Wyższa Szkoła Bankowa we Wrocławiu

O prowadzącym

Kamil Musiał

Tester w Tieto (wcześniej 7 lat tester / integrator / verification specialist w Nokia)

certyfikowany tester ISTQB (full advanced)

od 5 lat wykładowca WSB Wrocław (testowanie, telekomunikacja, sieci, IoT, Python)

doktorant Politechnika Wrocławska

fan morsowania zanim to było modne

kamil.musial@wsb.wroclaw.pl
kamil.musial@chorzow.wsb.pl

Test-Driven Development (4h)

01 Python: funkcje

02 Teoria: część 1.

03 Podstawowe testy w Pytest

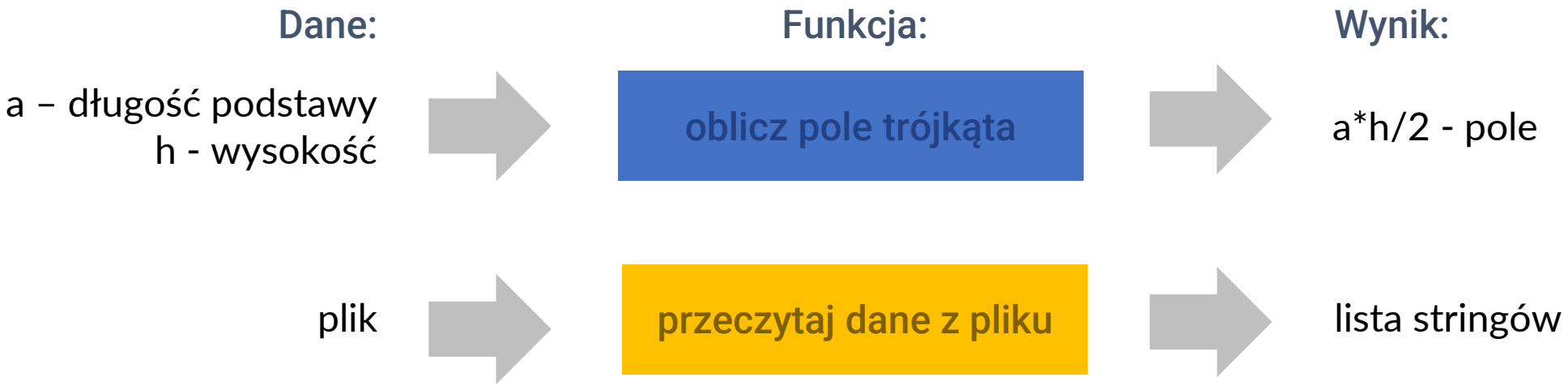
04 Teoria: cześć 2.

05 Przykład FizzBuzz

06 Ćwiczenie

Python: funkcje

Funkcja – część programu, zgrupowany kod programu odpowiadający za pewną funkcjonalność (możliwość wielokrotnego użycia)



Teoria część 1.

Za pomocą testów jednostkowych weryfikujemy funkcjonalność programu na najbardziej podstawowym poziomie

Testujemy każdą jednostkę kodu, (zazwyczaj metodę / funkcję), w izolacji od innych, aby sprawdzić, czy w określonych warunkach reaguje w oczekiwany sposób

Przeniesienie testowania na ten poziom daje wysokie prawdopodobieństwo, że każda część aplikacji będzie zachowywać się zgodnie z oczekiwaniami i umożliwia wykrycie przypadków brzegowych, w których aplikacja może działać w niestandardowy sposób i odpowiednio radzić sobie z nimi

