Writing Python like it's Rust

Writing Python like it's Rust

May 20, 2023

I started programming in Rust several years ago, and it has gradually changed t
in other programming languages, most notably in Python. Before I started using

Jakub Beránek

github.com/kobzol

@ jakub@berankovi.net

Jakub Beránek

github.com/kobzol

@ jakub@berankovi.net

- PhD student (52 days left), teacher @ VSB-TUO university

Jakub Beránek

github.com/kobzol

@ jakub@berankovi.net

- PhD student (52 days left), teacher @ VSB-TUO university
- Researcher @ IT4Innovations HPC center

Jakub Beránek

github.com/kobzol

@ jakub@berankovi.net

- PhD student (52 days left), teacher @ VSB-TUO university
- Researcher @ IT4Innovations HPC center
- Rust Project open-source contributor

Jakub Beránek

github.com/kobzol

@ jakub@berankovi.net

- PhD student (52 days left), te
- Researcher @ IT4Innovation
- Rust Project open-source co
  - Compiler performance

Compiler performance working group

Improving rustc compilation performance (build times)

Members

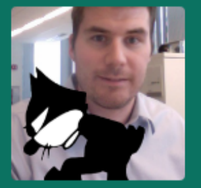
Mark Rousskov
GitHub: Mark-Simulacrum
Team leader

Jakub Beránek
GitHub: Kobzol

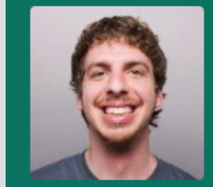
Rémy Rakic
GitHub: lqd

Michael Woerister
GitHub: michaelwoerister

Nicholas Nethercote
GitHub: nnethercote



Felix Klock
GitHub: pnkfelix



Ryan Levick
GitHub: rylev

Tyson Nottingham
GitHub: tgnottingham

Jakub Beránek

github.com/kobzol

@ jakub@berankovi.net

- PhD student (52 days left), te
- Researcher @ IT4Innovation
- Rust Project open-source co
  - Compiler performance
  - Infrastructure

Infrastructure team

Managing the infrastructure supporting the Rust project itself, including CI, releases, bots, and metrics

Members

Jan David Nose

GitHub: jdno

Team leader

Jake Goulding

GitHub: shepmaster

Team leader

kennytm

GitHub: kennytm

Jakub Beránek

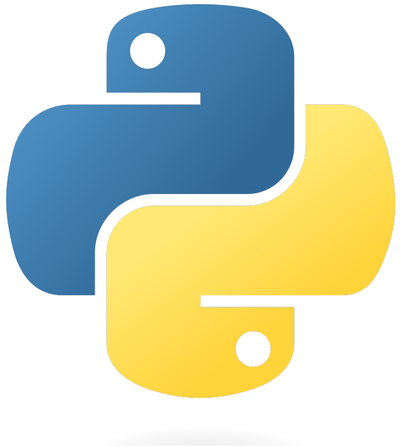
GitHub: Kobzol

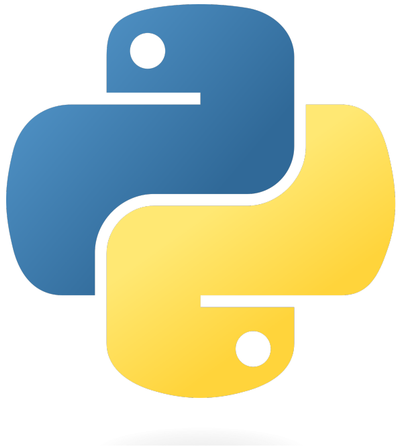
Mark Rousskov

GitHub: Mark-Simulacrum

Pietro Albini

GitHub: pietroalbini

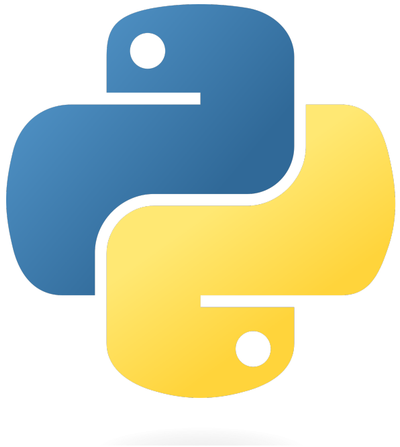




How I used to write Python:

How I used to write Python:

- No type hints



How I used to write Python:

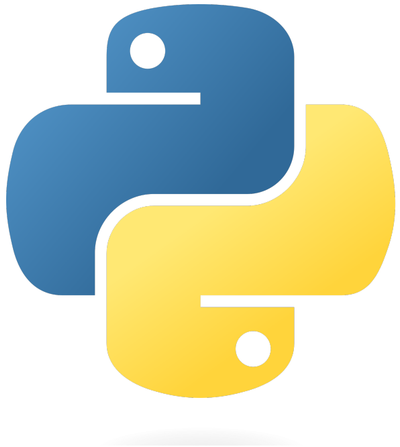
- No type hints
- Dictionaries everywhere

How I used to write Python:

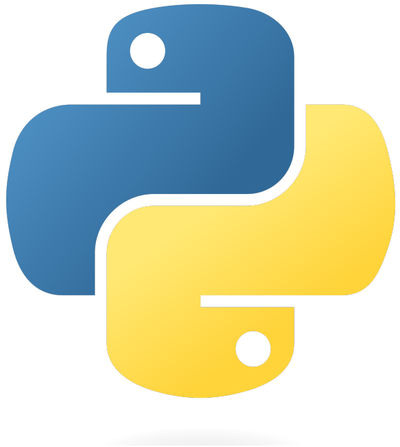
- No type hints
- Dictionaries everywhere
- "Stringly typed"

How I used to write Python:

- No type hints
- Dictionaries everywhere
- "Stringly typed"
- Monkey patching



Symptoms:



Symptoms:

- Easy to cause bugs

Symptoms:

- Easy to cause bugs
- (Too) many runtime crashes

Symptoms:

- Easy to cause bugs
- (Too) many runtime crashes
- Hard to understand

Symptoms:

- Easy to cause bugs
- (Too) many runtime crashes
- Hard to understand
- Difficult to refactor

Symptoms:

- Easy to caus
- (Too) r me crashes
- Hard t derstand
- Difficult to refactor

My experience with Rust:

