Tr'okhizbenka

WAR IS STUPID
SPACE
IS WONDERFUL

GRADUAL TYPING

OF PRODUCTION APPLICATIONS

ŁUKASZ LANGA

ŁUKASZ LANGA

ambv @ #python

fb.me/ambv

@llanga

lukasz@langa.pl

WHAT WE'RE GOING TO TALK ABOUT

Intro to PEP 484

- Why annotate?
- What's the syntax?
- Types are more than classes
- The typing module
- Generics
- Type inference
- Runtime errors are hard to debug

Gradual typing

- What's the problem?
- What tools exist?
- Where to start?
- Continuous integration
- Fighting regressions
- Metrics of success
- Typeshed

Common gotchas

- Which functions are typed
- Unions
- Out of order definitions
- Invariance
- Self-types
- Bugs in tooling

WHY ANNOTATE PYTHON CODE WITH TYPES?

WHY ANNOTATE PYTHON CODE WITH TYPES?

```
def process_all(items):
    for item in items:
        item.children.process()
```

WHY ANNOTATE PYTHON CODE WITH TYPES?

- Help readers **understand** the code at hand

WHY ANNOTATE PYTHON CODE WITH TYPES?

- Help readers **understand** the code at hand
- **Validate assumptions** programmers make while writing

WHY ANNOTATE PYTHON CODE WITH TYPES?

- Help readers **understand** the code at hand
- **Validate assumptions** programmers make while writing
- **Minimize amount of state** developers need to keep in their heads

WHY ANNOTATE PYTHON CODE WITH TYPES?

- Help readers **understand** the code at hand
- **Validate assumptions** programmers make while writing
- **Minimize amount of state** developers need to keep in their heads
- **Find existing bugs, prevent future ones**

```
def f(argument):
    ...
```

```
def f(argument: T) -> R:
    ...
```

```
def f(argument: T) -> R:
    ...
```

```
def f(argument: str)
def f(argument: int)
def f(argument: list)
def f(argument: MediaType)
...
```

```
def f(argument: T) -> R:
    ...
```

```
def f() -> str: ...
def f() -> int: ...
def f() -> None: ...
```

Atom File Edit View Selection Find Packages Nuclide Window

example.py

```python
def gcd(a: int, b: int) -> int:
    while b:
        a, b = b, a % b
    return a


print(gcd(10, 15))

```

Diagnostics

Errors: 0 Warnings: 0

Type	Source	Line	Description

Atom File Edit View Selection Find Packages Nuclide Window

example.py

```python
def gcd(a: int, b: int) -> int:
    while b:
        a, b = b, a % b
    return a


print(gcd('x', 15))
```

Diagnostics

Errors: 0 Warnings: 1

Type	Source	Line	Description
Warning	flake8: T484	7	Argument 1 to "gcd" has incompatible type "str"; expected "int"

TYPE CHECK ERROR VS. RUNTIME ERROR

Type error:

```
.local/example.py:7:6: error: Argument 1 to "gcd" has incompatible type
                              "str"; expected "int"
```

Runtime error:

```
Traceback (most recent call last):
  File ".local/example.py", line 7, in <module>
    print(gcd('x', 15))
  File ".local/example.py", line 3, in gcd
    a, b = b, a % b
TypeError: not all arguments converted during string formatting
```

Atom File Edit View Selection Find Packages Nuclide Window

example.py

```python
class Point:
    def __init__(self, x: int, y: int) -> None:
        self.x = x
        self.y = y


p = Point(0, 1)
print(p.x, p.y, p.z)
reveal_type(p.x)

```

Diagnostics

Errors: 0 Warnings: 2

Type	Source	Line	Description
Warning	flake8: T484	8	"Point" has no attribute "z"
Warning	flake8: T484	9	Revealed type is 'builtins.int'

THERE'S MORE TO TYPES THAN JUST CLASSES

THERE'S MORE TO TYPES THAN JUST CLASSES

- What if a function returns **anything**?
- What if a function only accepts **lists with string elements**?
- What if a function accepts a string **or** None?
- What if a function accepts instances of TypeA **or** TypeB?
- What if a function returns a value of the **same type** as passed as an argument?
- What if a function accepts an int but it's really expecting **only fbids**?
- What if a function accepts a class, not an object?

