РЕФАКТОРИНГ

КРАТКОЕ РУКОВОДСТВО

ПЛАН УРОКА

ТЕОРИЯ

Зачем?

Что?

Как?

ИНСТРУМЕНТЫ

Выявляем проблемы

Чиним

Тестируем

ПРАКТИКА

Пример рефакторинга

PRODUCTION CODE

```
def generate_cartesian_product(generated_config, test_id, params, legit_config_values):
    generated_config[test_id] = []
    if "constraints" in params.keys():
        constrainted_dimensions = set(params["constraints"].keys())
        legit_dimensions = set(params["dimensions"]) - constrainted_dimensions
        if len(legit_dimensions) > 0:
            cartesian_product = list(map(dict, product(*(
                [
                    [(k, v) for v in vv] for k, vv
                    in legit_config_values["dimensions"].items()
                    if k in legit_dimensions
                ] + [
                    [(k, v) for v in vv]
                    for k, vv in params["constraints"].items()
                ] + [
                    [(k, v) for v in vv]
                    for k, vv in {"metric_name": params["metric_name"]}.items()
                ]
            ))))
            return cartesian_product
...
```

REFACTORING

```
from itertools import repeat
from itertools import chain

cat = chain.from_iterable

def project(keys: Sequence, col: dict) -> dict:
    """Возвращает "срез" словаря"""
    target = set(keys)
    return {k: v for k, v in col.items() if k in target}

def unwrap(col: dict) -> list[list]:
    """Создает пары (ключ: значение) для каждого значения"""
    return (zip(repeat(k), v) for k, v in col.items())
```

REFACTORING

```
from itertools import product
from lesson import cat, unwrap, project

def cartesian(*items: dict) -> list[tuple]:
    return product(*cat(map(unwrap, items)))

result = cartesian(
    project(legit_dimensions, legit_config_values['dimensions']),
    params['constraints'],
    project(['metric_name'], params),
)
```

ПЛОХОЙ КОД → КОНФЕТКА

ДО

```
cartesian_product = list(map(dict, product(*(
    [
        [(k, v) for v in vv] for k, vv
        in legit_config_values["dimensions"].items()
        if k in legit_dimensions
     ] + [
         [(k, v) for v in vv]
         for k, vv in params["constraints"].items()
     ] + [
         [(k, v) for v in vv]
         for k, vv in
         {"metric_name": params["metric_name"]}.items()
     ]
```

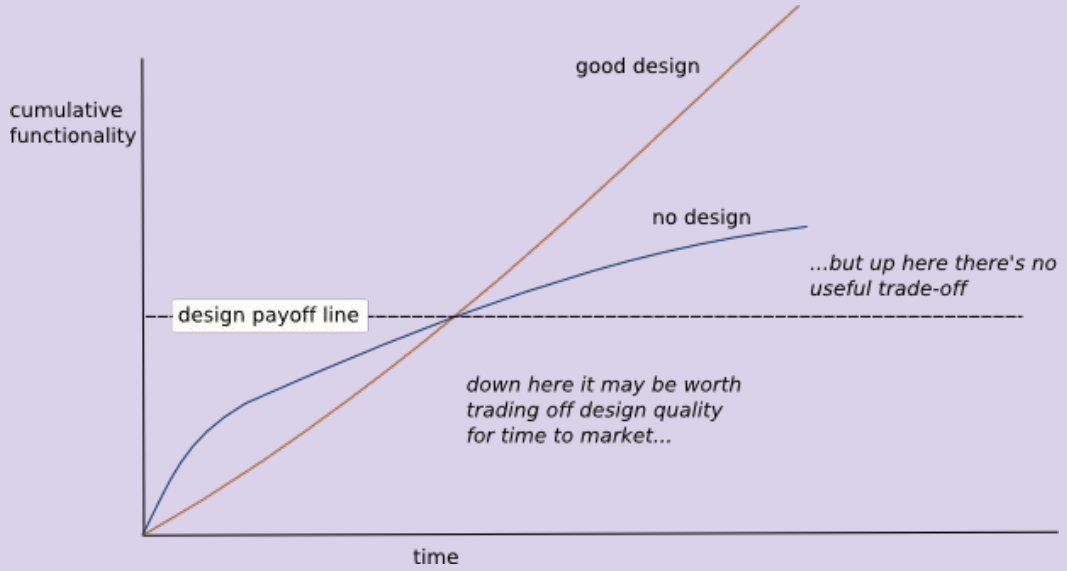
ПОСЛЕ

```
from itertools import product
from lesson import cat, unwrap, project

def cartesian(*items: dict) -> list[tuple]:
    return product(*cat(map(unwrap, items)))

result = cartesian(
    project(legit_dimensions, legit_config_values['dimensions']),
    params['constraints'],
    project(['metric_name'], params),
)
```