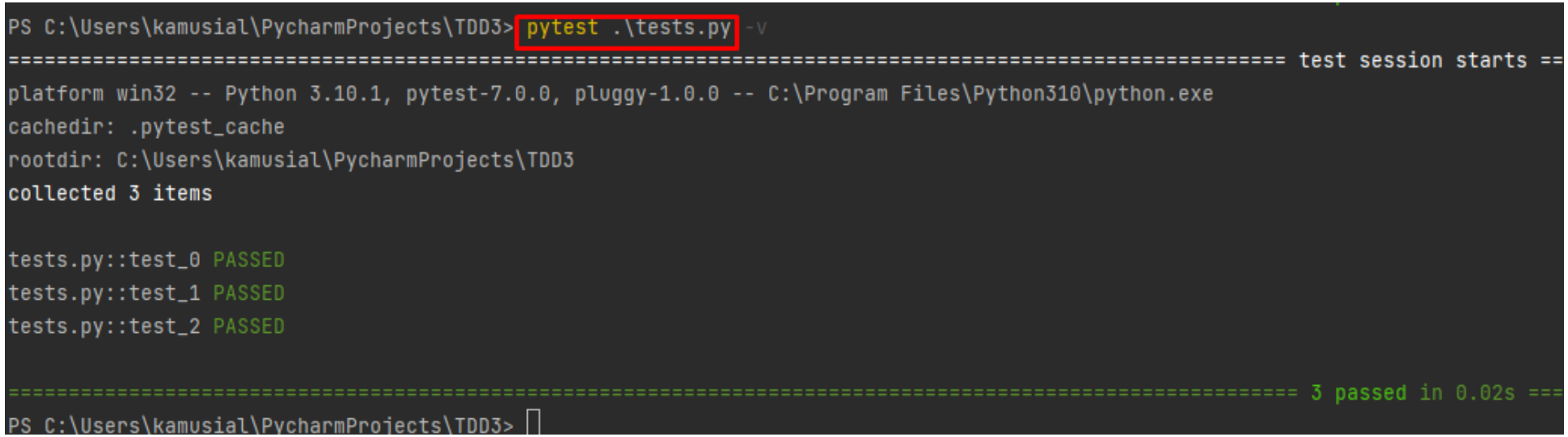
Podstawowe testy w Pytest

W terminalu wpisujemy:

pytest <nazwa_pliku> -v

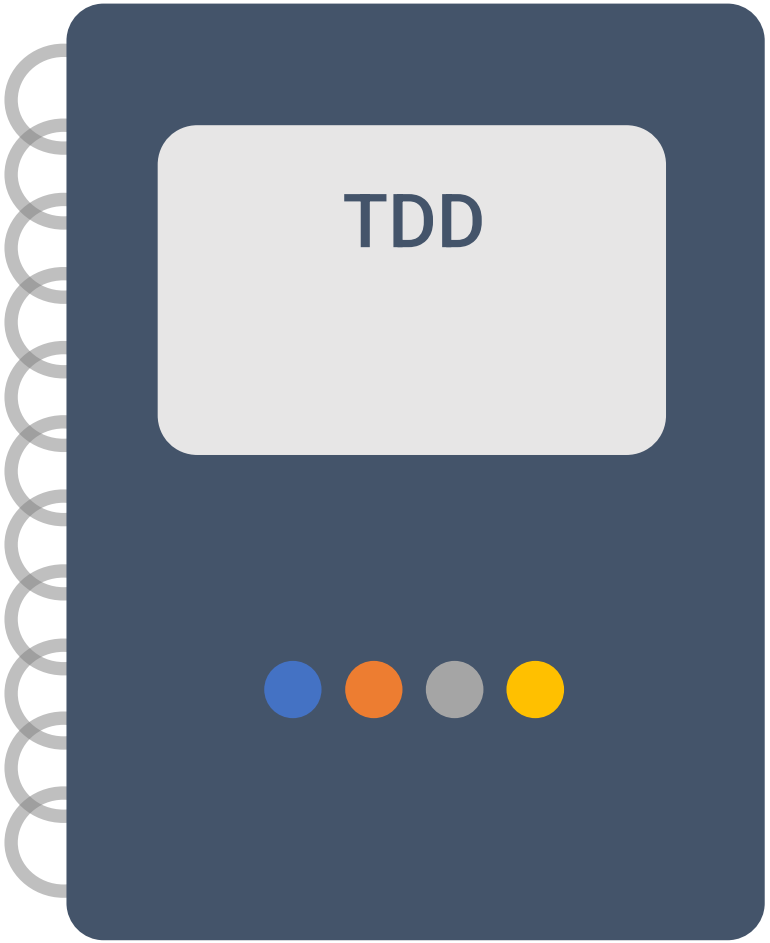
gdzie:

-v oznacza verbose, pokazanie szczegółów



Teoria część 2.

Programowanie sterowane / kierowane / zarządzane testami



„Dopiero pisząc testy zrozumiem, jak mało rozumiem o definicji problemu, który dostałem”