My experience with Rust:

- <compiling>

My experience with Rust:

- <compiling>
- <compiling>

My experience with Rust:

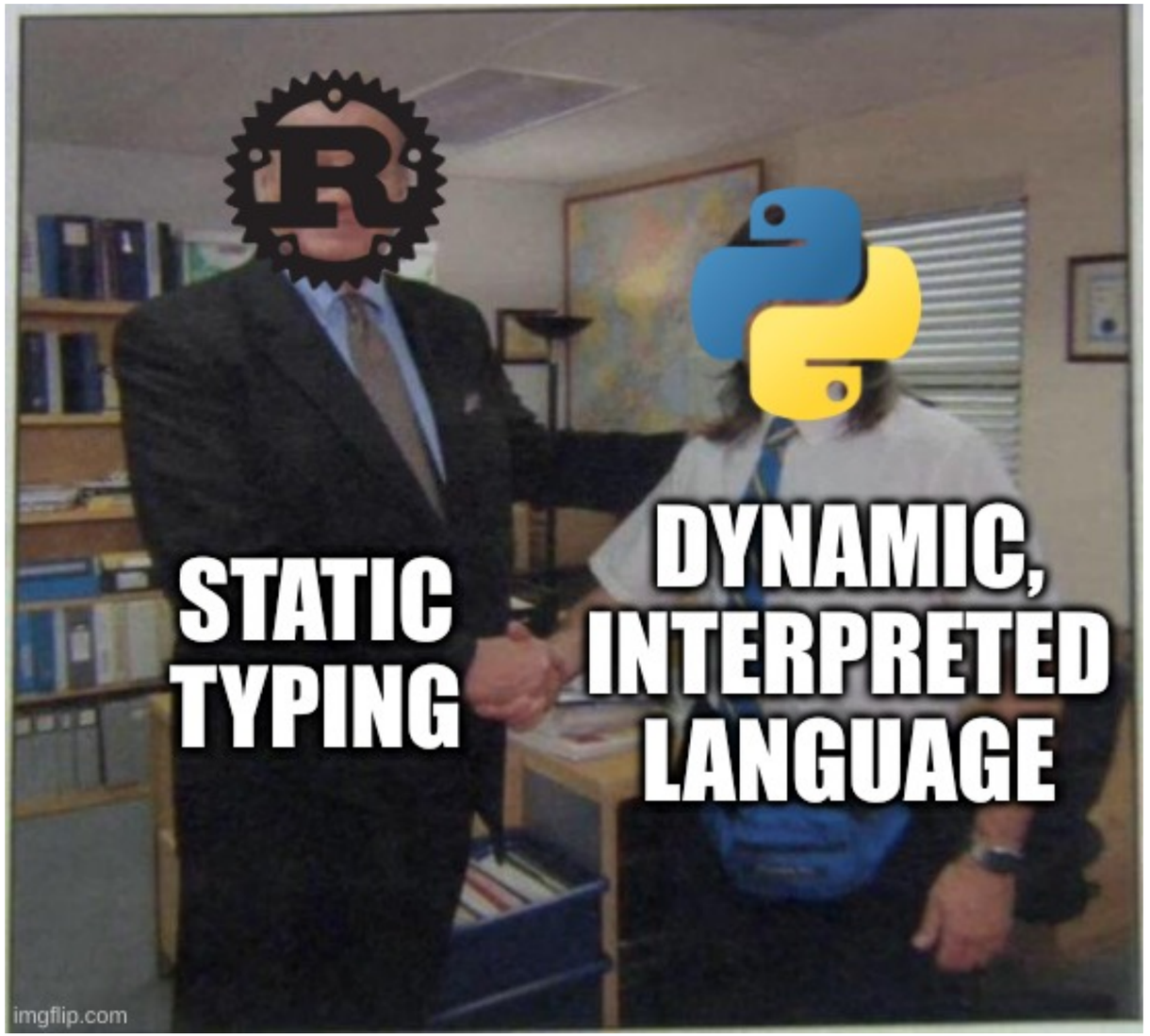
- <compiling>
- <compiling>
- <fighting the compiler>

My experience with Rust:

- <compiling>
- <compiling>
- <fighting the compiler>
- **It just works!**

Why write Python like Rust (and how?)

Why write Python like Rust (and how?)



(disclaimer)

These are just my opinions :)

Step 1

Type hints, type hints, type hints

```
def visit_europython(visitor: Visitor):
    ...
```

PEP 484 – Type Hints

Author:	Guido van Rossum <guido at python.org>, Jukka Lehtosalo <jukka.lehtosalo at iki.fi>, Łukasz Langa <lukasz at python.org>
BDFL-Delegate:	Mark Shannon
Discussions-To:	Python-Dev list
Status:	Final
Type:	Standards Track
Topic:	Typing

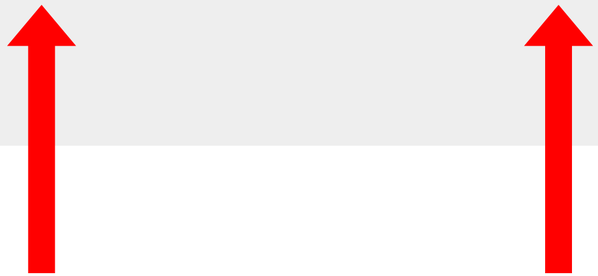
Where to use type hints?

Interface boundaries

```
def add(x: int, b: int) -> int:
    ...
```

Interface boundaries

```
def add(x: int, b: int) -> int:
    ...
```



Interface boundaries

```
def add(x: int, b: int) -> int:
    ...
```

(Data)class fields

```
@dataclass
class Person:
    name: str
    age: int
```

(Rarely) variables

```
result: MyClass = foo().get("bar")[0]
```

(Rarely) variables

```
result: MyClass = foo().get("bar")[0]
x: int = 0
```

```
x: int
```

```
x: int
x: Person
```

```
x: int
x: Person
x: List[int]
```

```
x: int
x: Person
x: List[int]
x: Dict[str, int]
```

```
x: int
x: Person
x: List[int]
x: Dict[str, int]
x: int | str
```

```
x: int
x: Person
x: List[int]
x: Dict[str, int]
x: int  | str
x: bool | None
```

```python
x: int
x: Person
x: List[int]
x: Dict[str, int]
x: int  | str
x: bool | None
x: Literal["GET", "POST"]
```

```python
x: int
x: Person
x: List[int]
x: Dict[str, int]
x: int | str
x: bool | None
x: Literal["GET", "POST"]
from typing import ...
```

Why type hints?