ЗАЧЕМ?

~~КАЧЕСТВО КОДА~~
~~ЧИСТЫЙ КОД~~
~~ПРОФЕССИОНАЛИЗМ~~

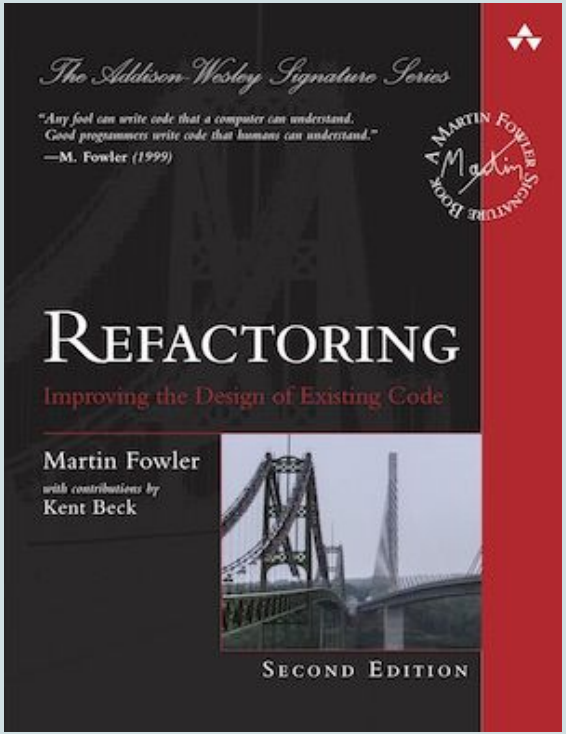
ЭКОНОМИКА



ВЫВОД

УМЕНЬШАЕМ СЛОЖНОСТЬ КОДА,
ЧТОБЫ **БЫСТРЕЕ**
ПРИНОСИТЬ ПОЛЬЗУ
БИЗНЕСУ

REFACTORING



НЕ МЕНЯЕМ ВИДИМОЕ ПОВЕДЕНИЕ

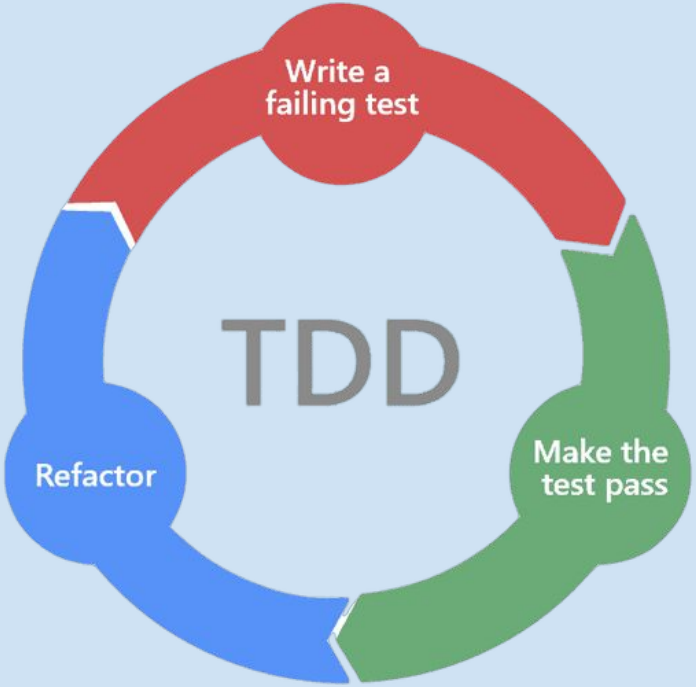
> Refactoring (noun):
> a change made to the
> internal structure
> of software to make it
> easier to understand
> and cheaper to modify
> without changing its
> observable behavior.