Zanim napiszemy test należy w pełni zrozumieć problem, który chcemy testować

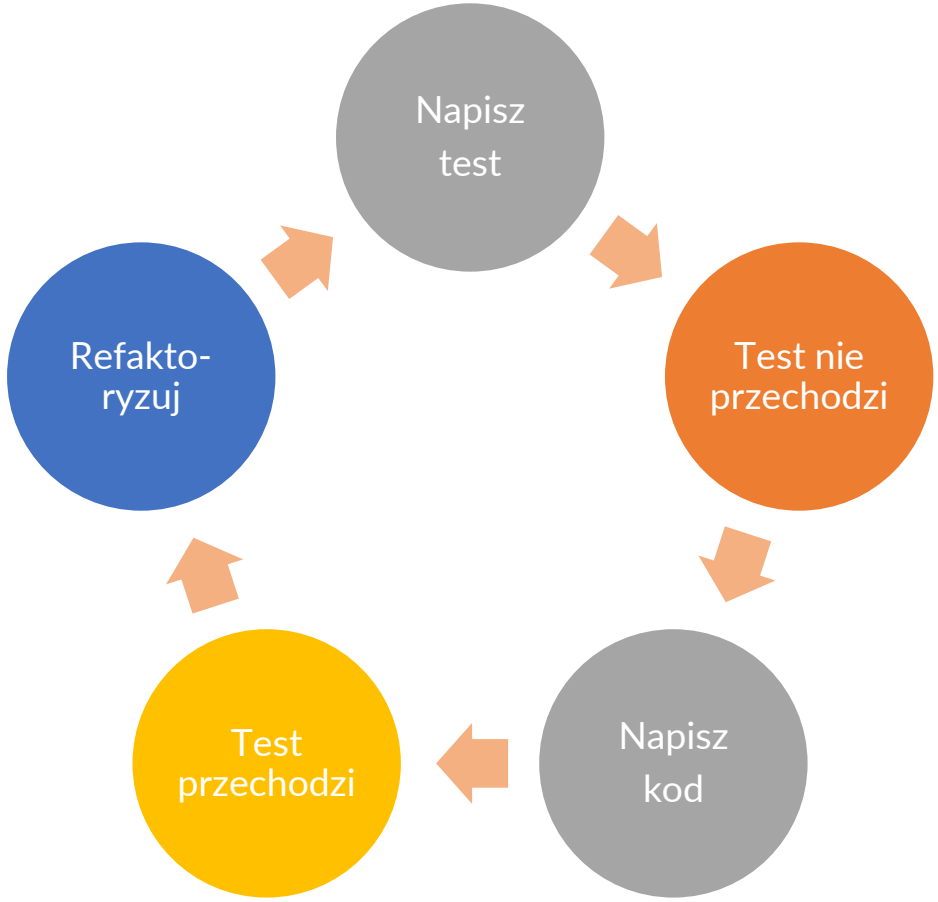
TDD formalizuje wymagania i zmusza do spisania wymagań bardzo szczegółowo (bez wymagań nie da się napisać testów)

TDD wymusza poprawną i modularną organizację kodu

Pisanie dobrych testów wymaga większego doświadczenia niż pisanie dobrego kodu

Teoria część 2.

Programowanie sterowane / kierowane / zarządzane testami

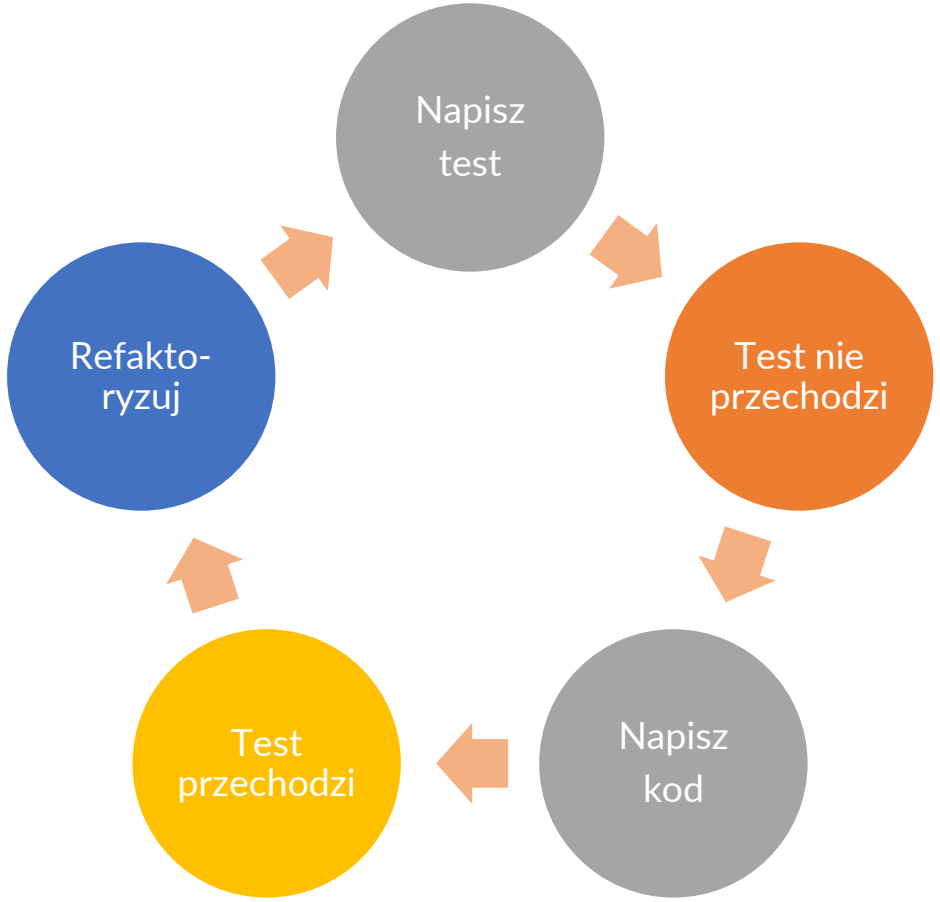


TDD – (Test-Driven Development) to paradygmat,
w ramach którego wdraża się nową funkcję lub wymaganie:

- najpierw pisząc testy
- obserwując, jak kończą się niepowodzeniem
- a następnie pisząc kod, aby testy które zakończyły się niepowodzeniem – teraz przeszły

Teoria część 2.

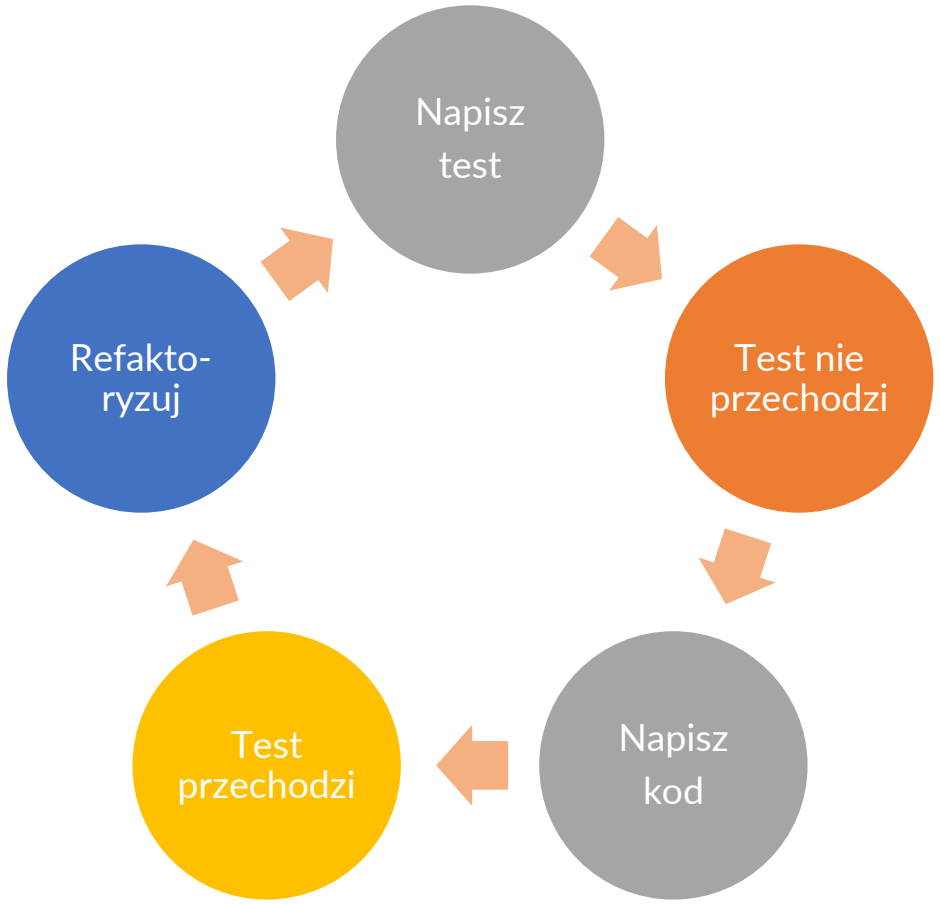
Etap 1: Napisz test



- Dobry test powinien zastąpić dokumentację techniczną – zysk czasowy
- Code review to w 90% oglądanie testów
- Pojedynczy unit test powinien weryfikować elementarną funkcjonalność – zbyt duży test wskazuje na zbyt dużą i skomplikowaną funkcję / metodę
- Nie testujemy implementacji kompilatora, linkera, parsera, czy maszyny wirtualnej

Teoria część 2.