Types help me understand...

```
def find_item(items, check):
```

```
def find_item(
```

```
def find_item(
    items: Iterable[Item],
```

```
def find_item(
    items: Iterable[Item],
    check: Callable[[Item], bool]
```

```
def find_item(
    items: Iterable[Item],
    check: Callable[[Item], bool]
) -> Item | None:
```

```
def find_item(
    items: Iterable[Item],
    check: Callable[[Item], bool]
) -> Item | None:
```

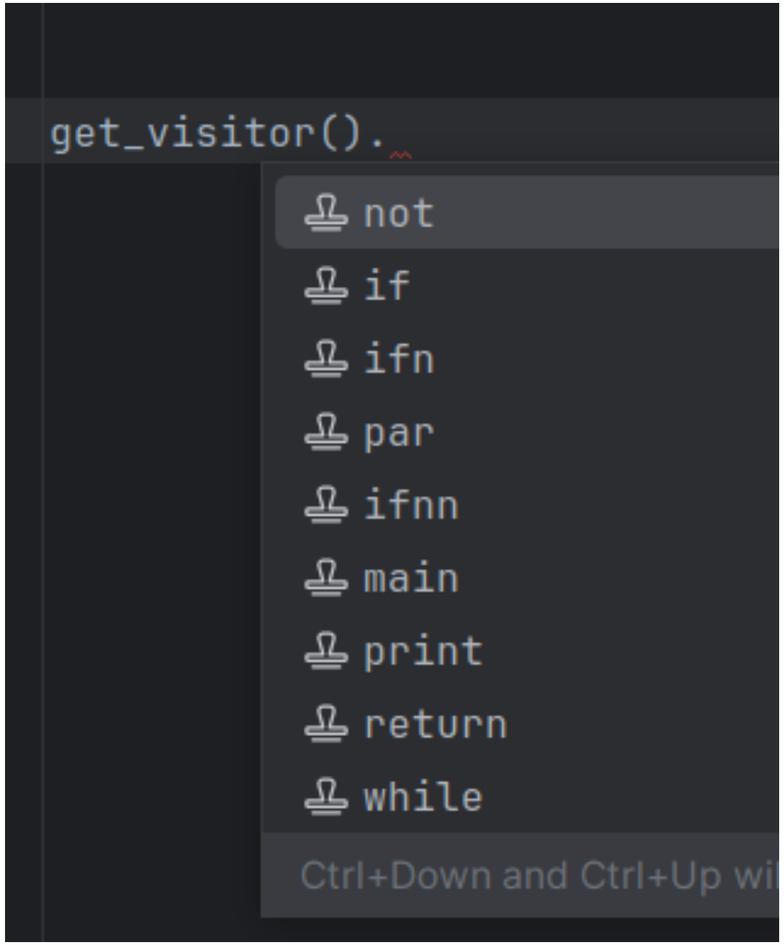
Documentation

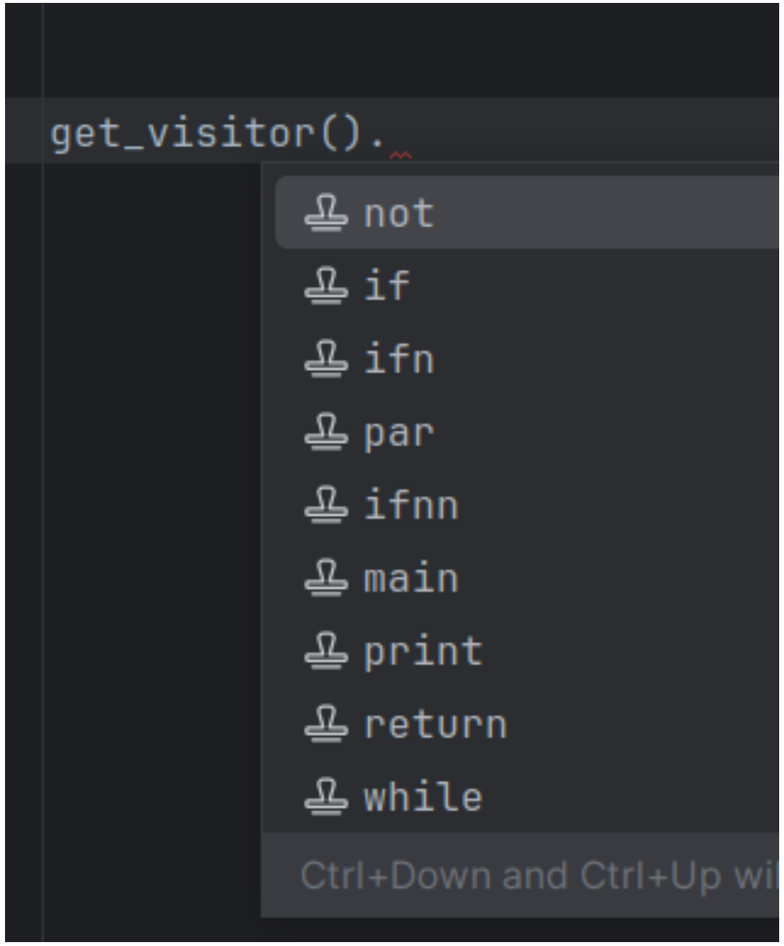
...and remember

```
def __init__(
    self,
    address=None,
    name=None,
    value=None,
    type=None,
    path=None
):
```