WHAT IF A FUNCTION RETURNS ANYTHING?

```
from typing import Any
def f() -> Any: ...
```

WHAT IF A FUNCTION ONLY ACCEPTS LISTS WITH STRING ELEMENTS?

```
from typing import List
def f(l: List[str]) -> None:
    ...
```

WHAT IF A FUNCTION ACCEPTS A STRING OR NONE?

```
from typing import Optional
def f(s: Optional[str]) -> None:
    ...
```

WHAT IF A FUNCTION ACCEPTS A STRING OR NONE?

```
from typing import Optional
def f(s: Optional[str]) -> None:
    ...
def g(s: str = None) -> None:
    ...
```

WHAT IF A FUNCTION ACCEPTS TYPE A OR TYPE B?

```
from typing import Union
def f(s: Union[A, B]) -> None:
    ...
```

WHAT IF A FUNCTION RETURNS A VALUE OF THE SAME TYPE AS PASSED AS AN ARGUMENT?

```
from typing import TypeVar
T = TypeVar('T')
def f(s: T) -> T:
    ...
```

WHAT IF A FUNCTION RETURNS A VALUE OF THE SAME TYPE AS PASSED AS AN ARGUMENT?

```
from typing import List, TypeVar
T = TypeVar('T')
def f(s: List[T]) -> T:
    ...
```

WHAT IF A FUNCTION ACCEPTS AN INT BUT IT'S REALLY EXPECTING ONLY FBIDS?

```
from typing import NewType
FBID = NewType('FBID', int)
def unixname(id: FBID) -> str:
    ...
```

WHAT IF A FUNCTION ACCEPTS A CLASS, NOT AN OBJECT?

```
from typing import Type


def factory(
    t: Type[django.Model]
) -> django.Model:
    ...
```

WHAT IF A FUNCTION ACCEPTS A CLASS, NOT AN OBJECT?

```
from typing import TypeVar, Type
M = TypeVar('M', bound=django.Model)

def factory(
    t: Type[M]
) -> M:
    ...
```

TYPE INFERENCE

TYPE INFERENCE

- The first assignment to a variable defines its type

```
i = 1              # reveal_type(i) -> int
l = ['a', 'b']     # reveal_type(l) -> list[str]
```

TYPE INFERENCE

- The first assignment to a variable defines its type

```
i = 1              # reveal_type(i) -> int
l = ['a', 'b']     # reveal_type(l) -> list[str]
```

- If this assignment is an empty collection or None, inference fails

```
i = None           # reveal_type(i) -> None
l = []             # error: need type annotation for variable
                   # reveal_type(l) -> Any
```

TYPE INFERENCE

- We need generics for collections to infer what elements they contain

```
def f(l: List[str]) -> None:
    s = l.pop() # reveal_type(s) -> str
```

TYPE INFERENCE

- We need generics for collections to infer what elements they contain

```
def f(l: List[str]) -> None:
    s = l.pop() # reveal_type(s) -> str
```

- We can improve inference by constraining the flow paths of the function

```
def f(message: str = None) -> None:
    if message is None:
        return
    reveal_type(message)  # -> str
```

- Same goes for using `assert isinstance(...)` and similar

TYPE INFERENCE

- We can use variable annotations for the non-obvious cases

```
def f() -> None:
    l: List[str] = []
    m: str   # no need for initial value
```

- This requires Python 3.6
- In Python 3.5 and older, you had to use type comments for the same effect

```
def f() -> None:
    l = []  # type: List[str]
    m = ... # type: str     # `...` means no initial value
```