Refactoring,
Martin Fowler

РЕФАКТОРИНГ != ОПТИМИЗАЦИЯ ПРОИЗВОДИТЕЛЬНОСТИ

TDD

TEST
DRIVEN
DEVELOPMENT



КОГДА?

ВСЕГДА.

ПО ЧУТЬ-ЧУТЬ

ЦИКЛИЧНО

ПРАВИЛО

ВСЕГДА ОСТАВЛЯЙ ЗА СОБОЙ КОД ЧИЩЕ,
ЧЕМ ОН БЫЛ РАНЬШЕ

КОГДА?

ЕСЛИ НАТКНУЛСЯ НА КОД,
КОТОРЫЙ ТЕБЕ СЛОЖНО ПОНЯТЬ —
ЭТО ХОРОШИЙ МОМЕНТ ДЛЯ РЕФАКТОРИНГА.
ПЕРЕНЕСИ ЭТО “ПОНИМАНИЕ” В КОД

КОГДА?

ЕСЛИ ПОНИМАЕШЬ, ЧТО, ДЛЯ ТОГО ЧТОБЫ
БЫЛО ПРОЩЕ ДОБАВИТЬ НОВЫЙ ФУНКЦИОНАЛ,
ЛУЧШЕ ПОМЕНЯТЬ СТАРЫЙ —
ЭТО ПОВОД РЕФАКТОРИТЬ

ЗАПЛАНИРУЕМ РЕФАКТОРИНГ?

Почти всегда

Если есть, код, который ты никогда не используешь, не будешь использовать, то можно его не трогать

ПОРЯДОК ДЕЙСТВИЙ

УБЕЖДАЕМСЯ В НАЛИЧИИ ТЕСТОВ

ВЫЯВЛЯЕМ ПРОБЛЕМЫ

как понять, что чинить?

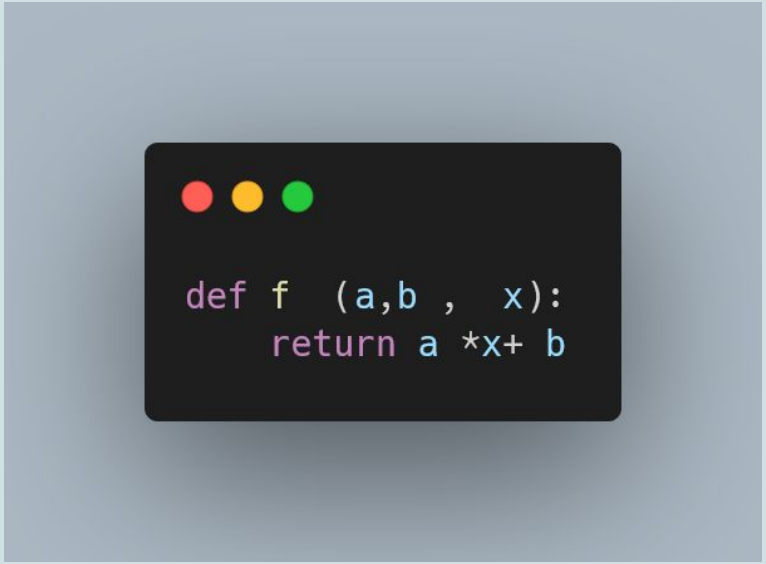
ИСПРАВЛЯЕМ

как чинить?

ПРОВЕРЯЕМ ТЕСТИРОВАНИЕМ

ничего не сломали?