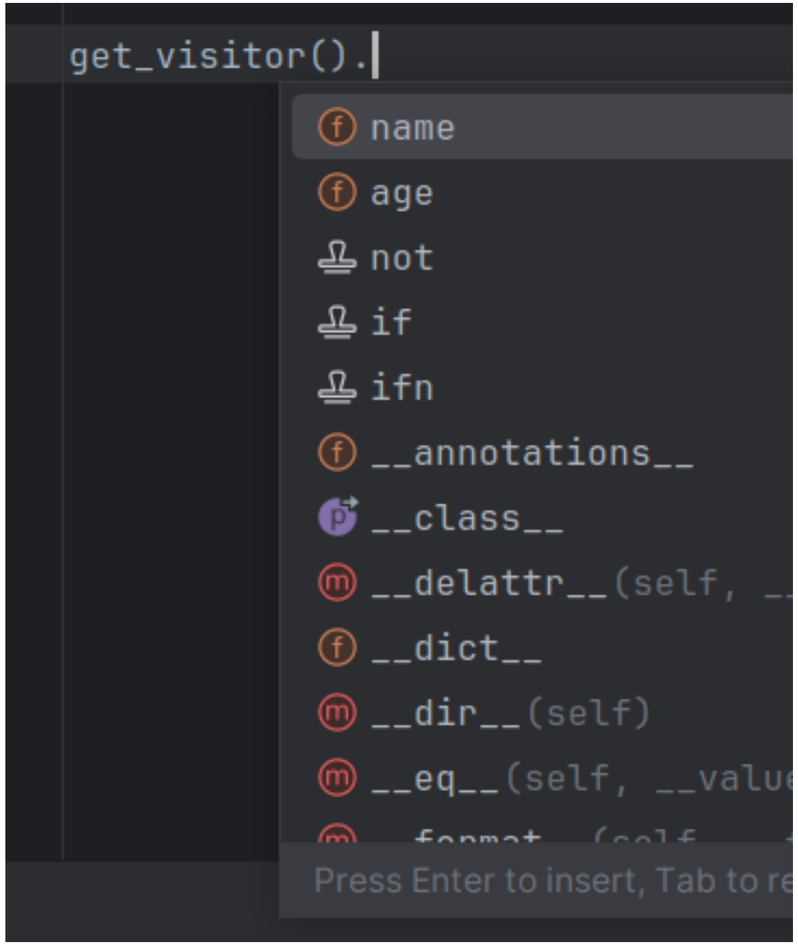

I HAVE NO
MEMORY OF THIS CODE
imgflip.com

Types help me write code faster


get_visitor().
not
if
ifn
par
ifnn
main
print
return
while
Ctrl+Down and Ctrl+Up wi


get_visitor().
not
if
ifn
par
ifnn
main
print
return
while
Ctrl+Down and Ctrl+Up wi

VS


get_visitor().
name
age
not
if
ifn
__annotations__
__class__
__delattr__(self,
__dict__
__dir__(self)
__eq__(self, __value
Press Enter to insert, Tab to re


@dataclasses.dataclass
class Visitor
def store_visitor(visitor: Visitor):


@dataclasses.dataclass
class Visitor
def store_visitor(visitor: Visitor):
@dataclasses.dataclass
class Visitor:
name: str
age: int

Types help me detect complex code

```
def store_visited_talks(
  visitors: Visitor | List[Visitor] | Dict[str, Visitor],
  talks: List[str | Dict[str, Dict[str, int]]]
) -> List[Tuple[str | Dict[str, str], int]] | int
```

```python
def store_visited_talks(
  visitors: Visitor | List[Visitor] | Dict[str, Visitor],
  talks: List[str | Dict[str, Dict[str, int]]]
) -> List[Tuple[str | Dict[str, str], int]] | int
```

Hard to type => hard to understand?

Type hints are still optional!

Type hints are still optional!

```
def log_data(data: Any):
    ...
```

Type hints are still optional!

```
def log_data(data: Any):
    ...
```

This is fine!

Types are introspectable

```python
app = FastAPI()


@app.get("/items/{item_id}")
def get_item(item_id: int):
    ...
```

```python
app = FastAPI()


@app.get("/items/{item_id}")
def get_item(item_id: int):
    ...
```

Parsed as a number

Type checking

Type checking

`pyright`, `mypy`, …

```
> mypy ide.py
ide.py:14: error: Argument 1 to "get_visitor" has incompatible ty
pe "int"; expected "str"  [arg-type]
Found 1 error in 1 file (checked 1 source file)
```

```
> mypy ide.py
ide.py:14: error: Argument 1 to "get_visitor" has incompatible ty
pe "int"; expected "str"  [arg-type]
Found 1 error in 1 file (checked 1 source file)
```

Configure type-checking in CI!