TYPE INFERENCE

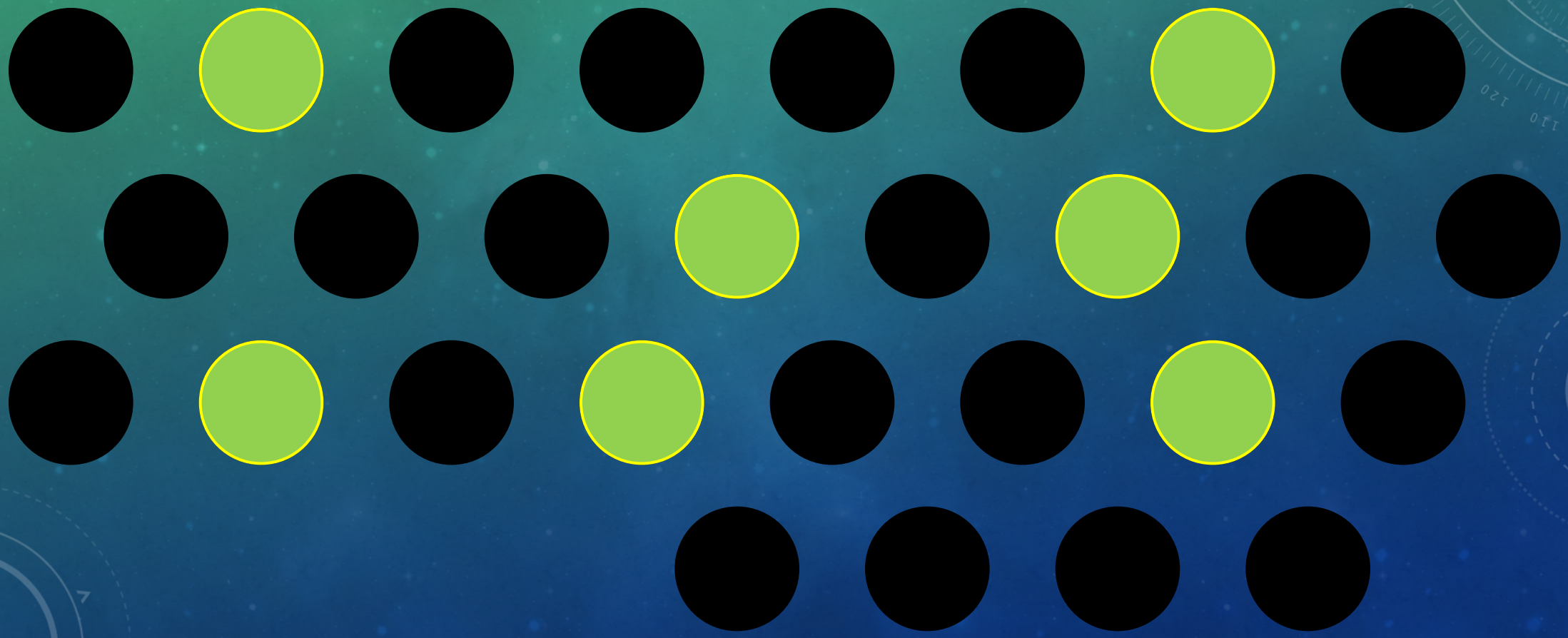
- When everything else fails, you can cast:

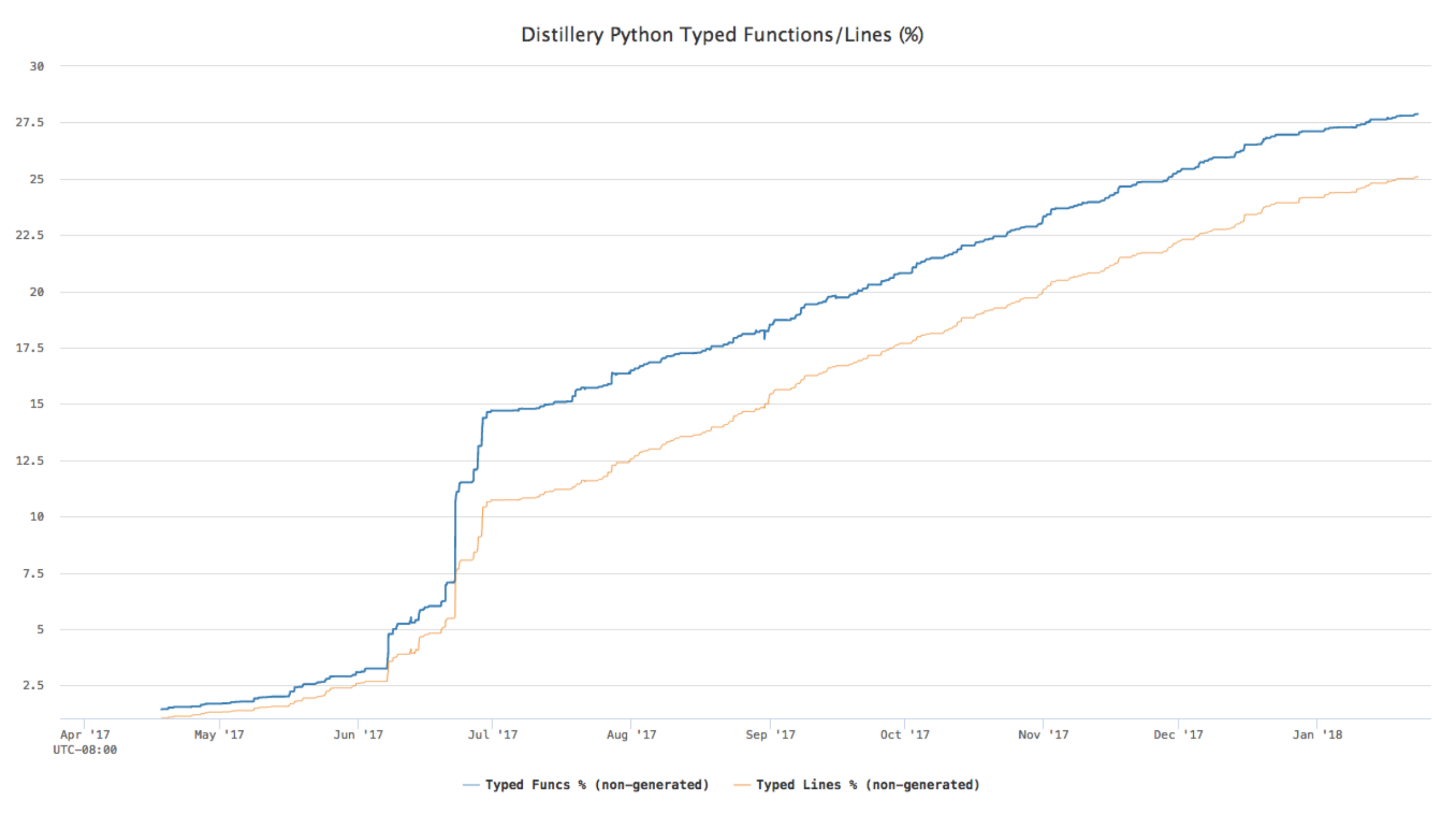
```
from typing import cast

some_int = function_returning_ints()
fbid = cast(FBID, some_int)
```