CODE STYLE



ЛИНТЕРЫ

pycodestyle(pep8)

pylint

```
(venv) → lecture_refactoring_v2 git:(master) × pylint after_*
************* Module after_employee
after_employee.py:1:0: C0114: Missing module docstring (missing-module-docstring)
after_employee.py:26:19: W0703: Catching too general exception Exception (broad-except)
************* Module after_main
after_main.py:1:0: C0114: Missing module docstring (missing-module-docstring)

-------------------------------------------------------------------
Your code has been rated at 9.56/10 (previous run: 9.56/10, +0.00)
```

СОРТИРОВКА ИМПОРТОВ

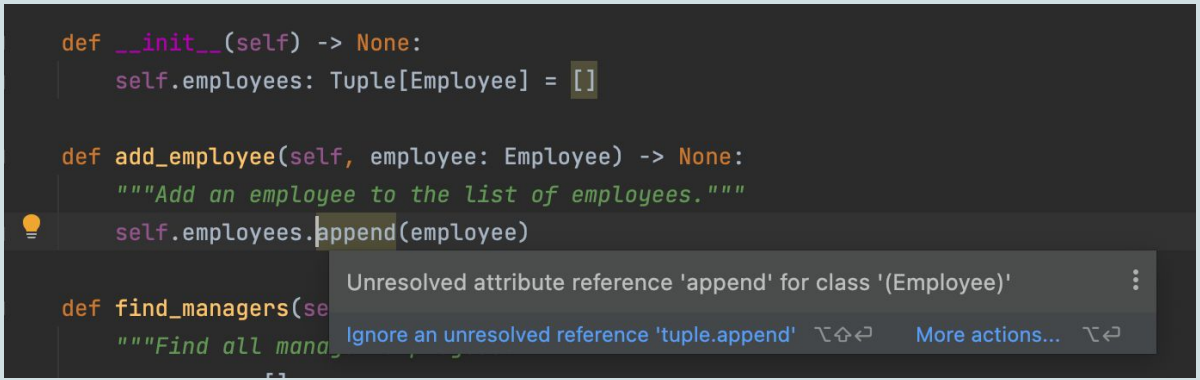
```
from my_lib import Object

import os

from my_lib import Object3

from my_lib import Object2

import sys

from third_party import lib15, lib1, lib2, lib3, lib4, lib5, lib6, lib7, lib8, lib9, lib10, li

import sys

from __future__ import absolute_import

from third_party import lib3

print("Hey")
print("yo")
```