Each type becomes a mini-test

Each type becomes a mini-test

- Does not need maintenance/refactoring

Each type becomes a mini-test

- Does not need maintenance/refactoring
- Low-latency feedback

Making a code change

Making a code change

Without type checking

Making a code change

Without type checking

Making a code change

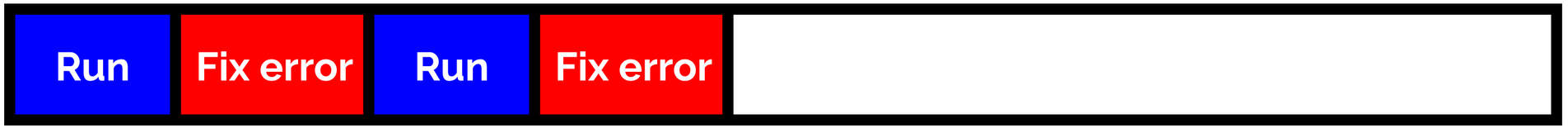
Without type checking

Making a code change

Without type checking

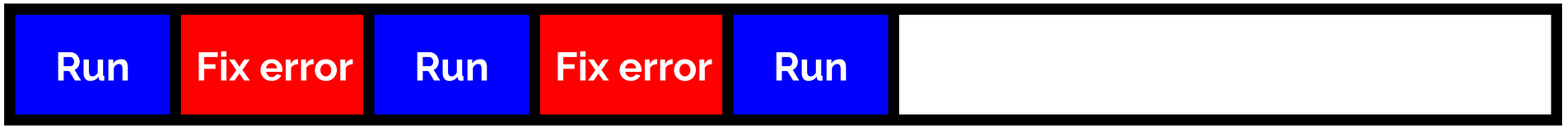
Making a code change

Without type checking



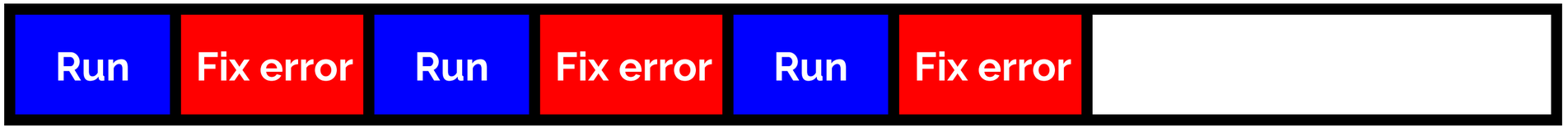
Making a code change

Without type checking



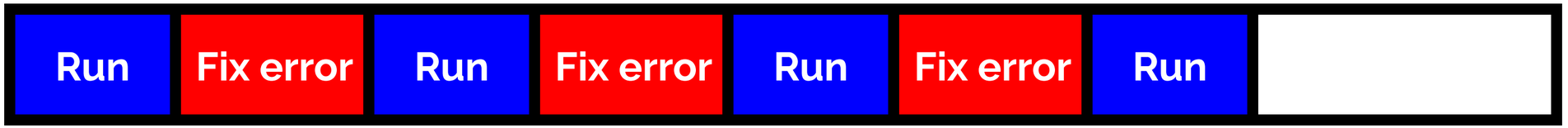
Making a code change

Without type checking



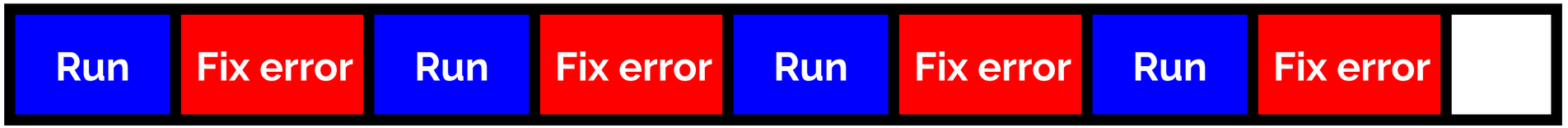
Making a code change

Without type checking



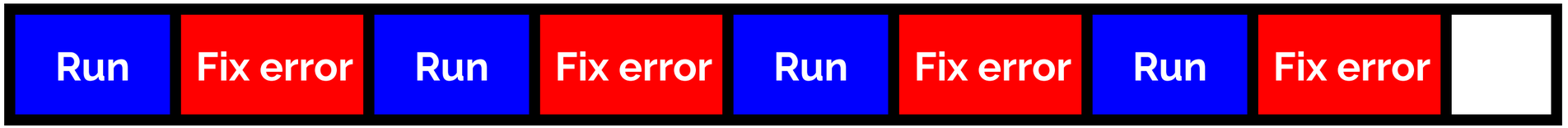
Making a code change

Without type checking



Making a code change

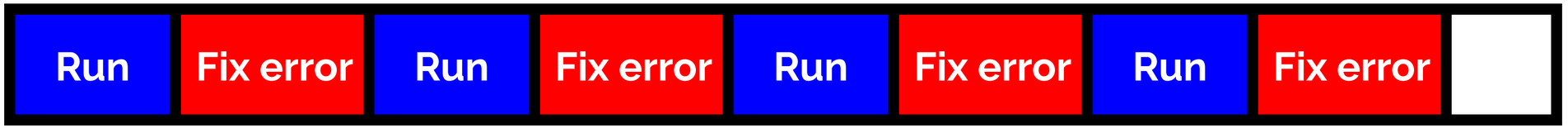
Without type checking



With type checking

Making a code change

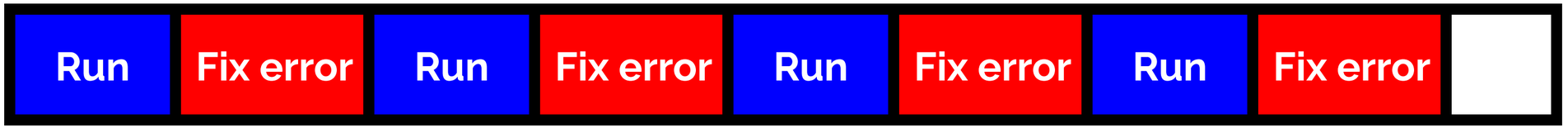
Without type checking



With type checking

Making a code change

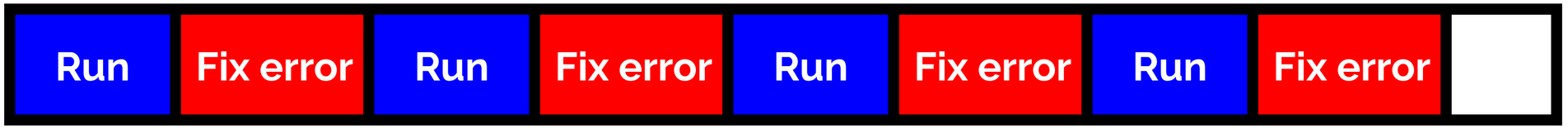
Without type checking



With type checking

Making a code change

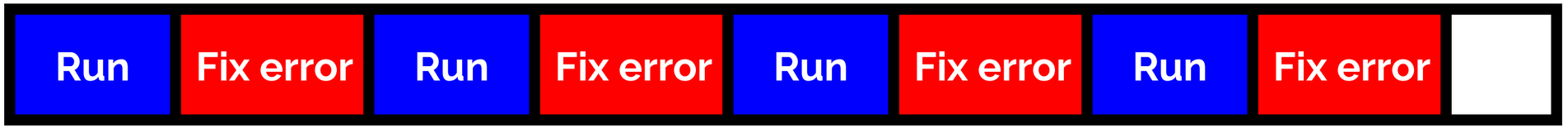
Without type checking



With type checking

Making a code change

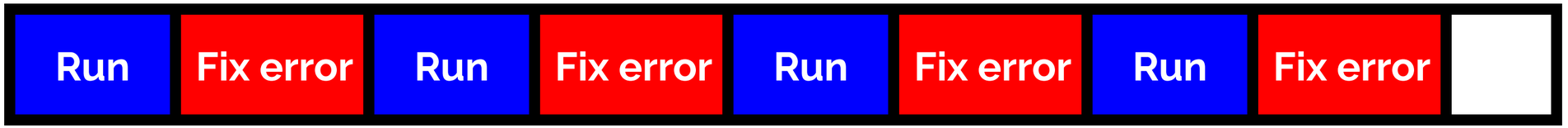
Without type checking



With type checking

Making a code change

Without type checking



With type checking

You can combine both!