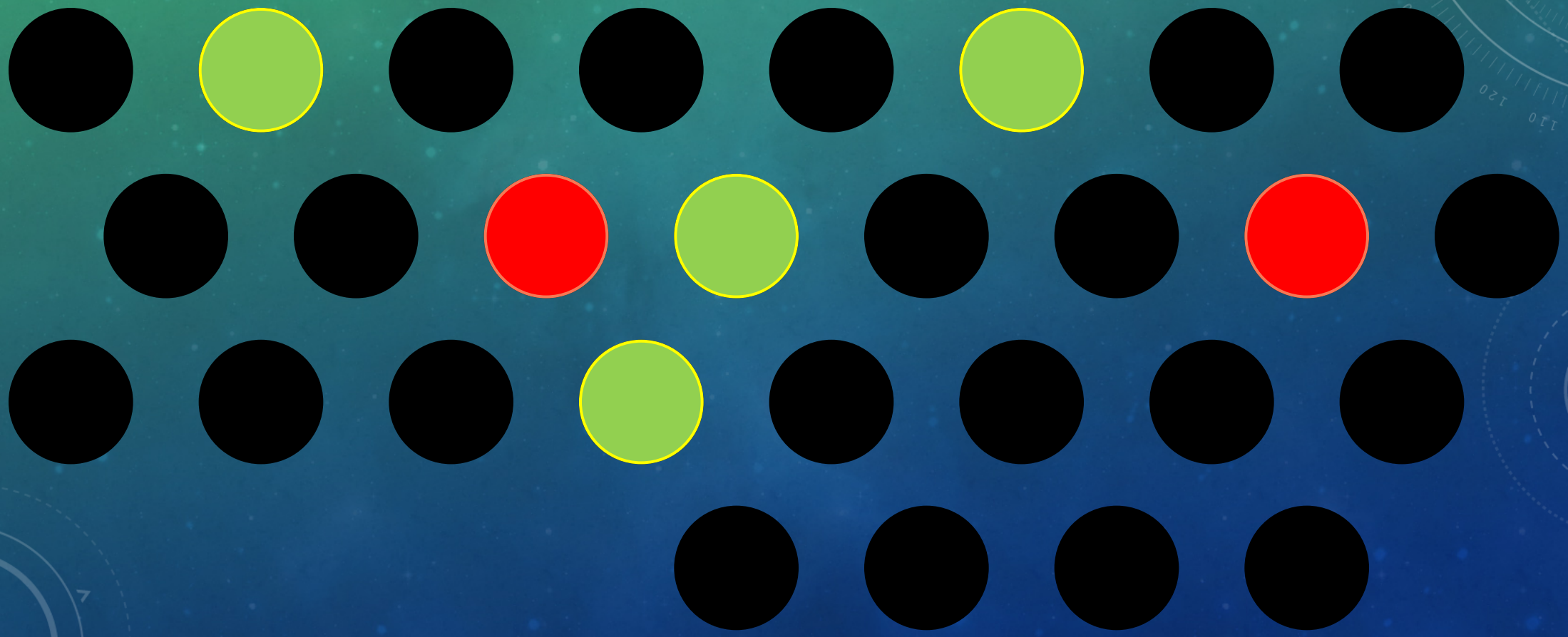
GRADUAL TYPING

GRADUAL TYPING




Distillery Python Typed Functions/Lines (%)
30
27.5
25
22.5
20
17.5
15
12.5
10
7.5
5
2.5
Apr '17
UTC-08:00
May '17
Jun '17
Jul '17
Aug '17
Sep '17
Oct '17
Nov '17
Dec '17
Jan '18
Typed Funcs % (non-generated)
Typed Lines % (non-generated)