Etap 1: Napisz test



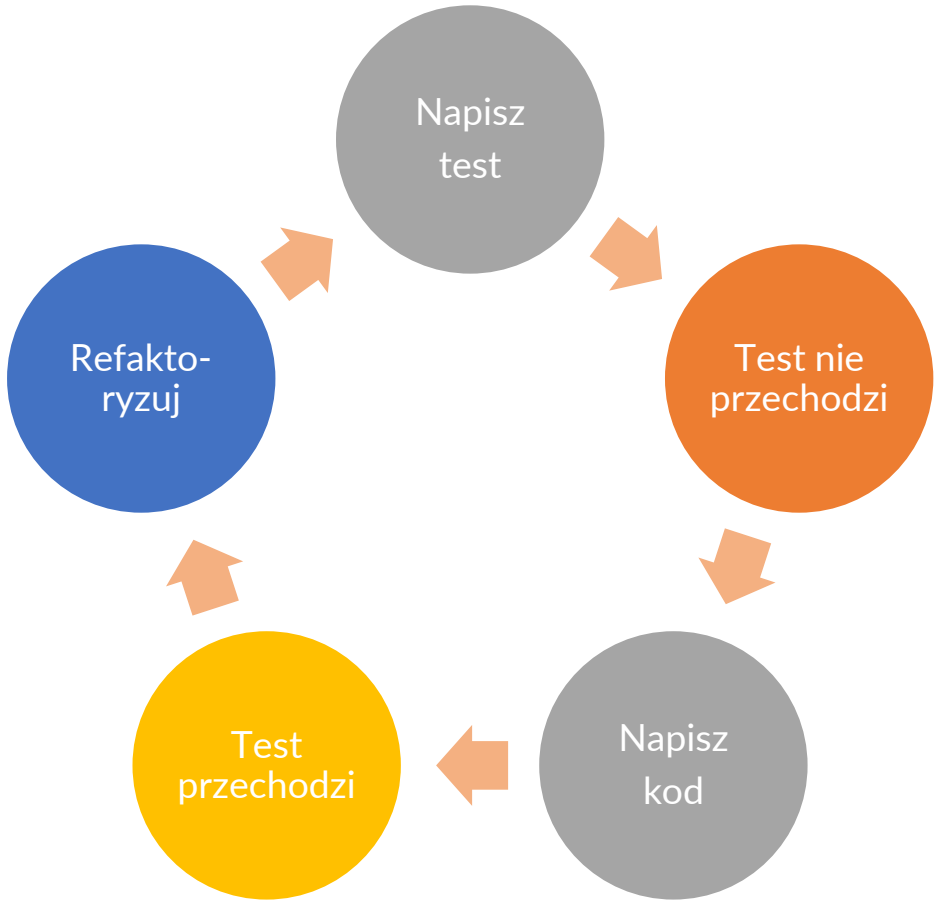
Przykład:

funkcja `sqrt()`:

- Przetestowanie dla wartości argumentu 0, 1, 49, 10
- Czy `sqrt(0.25) = 0.5` ?
- Czy `sqrt(7) * sqrt(7)  = (6.99999; 7.0001)` ?
- Wartości ujemne
- Wartość null
- `Sqrt(100**10)` ?

Teoria część 2.