ТИПИЗАЦИЯ

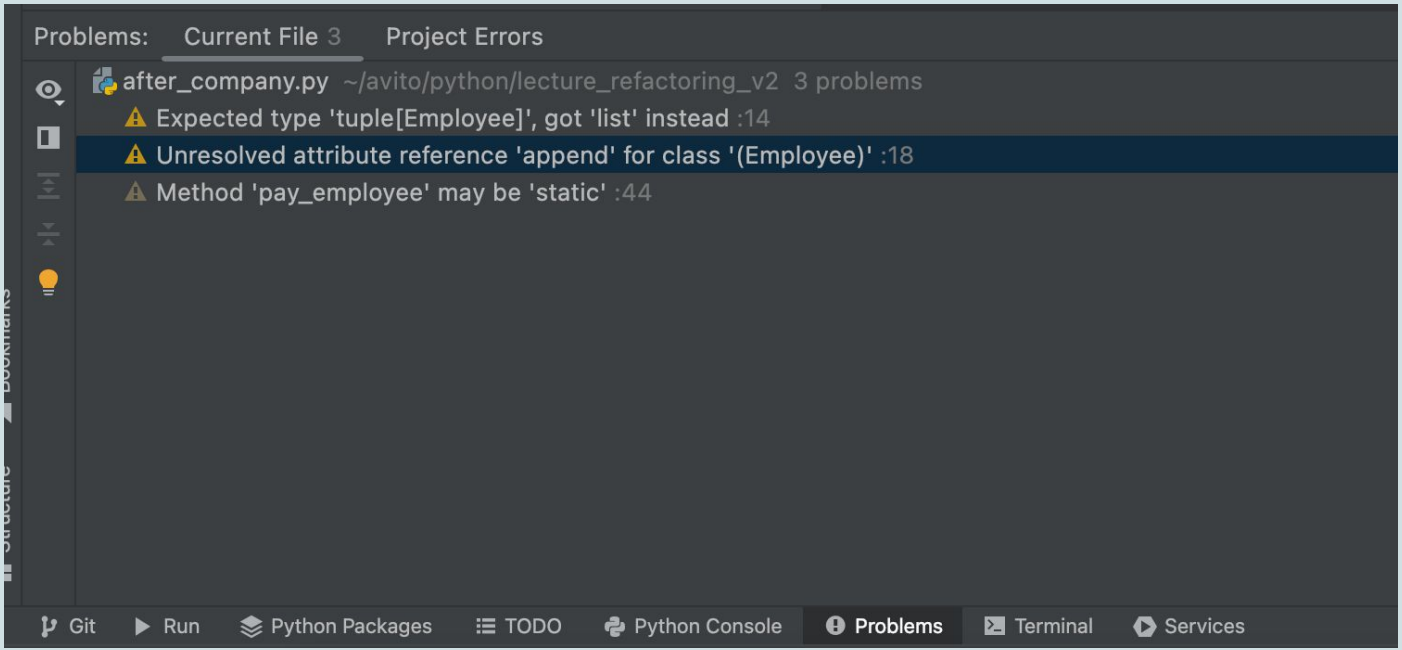
mypy

```
→  lecture_refactoring_v2 git:(master) ✗ mypy after_*
after_company.py:14: error: Incompatible types in assignment (expression has type "List[<nothing>]",
variable has type "Tuple[Employee]")  [assignment]
after_company.py:18: error: "Tuple[Employee]" has no attribute "append"  [attr-defined]
Found 2 errors in 1 file (checked 3 source files)
```

IDE


def __init__(self) -> None:
self.employees: Tuple[Employee] = []
def add_employee(self, employee: Employee) -> None:
"""Add an employee to the list of employees."""
self.employees.append(employee)
Unresolved attribute reference 'append' for class '(Employee)'
Ignore an unresolved reference 'tuple.append'
More actions...
def find_managers(se
"""Find all

IDE


Problems: Current File 3 Project Errors
after_company.py ~/avito/python/lecture_refactoring_v2 3 problems
Expected type 'tuple[Employee]', got 'list' instead :14
Unresolved attribute reference 'append' for class '(Employee)' :18
Method 'pay_employee' may be 'static' :44
Git
Run
Python Packages
TODO
Python Console
Problems
Terminal
Services

CODE SMELLS

РАЗДУВАЛЬЩИКИ

НАРУШИТЕЛИ ОБЪЕКТНОГО ДИЗАЙНА

УТЯЖЕЛИТЕЛИ ИЗМЕНЕНИЙ

ЗАМУСОРИВАТЕЛИ

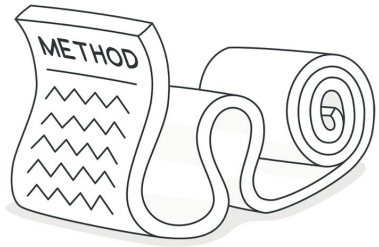
ОПУТЫВАТЕЛИ СВЯЗЯМИ

Длинный метод

Также известен как: Long Method

Симптомы и признаки

Метод содержит слишком большое число строк кода. Длина метода более десяти строк должна начинать вас беспокоить.



Причины появления

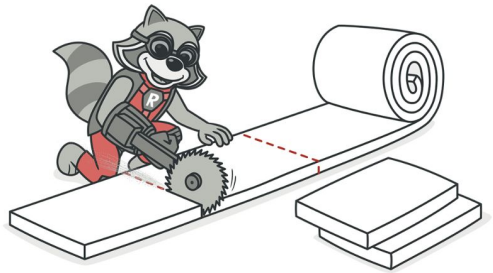
В метод всё время что-то добавляется, но ничего не выносится. Так как писать код намного проще, чем читать, этот запах долго остаётся незамеченным — до тех пор пока метод не превратится в настоящего монстра.