GRADUAL TYPING



utils.py

```python
# from typing import List

def process_all():
    return process_strings("string1", "string2")

def process_strings(to_repeat, num_repeats):
    """Sorts and repeats the strings."""
    return process_input(to_repeat) * num_repeats

def process_input(input):
    return input.sort()

```

Diagnostics

Errors: 0 Warnings: 0

Current file only

Open All

Type	Source	File	Line	Description

No diagnostic messages

0 0

1 deprecation Normal MagicPython master

utils.py

```python
from typing import List

def process_all():
    return process_strings("string1", "string2")

def process_strings(to_repeat, num_repeats):
    """Sorts and repeats the strings."""
    return process_input(to_repeat) * num_repeats

def process_input(input: List[str]) -> List[str]:
    return input.sort()

```

Diagnostics

Errors: 1 Warnings: 0

Current file only | Open All

Type	Source	File	Line	Description
Err...	flake8: T484	utils.py	11	"sort" of "list" does not return a value

1 0 | 1 deprecation | Normal | MagicPython | master

utils.py

```python
from typing import List

def process_all() -> None:
    return process_strings("string1", "string2")

def process_strings(to_repeat, num_repeats):
    """Sorts and repeats the strings."""
    return process_input(to_repeat) * num_repeats

def process_input(input: List[str]) -> List[str]:
    return input.sort()

```

Diagnostics

Errors: 1 Warnings: 0

Current file only | Open All

Type	Source	File	Line	Description
Err...	flake8: T484	utils.py	11	"sort" of "list" does not return a value

1 0 1 deprecation Normal MagicPython master

utils.py

```python
from typing import List

def process_all() -> None:
    return process_strings("string1", "string2")

def process_strings(to_repeat: List[str], num_repeats: int) -> List[str]:
    """Sorts and repeats the strings."""
    return process_input(to_repeat) * num_repeats

def process_input(input: List[str]) -> List[str]:
    return input.sort()