FastAPI "benchmark"

FastAPI "benchmark"

- pytest: ~30s

FastAPI "benchmark"

- pytest: ~30s
- mypy: ~1s

Runtime type checks

Runtime type checks

```
@beartype
def get_visitor(name: str):
    ...

get_visitor(1)
```

Runtime type checks

```python
@beartype
def get_visitor(name: str):
    ...

get_visitor(1)
```

```
beartype.roar.BeartypeCallHintParamViolation: Function __main__.get_visitor() parameter name=1 violates type hint <class 'str'>, as int 1 not instance of str.
```

Improves **confidence** in code

Improves **confidence** in code

- Easier code review

Improves **confidence** in code

- Easier code review
- "Fearless" refactoring

Disadvantages?

> More characters to type

> More characters to type

> “
> Useless for throwaway code
> ”

Me: <gets an idea>

Me: <gets an idea>

Let's write a prototype to evaluate it!

Me: <gets an idea>

Let's write a prototype to evaluate it!

Should I use type hints? 🤔

Me: <gets an idea>

Let's write a prototype to evaluate it!

Should I use type hints? 🤔

Nah, this code will be gone by next week

Me: <gets an idea>

Let's write a prototype to evaluate it!

Should I use type hints? 🤔

Nah, this code will be gone by next week

(three years later)

Me: <gets an idea>

Let's write a prototype to evaluate it!

Should I use type hints? 🤔

Nah, this code will be gone by next week

(three years later)

Sh*t! It crashed in production again...

Me: <gets an idea>

Let's write a prototype to evaluate it!

Should I use type hints? 🤔

Nah, this code will be gone by next week

(three years later)

Sh*t! It crashed in production again...
Wait, how did this ever work?!

Me: <gets an idea>

Let's write a prototype to evaluate it!

Should I use type hints? 🤔

Nah, this code will be gone by next week

(three years later)

Sh*t! It crashed in production again…
Wait, how did this ever work?!

Sounds familiar? :)

> “
> False sense of security
> ”

> “
> False sense of security
> ”

Remember: type hints are not perfect!

> “It feels like lipstick on a pig.”
>
> HackerNews user

Step 2

Use *dataclasses*

```python
def get_person() -> Tuple[str, int]:
    ...
```

```python
def get_person() -> Tuple[str, int]:
    ...
```

```python
def get_person() -> Dict[str, Any]:
    ...
```

```python
def get_person() -> Tuple[str, int]:
  ...
```

```python
def get_person() -> Dict[str, Any]:
  ...
```

```python
def get_person() -> Person:
  ...
```

```python
def get_person() -> Tuple[str, int]:
  ...
```

```python
def get_person() -> Dict[str, Any]:
  ...
```

```python
def get_person() -> Person:
  ...
```

```python
@dataclass
class Person:
  name: str
  age: int
```

Step 3

Embrace *soundness*

Sound code = impossible* to misuse

Sound code = impossible* to **misuse**

Sound code = impossible* to **misuse**

- Break invariants

Sound code = impossible* to **misuse**

- Break invariants
- Unintended usage

Sound code = impossible* to **misuse**

- Break invariants
- Unintended usage
- Runtime failures, bugs

Sound code = **impossible*** to misuse

Sound code = **impossible*** to misuse

- Rust: compile-error

Sound code = **impossible*** to misuse

- Rust: compile-error
- Python: type-check error

```python
def get_car_id(brand: str) -> int:
```

```python
def get_car_id(brand: str) -> int:
def get_driver_id(name: str) -> int:
```

```python
def get_car_id(brand: str) -> int:
def get_driver_id(name: str) -> int:
def get_race(car: int, driver: int) -> Race:
```

```
def get_car_id(brand: str) -> int:
def get_driver_id(name: str) -> int:
def get_race(car: int, driver: int) -> Race:
```

```
car_id    = get_car_id("Mazda")
```

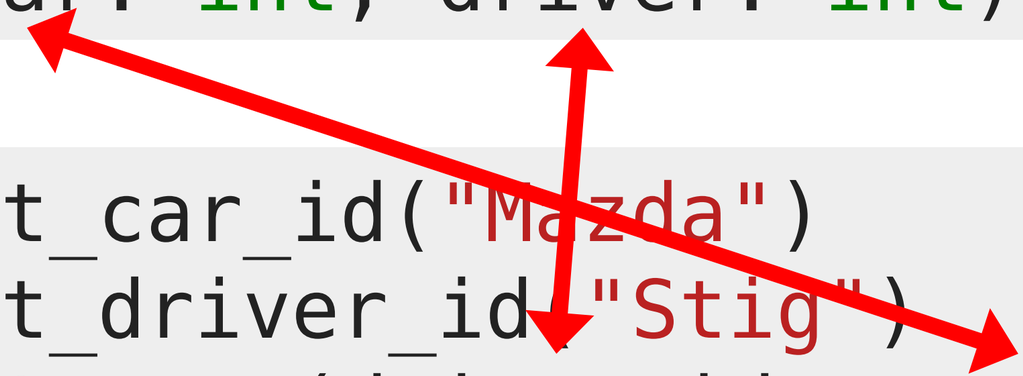
```
def get_car_id(brand: str) -> int:
def get_driver_id(name: str) -> int:
def get_race(car: int, driver: int) -> Race:
```

```
car_id    = get_car_id("Mazda")
driver_id = get_driver_id("Stig")
```

```
def get_car_id(brand: str) -> int:
def get_driver_id(name: str) -> int:
def get_race(car: int, driver: int) -> Race:
```

```
car_id    = get_car_id("Mazda")
driver_id = get_driver_id("Stig")
race      = get_race(driver_id, car_id)
```

```python
def get_car_id(brand: str) -> int:
def get_driver_id(name: str) -> int:
def get_race(car: int, driver: int) -> Race:
```



```python
car_id    = get_car_id("Mazda")
driver_id = get_driver_id("Stig")
race      = get_race(driver_id, car_id)
```

Newtype pattern

```
# New type "CarId", internally an int
CarId    =  typing.NewType("CarId",    int)
DriverId =  typing.NewType("DriverId", int)
```

Newtype pattern

```
# New type "CarId", internally an int
CarId    =  typing.NewType("CarId",    int)
DriverId =  typing.NewType("DriverId", int)
```

```
def get_car_id(brand: str) -> CarId:
def race(car: CarId, driver: DriverId) -> Race:
```