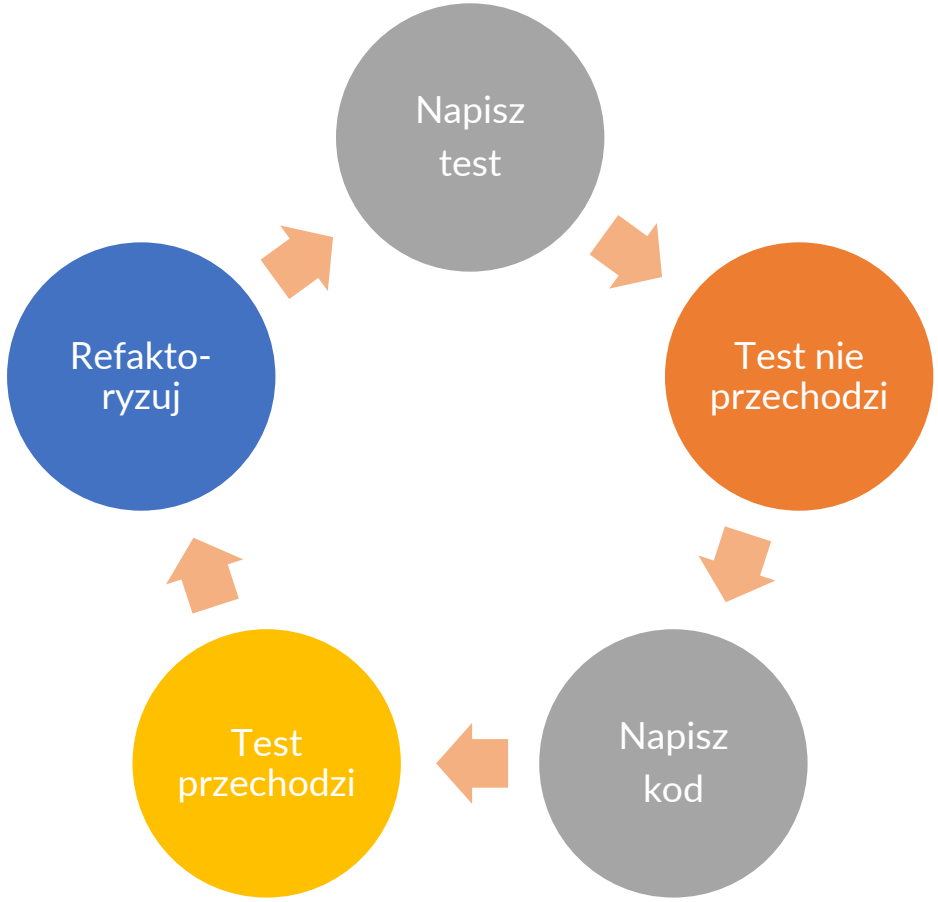
Etap 2: Test nie przechodzi



- Stare testy powinny przechodzić, jak do tej pory
- Nowe testy powinny zgłaszać niepowodzenia

Teoria część 2.

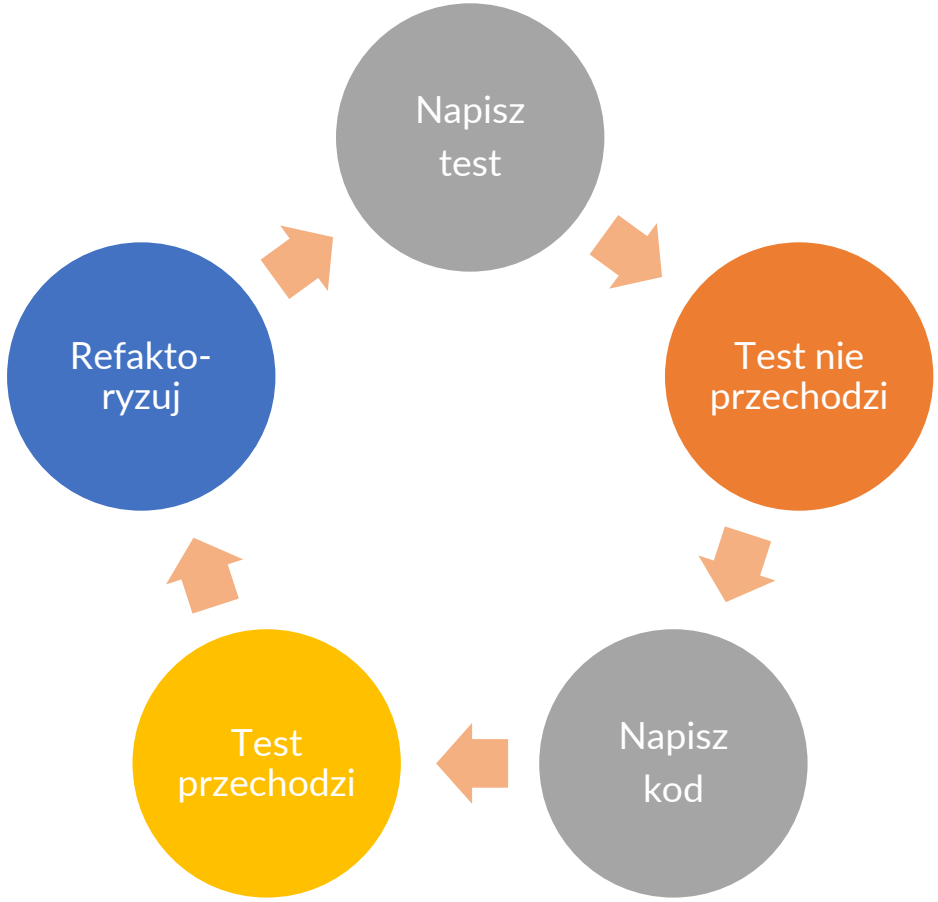
Etap 3: Napisz kod



- Piszemy fragment kodu
- Kod może być niedoskonały
- Najważniejsze, aby testy przechodziły
- Teoretycznie nie powinniśmy wracać do testu. W praktyce – często się okazuje, że czegoś jeszcze nie przetestowaliśmy i trzeba wrócić do testu
- Nie należy pisać kodu, którego nie obejmują przygotowane testy

Teoria część 2.