Стоит помнить, что человеку зачастую ментально сложнее создать новый метод, чем дописать что-то в уже существующий: «Как же, мне нужно добавить всего две строки, не буду же я создавать для этого целый метод».

Таким образом, добавляется одна строка за другой, а в результате метод превращается в большую тарелку спагетти.

Лечение

Следует придерживаться такого правила: если ощущается необходимость что-то прокомментировать внутри метода, этот код лучше выделить в новый метод. Даже одну строку имеет смысл выделить в метод, если она нуждается в разъяснениях. К тому же, если у метода хорошее название, то не нужно будет смотреть в его код, чтобы понять, что он делает.



- Для сокращения тела метода достаточно применить **извлечение метода**.
- Если локальные переменные и параметры препятствуют выделению метода, можно применить **замену временной переменной вызовом метода**, **замену параметров объектом** и **передачу всего объекта**.
- Если предыдущие способы не помогли, можно попробовать выделить весь метод в отдельный объект с помощью **замены метода объектом методов**.
- Условные операторы и циклы свидетельствуют о возможности выделения кода в отдельный метод. Для работы с условными выражениями подходит **декомпозиция условных операторов**. Для работы с циклом — **извлечение метода**.

DRY KISS YAGNI

DRY = DON’T REPEAT YOURSELF

KISS = KEEP IT SIMPLE

YAGNI = YOU AREN’T GONNA NEED IT

АВТОФОРМАТТЕРЫ

- black
- yapf

```
(venv) ➜  lecture_refactoring_v2 git:(master) ✗ black after_company.py
reformatted after_company.py

All done! ✨🍰 ✨
1 file reformatted.
(venv) ➜  lecture_refactoring_v2 git:(master) ✗ git diff after_company.py
```

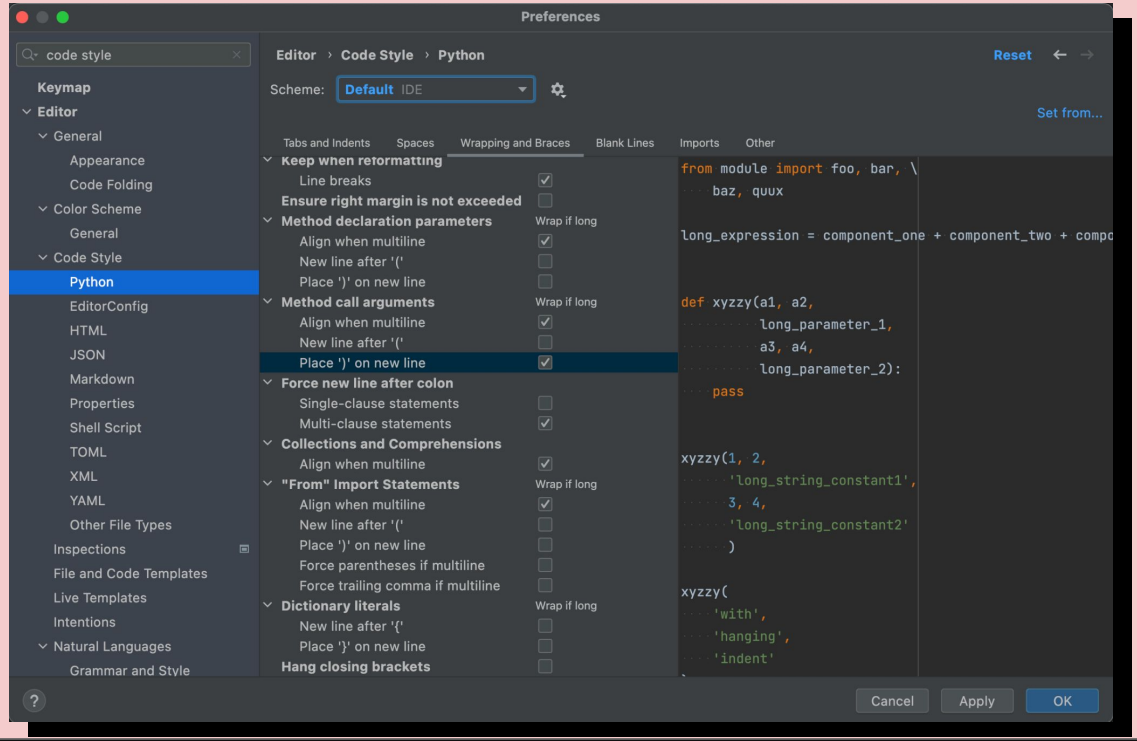
АВТОФОРМАТТЕРЫ

```
            if employee.role == "manager":
-               managers.append(
-                   employee
-               )
+               managers.append(employee)
        return managers

    def find_vice_presidents(self) -> List[Employee]:
@@ -42,24 +38,17 @@ class Company:
        interns = []
        for employee in self.employees:
            if employee.role == "intern":
-               interns.append(
-                   employee
-               )
+               interns.append(employee)
        return interns

    def pay_employee(self, employee: Employee) -> None:
        """Pay an employee."""
-       if isinstance(
-               employee, SalariedEmployee
-       ):
+       if isinstance(employee, SalariedEmployee):
            print(
                f"Paying employee "
                f"{employee.name} a monthly salary of ${employee.monthly_salary}."
            )
-       elif isinstance(
-               employee,
-               HourlyEmployee
-       ):
+       elif isinstance(employee, HourlyEmployee):
            print(
                f"Paying employee {employee.name} a hourly rate of \
```