Newtype pattern

```python
# New type "CarId", internally an int
CarId    =  typing.NewType("CarId",    int)
DriverId =  typing.NewType("DriverId", int)
```

```python
def get_car_id(brand: str) -> CarId:
def race(car: CarId, driver: DriverId) -> Race:
```

```
get_race(driver_id, car_id)
```

Newtype pattern

```
# New type "CarId", internally an int
CarId    =  typing.NewType("CarId",    int)
DriverId =  typing.NewType("DriverId", int)
```

```
def get_car_id(brand: str) -> CarId:
def race(car: CarId, driver: DriverId) -> Race:
```

```
get_race(driver_id, car_id)
```

```
error: Argument 1 to "race" has incompatible type "DriverId"; expected "CarId"

error: Argument 2 to "race" has incompatible type "CarId"; expected "DriverId"
```

```
class Client:
```

```
class Client:
  def connect(self):
```

```
class Client:
  def connect(self):
  def send(self):
```

```
class Client:
  def connect(self):
  def send(self):
  def close(self):
```

```
c.send()
c.connect()
```

```
class Client:
  def connect(self):
  def send(self):
  def close(self):
```

```
class Client:
  def connect(self):
  def send(self):
  def close(self):
```

```
c.send()
c.connect()
```

```
c.connect()
c.connect()
```

```python
class Client:
  def connect(self):
  def send(self):
  def close(self):
```

```
c.send()
c.connect()
```

```
c.connect()
c.connect()
```

```
c.close()
c.send()
```

```python
class Client:
  def connect(self):
  def send(self):
  def close(self):
```

```python
c.send()
c.connect()
```

```python
c.connect()
c.connect()
```

```python
c.close()
c.send()
```

```python
c.connect()
c.send()
# forgot to close
```

```
class ConnectedClient:
  def send(self):
```

```
class ConnectedClient:
  def send(self):
```

```
@contextmanager
def connect(address: str):
```

```
class ConnectedClient:
  def send(self):
```

```
@contextmanager
def connect(address: str):
  client = create_client(address)
```

```
class ConnectedClient:
  def send(self):
```

```
@contextmanager
def connect(address: str):
  client = create_client(address)
  try:
    yield client
```

```
class ConnectedClient:
  def send(self):
```

```
@contextmanager
def connect(address: str):
  client = create_client(address)
  try:
    yield client
  except:
    close_client(client)
```

```
class ConnectedClient:
  def send(self):
```

```
@contextmanager
def connect(address: str):
  client = create_client(address)
  try:
    yield client
  except:
    close_client(client)
```

```
with connect("localhost:80") as client:
  client.send()
```

Make illegal states unrepresentable

```
class RequestBuilder:
```

```python
class RequestBuilder:
  def api_token(self, token: str):
```

```python
class RequestBuilder:
  def api_token(self, token: str):
  def password(self,  username: str, password: str):
```

```python
class RequestBuilder:
  def api_token(self, token: str):
  def password(self,  username: str, password: str):
  def build(self) -> Request:
```

```python
class RequestBuilder:
  def api_token(self, token: str):
  def password(self,  username: str, password: str):
  def build(self) -> Request:
```

```python
class RequestWithToken:
  def build(self) -> Request:
```

```python
class RequestBuilder:
  def api_token(self, token: str):
  def password(self,  username: str, password: str):
  def build(self) -> Request:
```

```python
class RequestWithToken:
  def build(self) -> Request:
```

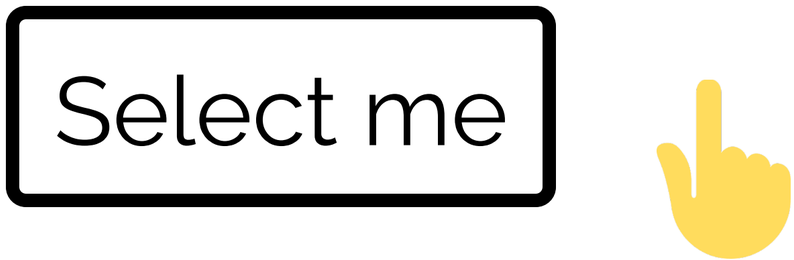
```python
class RequestWithPassword:
  def build(self) -> Request:
```

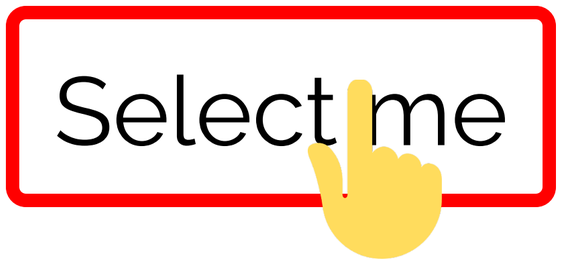
```python
class RequestBuilder:
  def api_token(self, token: str):
  def password(self,  username: str, password: str):
  def build(self) -> Request:
```

```python
class RequestWithToken:
  def build(self) -> Request:
```

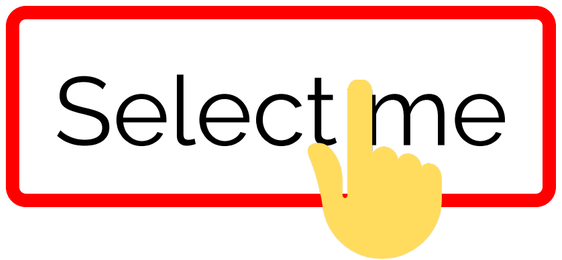
```python
class RequestWithPassword:
  def build(self) -> Request:
```

```python
class RequestBuilder:
  def api_token(...) -> RequestWithToken:
  def password(...) -> RequestWithPassword:
```