```

Diagnostics

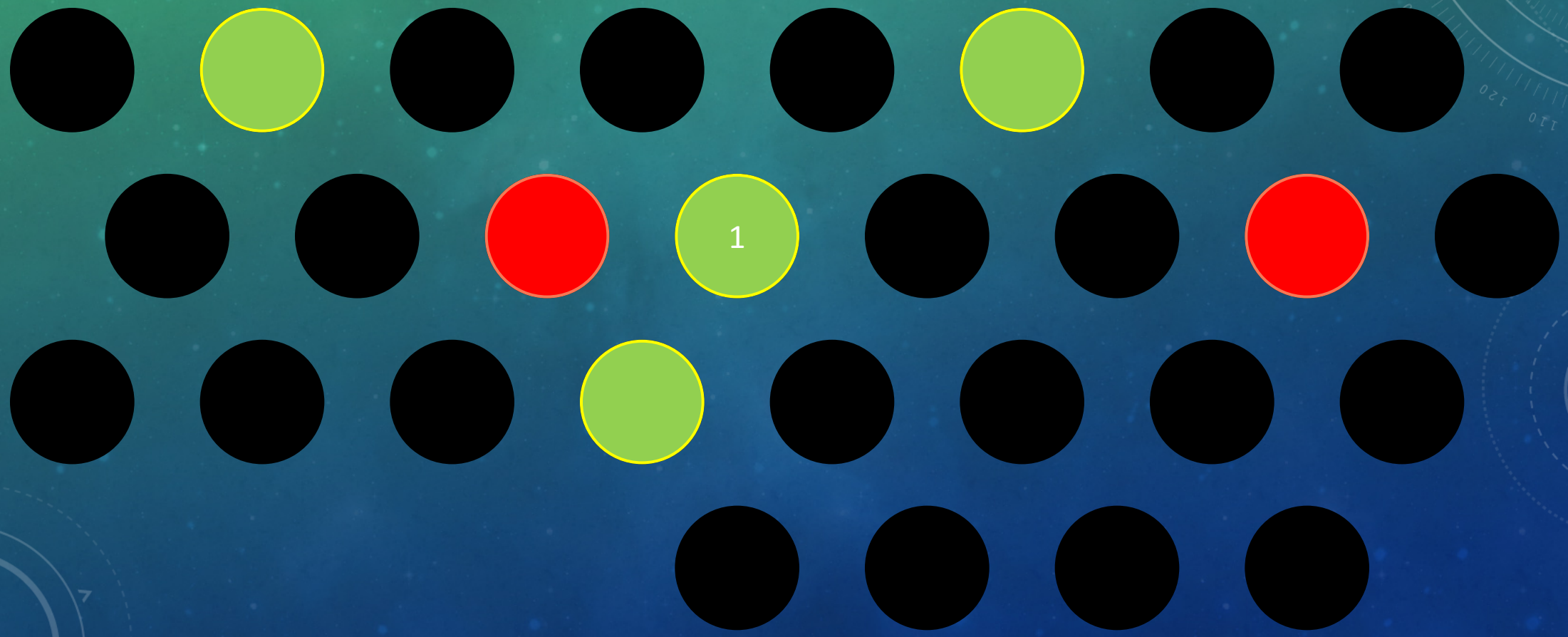
Errors: 4 Warnings: 0

Current file only | Open All

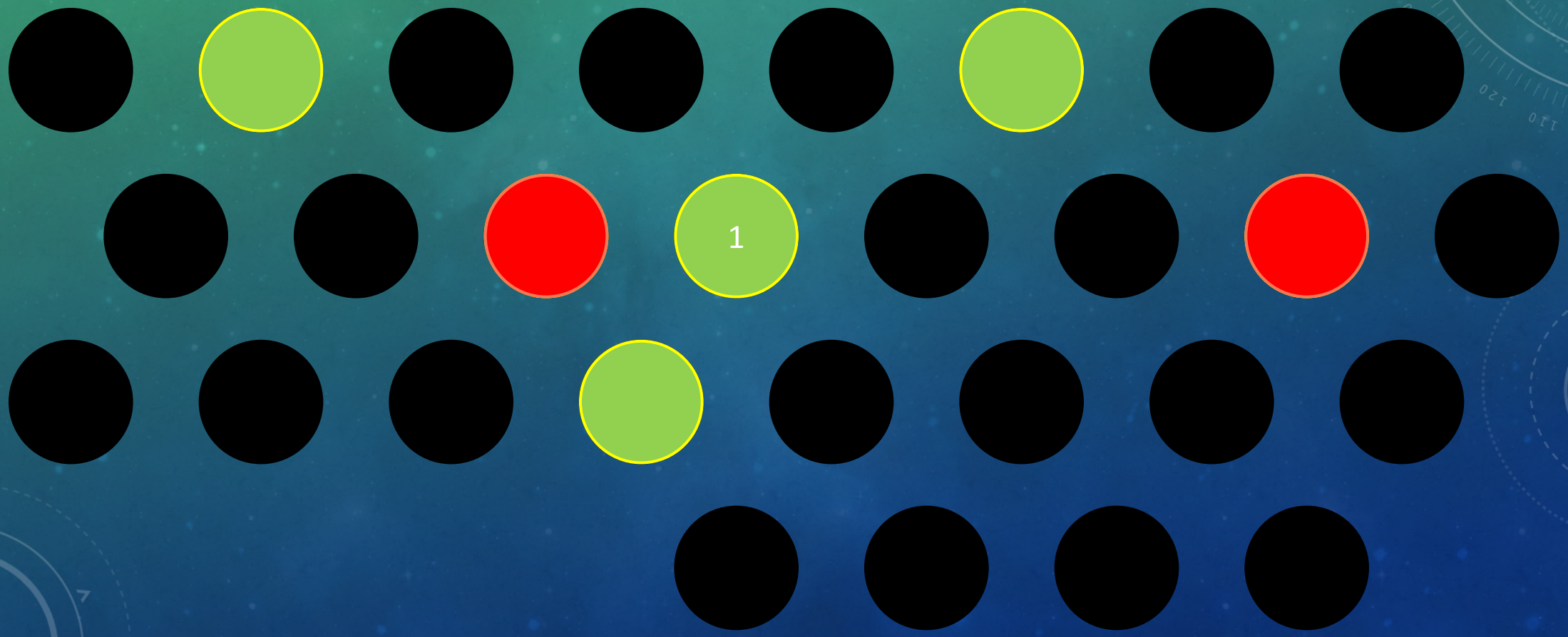
Type	Source	File	Line	Description
Err...	flake8: T484	utils.py	4	No return value expected
Err...	flake8: T484	utils.py	4	Argument 1 to "process_strings" has incompatible type "str"; expected List[str]
Err...	flake8: T484	utils.py	4	Argument 2 to "process_strings" has incompatible type "str"; expected "int"

4 0 1 deprecation Normal MagicPython master

GRADUAL TYPING



GRADUAL TYPING



WORKFLOW SUGGESTION #1

Find the most critical functions and start typing those first.

WORKFLOW SUGGESTION #2

Enable file-level linter type checking early.