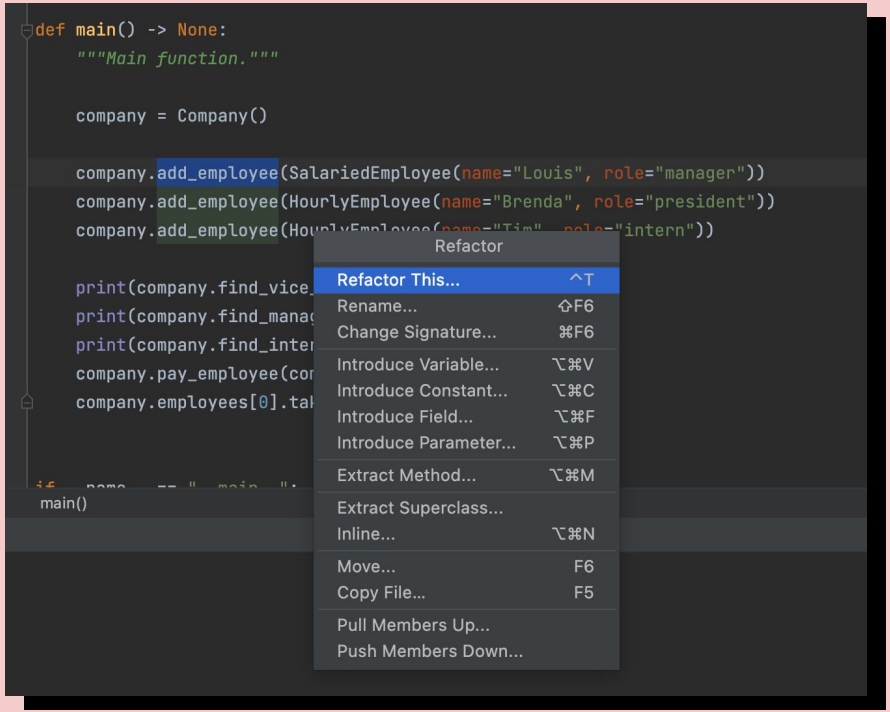
ISORT

```
from __future__ import absolute_import

import os
import sys

from third_party import (lib1, lib2, lib3, lib4, lib5, lib6, lib7, lib8,
                         lib9, lib10, lib11, lib12, lib13, lib14, lib15)

from my_lib import Object, Object2, Object3

print("Hey")
print("yo")
```