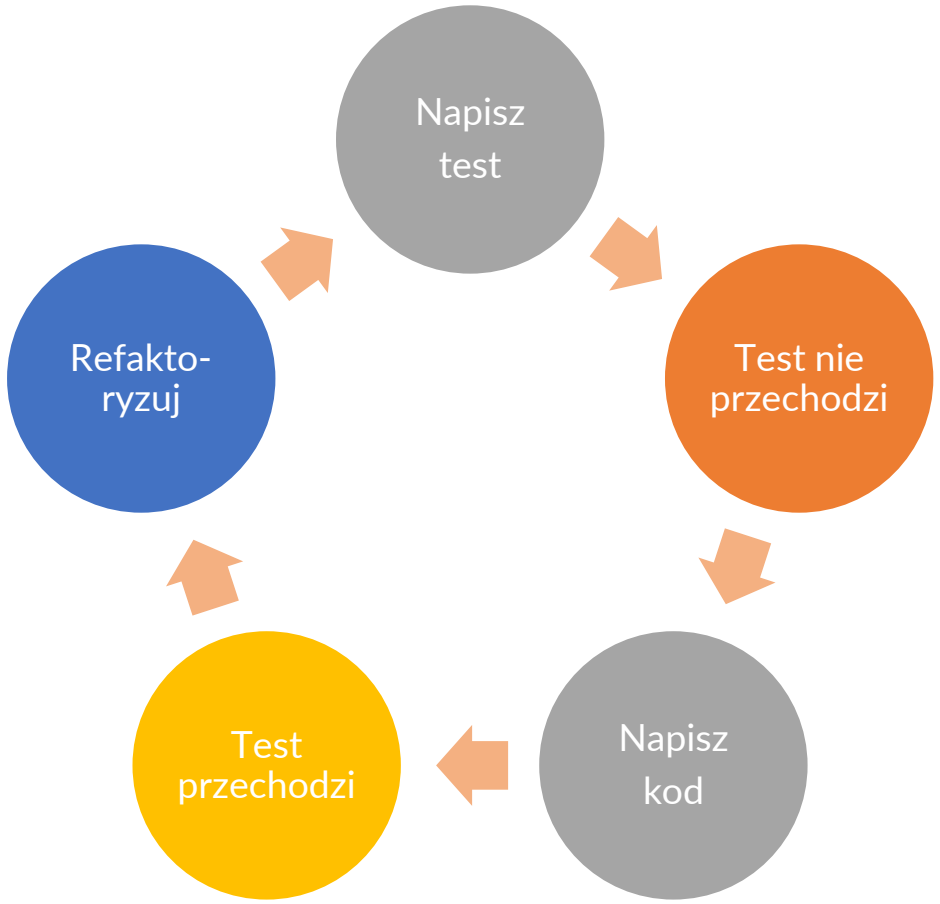
Etap 4: Test przechodzi



- Dopóki wszystkie testy nie przechodzą, wracamy do etapu 3

Teoria część 2.

Etap 5: Refaktoryzuj



- Poprawiamy kod, aby był dobry i elegancki
- Poprawiamy kod, aby przeszedł code review
- Poprawiamy kod, aby nie miał zbędnych śmieci

- W każdym momencie można wrócić do etapu 4.

- Celem refaktoryzacji **nie jest** nowa funkcjonalność
- Celem refaktoryzacji jest **podniesienie jakości wytwarzanego oprogramowania** oraz czytelność (porządek)

Biblioteki

Istnieją dwie konkurencyjne biblioteki: **pytest** i **unittest**

Pytest

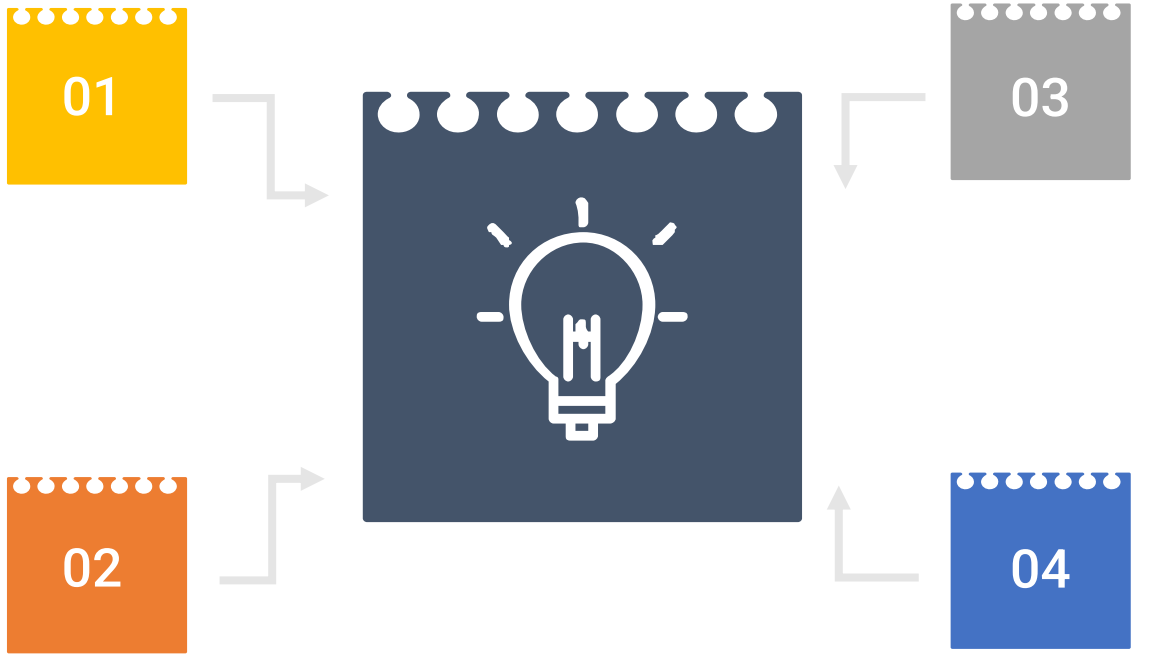
nowoczesny framework do uruchamiania testów automatycznych w języku Python