FILE-LEVEL VS. FULL PROGRAM ANALYSIS

- You will use types and functions from other modules, too, by importing them.

```
# a.py

from b import B
from f import function
```

```
# z.py

from f import function
import g
```

```
# b.py

from base import Model
from f import function
```

```
# f.py

def f():
    ...
```

```
# g.py

from base import Model
from utils import x
```

```
# base.py

class Model:
    ...
```

```
# utils.py
```

FILE-LEVEL VS. FULL PROGRAM ANALYSIS

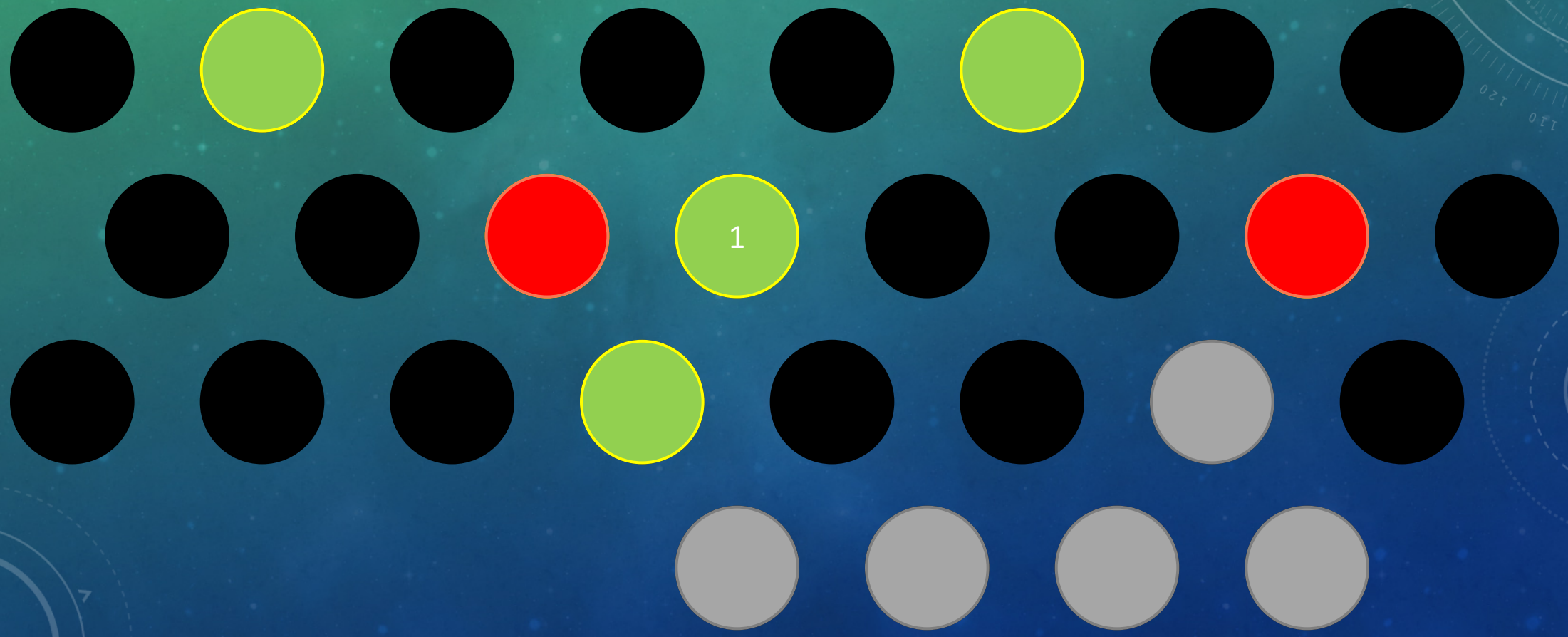
- You will use types and functions from other modules, too, by importing them.
- **mypy** performs full program analysis

```
$ pip install mypy
```

FILE-LEVEL VS. FULL PROGRAM ANALYSIS

- You will use types and functions from other modules, too, by importing them.
- **mypy** performs full program analysis *which makes it slow (so far)*
- If mypy cannot find an import or is configured to ignore imports, it assumes Any types everywhere.
- This hides usage errors: imported classes and functions are all treated as Any.
- **flake8-mypy** does this intentionally *to make checks fast*

GRADUAL TYPING



api.pyi

```python
# Stubs for requests.api (Python 3)
from typing import Optional, Union, Any, Iterable, Mapping, MutableMapping, Tuple, IO, T

from .models import Response

_ParamsMappingValueType = Union[Text, bytes, int, float, Iterable[Union[Text, bytes, int
_Data = Union[None, bytes, MutableMapping[Text, Text], IO]

def request(method: str, url: str, **kwargs) -> Response: ...
def get(url: Union[Text, bytes],
        params: Optional[
            Union[Mapping[Union[Text, bytes, int, float], _ParamsMappingValueType],
                  Union[Text, bytes],
                  Tuple[Union[Text, bytes, int, float], _ParamsMappingValueType],
                  Mapping[Text, _ParamsMappingValueType],
                  Mapping[bytes, _ParamsMappingValueType],
                  Mapping[int, _ParamsMappingValueType],
                  Mapping[float, _ParamsMappingValueType]]] = None,
        **kwargs) -> Response: ...
def options(url: Union[str, Text], **kwargs) -> Response: ...
def head(url: Union[str, Text], **kwargs) -> Response: ...
def post(url: Union[str, Text], data: Data = ..., json: Optional[MutableMapping] =
```

0 0 1 deprecation Normal MagicPython master

WORKFLOW SUGGESTION #3

Enable continuous integration to fight regressions.