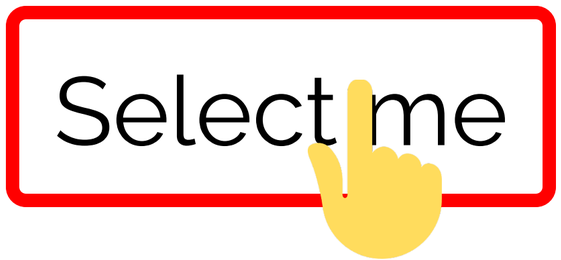

Select me



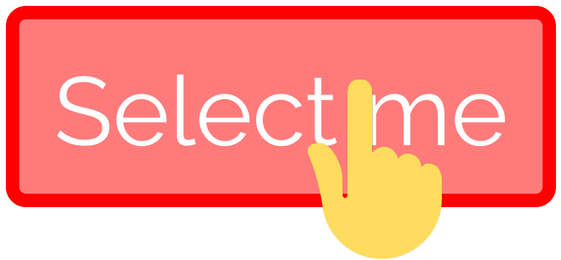
0.0s



0.8s



1.5s



3.0s

```
@dataclass
class ButtonState:
```

```
@dataclass
class ButtonState:
    hover: bool
```

```
@dataclass
class ButtonState:
    hover: bool
    selected: bool
```

```
@dataclass
class ButtonState:
   hover: bool
   selected: bool
   timer: Optional[timedelta]
```

```
@dataclass
class ButtonState:
  hover: bool
  selected: bool
  timer: Optional[timedelta]
```

hover == False + selected == True?

```
@dataclass
class ButtonState:
  hover: bool
  selected: bool
  timer: Optional[timedelta]
```

```
hover == False + selected == True?
  selected == True + timer < 3s?
```

```
def on_hand_leave(state: ButtonState):
    state.hover = False
```

```
def on_hand_leave(state: ButtonState):
    state.hover = False
    # What about selected/timer?
```

State machine + sum types

```
@dataclass
class ButtonInactive:
  pass
```

State machine + sum types

```
@dataclass
class ButtonInactive:
  pass

@dataclass
class ButtonHover:
  timer: timedelta
```

State machine + sum types

```
@dataclass
class ButtonInactive:
  pass

@dataclass
class ButtonHover:
  timer: timedelta

@dataclass
class ButtonSelected:
  pass
```

State machine + sum types

```
@dataclass
class ButtonInactive:
  pass

@dataclass
class ButtonHover:
  timer: timedelta

@dataclass
class ButtonSelected:
  pass

ButtonState = ButtonInactive | ButtonHover | ButtonSelected
```

```
def update_state(
```

```
def update_state(
  state: ButtonState,
```

```
def update_state(
  state: ButtonState,
  hover: bool,
```

```
def update_state(
  state: ButtonState,
  hover: bool,
  delta_time: timedelta
```

```
def update_state(
  state: ButtonState,
  hover: bool,
  delta_time: timedelta
) -> ButtonState:
```

```
def update_state(
  state: ButtonState,
  hover: bool,
  delta_time: timedelta
) -> ButtonState:
  match state:
```

```
def update_state(
  state: ButtonState,
  hover: bool,
  delta_time: timedelta
) -> ButtonState:
  match state:
    case ButtonInactive():
```

```
def update_state(
  state: ButtonState,
  hover: bool,
  delta_time: timedelta
) -> ButtonState:
  match state:
    case ButtonInactive():
      if hover:
        return ButtonHover(timer=timedelta())
```

```
def update_state(
  state: ButtonState,
  hover: bool,
  delta_time: timedelta
) -> ButtonState:
  match state:
    case ButtonInactive():
      if hover:
        return ButtonHover(timer=timedelta())
      return state
    ...
```

```
...
match state:
```

```
...
match state:
  case ButtonHover(timer):
```

```
...
match state:
  case ButtonHover(timer):
    if not hover:
```

```
...
match state:
  case ButtonHover(timer):
    if not hover:
      return ButtonInactive()
```

```
...
match state:
  case ButtonHover(timer):
    if not hover:
      return ButtonInactive()
    if timer + delta_time >= SELECT_TIME:
```

```
...
match state:
  case ButtonHover(timer):
    if not hover:
      return ButtonInactive()
    if timer + delta_time >= SELECT_TIME:
      return ButtonSelected()
```

```
...
match state:
  case ButtonHover(timer):
    if not hover:
      return ButtonInactive()
    if timer + delta_time >= SELECT_TIME:
      return ButtonSelected()
    return ButtonHover(timer=state.timer + delta_time)
```

```
...
match state:
  case ButtonHover(timer):
    if not hover:
      return ButtonInactive()
    if timer + delta_time >= SELECT_TIME:
      return ButtonSelected()
    return ButtonHover(timer=state.timer + delta_time)
  case ButtonSelected():
```

```
...
match state:
  case ButtonHover(timer):
    if not hover:
      return ButtonInactive()
    if timer + delta_time >= SELECT_TIME:
      return ButtonSelected()
    return ButtonHover(timer=state.timer + delta_time)
  case ButtonSelected():
    if not hover:
      return ButtonInactive()
```

```
...
match state:
  case ButtonHover(timer):
    if not hover:
      return ButtonInactive()
    if timer + delta_time >= SELECT_TIME:
      return ButtonSelected()
    return ButtonHover(timer=state.timer + delta_time)
  case ButtonSelected():
    if not hover:
      return ButtonInactive()
    return state
```

```
...
match state:
  case ButtonHover(timer):
    if not hover:
      return ButtonInactive()
    if timer + delta_time >= SELECT_TIME:
      return ButtonSelected()
    return ButtonHover(timer=state.timer + delta_time)
  case ButtonSelected():
    if not hover:
      return ButtonInactive()
    return state
  case _ as unreachable:
    typing.assert_never(unreachable)
```

More examples in my blog post

`kobzol.github.io`

Soundness

Soundness

- Make it hard to make mistakes

Soundness

- Make it hard to make mistakes
- Pit of success

Soundness

- Make it hard to make mistakes
- Pit of success

- Not always possible

Soundness

- Make it hard to make mistakes
- Pit of success

- Not always possible
- Not always worth it!

Summary

Summary

- Step 1: Use type hints

Summary

- Step 1: Use type hints
- Step 2: Use dataclasses

Summary

- Step 1: Use type hints
- Step 2: Use dataclasses
- Step 3: Make code hard to misuse

Summary

- Step 1: Use type hints
- Step 2: Use dataclasses
- Step 3: Make code hard to misuse
- Step 4: ???

Summary

- Step 1: Use type hints
- Step 2: Use dataclasses
- Step 3: Make code hard to misuse
- Step 4: ???
- Step 5: Profit!

Why not just use Rust?

Thank you for your attention!

Slides are available here:

Make slides with Python using `github.com/spirali/nelsie`

Several used icons are from the Twemoji icon pack