Unitest

jest implementacją standardu cross platformowego XUnit najpopularniejszego jako javowy JUnit

Przykład FizzBuzz

Algorytm FizzBuzz* przyjmuje liczbę i zwraca:



* PyCharm - pliki z kodem dostępne osobno

TDD

Podsumowanie

TDD

Wymusza na programiście przemyślenie, co dokładnie dany kod ma robić.

TDD

Testowane są bardzo małe fragmenty kodu – wymusza to zadbanie o architekturę systemu, o modularność, o to, aby metody nie były zbyt duże.

TDD

Powstaje system czytelny, modularny, bez dużych chaotycznych metod. Dobrze nazwane testy TDD odpowiadają za dokumentację,

TDD

Bardzo dobre do testów regresyjnych

TDD

Mamy pewność, że kod nie robi ani za dużo, ani za mało

Ćwiczenie

Napisz program*, który liczy, ilu studentów może przebywać w budynku:

01

Program przyjmuje dane:
- metraż podstawy budynku,
- liczba pięter,
- środki dezynfekcyjne (są bądź nie)

02

Program posiada funkcję liczącą ile osób może przebywać w budynku patrząc na metraż i liczbę pięter:
1 osoba na 7.5 metra kwadratowego

03

Program posiada funkcję przeliczającą liczbę osób patrząc na środki dezynfekcyjne.
- jeśli są: max liczba osób zostaje zwiększona o 10%,
- jeśli nie: max liczba osób zostaje zmniejszona o 50%

* PyCharm - pliki z kodem dostępne osobno

Ćwiczenie

Napisz program*, który liczy, ilu studentów może przebywać w budynku

Przykładowe działanie programu:

Wprowadź długość budynku (w metrach): **50**

Wprowadź szerokość budynku (w metrach): **20**

Wprowadź liczbę pięter: **3**

Budynek ma **1000** metrów kwadratowych razy **3** piętra.
W sumie **3000** metrów kwadratowych.
Może w nim przebywać **400** osób.

W budynku są środki dezynfekcyjne, więc max liczba osób to **440**.

lub

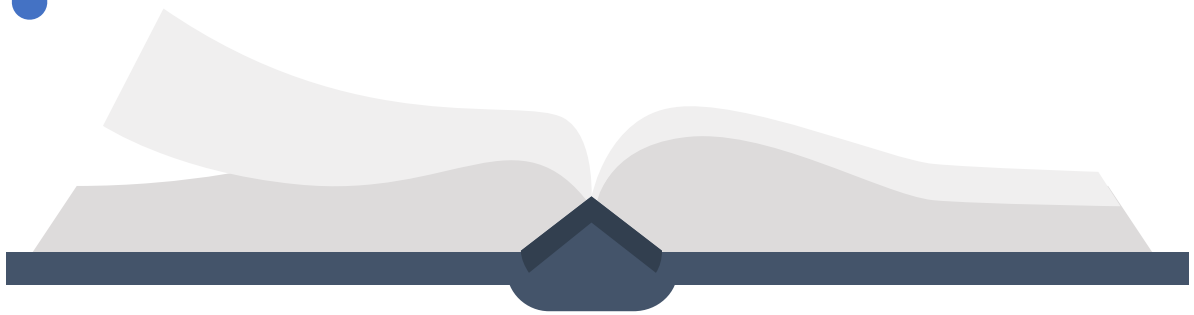
W budynku nie ma środków dezynfekcyjnych, więc max liczba osób to **200**.

* PyCharm - pliki z kodem dostępne osobno

Materiały

Bibioteka unittest: https://docs.python.org/3/library/unittest.html

„PYTHON dla testera”, Piotr Wróblewski



„Python 3. Projekty dla początkujących i pasjonatów”, Adam Jurkiewicz

Koniec

Dziękuję za uwagę