GRADUAL TYPING

- Only annotated functions are type checked.
- A clean run of the type checker doesn’t prove there’s no errors.
- New annotations can discover errors in other functions.
- flake8-mypy only presents type errors related to the current file and the standard library.
- Do full codebase checks with mypy as part of continuous integration.

WORKFLOW SUGGESTION #4

Measure function coverage and the number of TypeError/AttributeError exceptions in production.

GOTCHAS

ONLY ANNOTATED FUNCTIONS ARE TYPED

```
def gcd(a: int, b: int) -> int:
    while b:
        a, b = b, a % b
    return a

def f():
    a = gcd(1, 2)
    reveal_type(a)  # -> Any!
```

ONLY ANNOTATED FUNCTIONS ARE TYPED

```
def convert_to_str(obj):
    return obj

def to_upper(exc: Exception) -> str:
    return convert_to_str(exc).upper() # no error!
```

At runtime:

```
AttributeError: 'ValueError' object has
                no attribute 'upper'
```

UNIONS LIMIT ALLOWED BEHAVIOR

```
def f(arg: Union[str, int]) -> None:
    lowercase_str = arg.lower()
    bitlength_of_int = arg.bit_length()
```

UNIONS LIMIT ALLOWED BEHAVIOR

```
def f(arg: Union[str, int]) -> None:
    lowercase_str = arg.lower()
    bitlength_of_int = arg.bit_length()
```

```
:2:20: error: Some element of union has no
attribute "lower"
:3:23: error: Some element of union has no
attribute "bit_length"
```

UNIONS LIMIT ALLOWED BEHAVIOR

```
def template() -> Union[bytes, str]:
    return "s”

def g(arg: str) -> str:
    return template().format(arg)
```

UNIONS LIMIT ALLOWED BEHAVIOR

```
def template() -> Union[bytes, str]:
    return "s”

def g(arg: str) -> str:
    return template().format(arg)

:5:11: error: Some element of union has no
attribute "format"
```

OUT OF ORDER DEFINITIONS

```
class A:
    def f() -> 'B':
        ...

class B:
    ...
```

OUT OF ORDER DEFINITIONS IN PYTHON 3.7

```
from __future__ import annotations

class A:
    def f() -> B:
        ...

class B:
    ...
```

IMPORT CYCLES

```
from typing import TYPE_CHECKING

if TYPE_CHECKING:
    from module import B

class A:
    def f() -> 'B':
        ...
```

IMPORT CYCLES

```
from typing import TYPE_CHECKING

if TYPE_CHECKING:
    from module import B  # noqa

class A:
    def f() -> 'B':
        ...
```

IMPORT CYCLES IN PYTHON 3.7

```
from __future__ import annotations
from typing import TYPE_CHECKING

if TYPE_CHECKING:
    from module import B

class A:
    def f() -> B:
        ...
```

IMPORT CYCLES IN PYTHON 3.7

MAYBE?

```
from __future__ import annotations


if __typing__:
    from module import B

class A:
    def f() -> B:
        ...
```

MUTABLE COLLECTIONS ARE INVARIANT

```
def f(l: List[Optional[str]]) -> None:
    ...

def g(l: List[str]) -> None:
    f(l)
```

MUTABLE COLLECTIONS ARE INVARIANT

```
def f(l: List[Optional[str]]) -> None:
    ...

def g(l: List[str]) -> None:
    f(l)  # error: arg 1 to f() incompatible type
          # List[str]; expected List[Optional[str]]
```

CRAZY IDEAS

- Static type inference
- Runtime type inference
- Runtime type *checking*
- Duck typing + type checking
  - See PEP 544: Protocols
- Performance optimizations

WHERE TO GET SUPPORT?

- Read first:
  - comprehensive mypy docs: http://mypy.readthedocs.io/en/latest/
- #typing on Gitter for real-time support
- For PEP 484/PEP 526/PEP 544/PEP 563 issues:
  - github.com/python/typing
- For type checker issues:
  - github.com/python/mypy
- For issues with standard library and third-party annotations:
  - github.com/python/typeshed

THANK YOU

ambv @ #python

fb.me/ambv

@llanga

lukasz@langa.pl

SLIDES: fb.me/gradual-typing