IDE CODE STYLE SCHEMA



IDE


def main() -> None:
"""Main function."""
company = Company()
company.add_employee(SalariedEmployee(name="Louis", role="manager"))
company.add_employee(HourlyEmployee(name="Brenda", role="president"))
company.add_employee(HourlyEmployee(name="Tim", role="intern"))
print(company.find_vice_
print(company.find_mana
print(company.find_inter
company.pay_employee(cor
company.employees[0].tal
main()
Refactor
Refactor This... ^T
Rename... ⇧F6
Change Signature... ⌘F6
Introduce Variable... ⌥⌘V
Introduce Constant... ⌥⌘C
Introduce Field... ⌥⌘F
Introduce Parameter... ⌥⌘P
Extract Method... ⌥⌘M
Extract Superclass...
Inline... ⌥⌘N
Move... F6
Copy File... F5
Pull Members Up...
Push Members Down...

ТЕСТЫ

PYTEST

UNITTEST

MOCK

ДО

```
numbers = [-1, -2, -4, 0, 3, -7]
has_positives = False
for n in numbers:
    if n > 0:
        has_positives = True
        break
```

ПОСЛЕ

```
numbers = [-1, -2, -4, 0, 3, -7]
has_positives = any(n > 0 for n in numbers)
```

PRODUCTION CODE

```
def generate_cartesian_product(generated_config, test_id, params, legit_config_values):
    generated_config[test_id] = []
    if "constraints" in params.keys():
        constrainted_dimensions = set(params["constraints"].keys())
        legit_dimensions = set(params["dimensions"]) - constrainted_dimensions
        if len(legit_dimensions) > 0:
            cartesian_product = list(map(dict, product(*(
                [
                    [(k, v) for v in vv] for k, vv
                    in legit_config_values["dimensions"].items()
                    if k in legit_dimensions
                ] + [
                    [(k, v) for v in vv]
                    for k, vv in params["constraints"].items()
                ] + [
                    [(k, v) for v in vv]
                    for k, vv in {"metric_name": params["metric_name"]}.items()
                ]
            ))))
            return cartesian_product
...
```

REFACTORING

```
from itertools import repeat
from itertools import chain

cat = chain.from_iterable

def project(keys: Sequence, col: dict) -> dict:
    """Возвращает "срез" словаря"""
    target = set(keys)
    return {k: v for k, v in col.items() if k in target}

def unwrap(col: dict) -> list[list]:
    """Создает пары (ключ: значение) для каждого значения"""
    return (zip(repeat(k), v) for k, v in col.items())
```

REFACTORING

```
from itertools import product
from lesson import cat, unwrap, project

def cartesian(*items: dict) -> list[tuple]:
    return product(*cat(map(unwrap, items)))

result = cartesian(
    project(legit_dimensions, legit_config_values['dimensions']),
    params['constraints'],
    project(['metric_name'], params),
)
```

ПЛОХОЙ КОД → КОНФЕТКА

ДО

```
cartesian_product = list(map(dict, product(*(
    [
        [(k, v) for v in vv] for k, vv
        in legit_config_values["dimensions"].items()
        if k in legit_dimensions
     ] + [
         [(k, v) for v in vv]
         for k, vv in params["constraints"].items()
     ] + [
         [(k, v) for v in vv]
         for k, vv in
         {"metric_name": params["metric_name"]}.items()
     ]
```

ПОСЛЕ

```
from itertools import product
from lesson import cat, unwrap, project

def cartesian(*items: dict) -> list[tuple]:
    return product(*cat(map(unwrap, items)))

result = cartesian(
    project(legit_dimensions,
legit_config_values['dimensions']),
    params['constraints'],
    project(['metric_name'], params),
)
```

ПРАКТИКА

Пример рефакторинга

Изучили

1. Что такое рефакторинг
2. Когда и как рефакторить
3. Основные инструменты

МАТЕРИАЛЫ

Ошибки новичков в python

Martin Fowler - Refactoring

codesmells/refactorings (rafactoring guru)

ВОПРОСЫ?

ДЗ

Ссылка на дз

СПАСИБО ЗА ВНИМАНИЕ!