



## **B6B36ZAL: Základy algoritmizace**

### **PEP 8. Čistý kód.**

Petr Pošík

Katedra kybernetiky  
ČVUT FEL



# Formátování kódu



# Proč je důležité formátování?

---

## Formátování kódu

---

- Proč?
- PEP 8
- Doporučení

## Clean Code

---

## Příklad

---

- Kód je čten mnohem častěji než psán.
- Na čitelnosti záleží.
- Použitý formát/styl by měl čtenáři pomáhat
  - *vizuálně sdružovat* věci, které spolu souvisí, *vizuálně oddělovat* věci, které nejsou “těsně” svázané;
  - *vizuálně evokovat* začátek a konec výrazu, bloku kódu, funkce, metody, třídy, atd.; a neměl by čtenáře mást.



# PEP 8

---

## Python Enhancement Proposal 8: **Style guide for Python code**

- <https://www.python.org/dev/peps/pep-0008/>
- *Doporučení*, jak formátovat kód tak, aby formát nebránil v jeho snadném pochopení.

---

### Formátování kódu

- Proč?
- **PEP 8**
- Doporučení

---

### Clean Code

---

### Příklad

---



# PEP 8

---

## Python Enhancement Proposal 8: **Style guide for Python code**

- <https://www.python.org/dev/peps/pep-0008/>
- *Doporučení*, jak formátovat kód tak, aby formát nebránil v jeho snadném pochopení.

Konzistence s PEP 8 je důležitá, ale

- konzistence v rámci projektu je důležitější,
- konzistence v rámci jednoho modulu je nejdůležitější.

---

Formátování kódu

- Proč?
- **PEP 8**
- Doporučení

---

Clean Code

---

Příklad



# PEP 8

## Python Enhancement Proposal 8: **Style guide for Python code**

- <https://www.python.org/dev/peps/pep-0008/>
- *Doporučení*, jak formátovat kód tak, aby formát nebránil v jeho snadném pochopení.

Konzistence s PEP 8 je důležitá, ale

- konzistence v rámci projektu je důležitější,
- konzistence v rámci jednoho modulu je nejdůležitější.

Dobré důvody ignorovat určité doporučení z PEP 8:

- když doporučení dělá kód méně čitelným;
- když je třeba zachovat konzistenci v rámci modulu, či projektu (např. z historických důvodů (ale možná je to příležitost jak vyčistit nečitelný kód?));
- když je kód starší než doporučení a není žádný jiný důvod daný kód modifikovat;
- když kód musí zůstat kompatibilní se starší verzí Pythonu, která ještě nepodporuje doporučovaný způsob.

---

### Formátování kódu

- Proč?
- **PEP 8**
- Doporučení

---

### Clean Code

---

### Příklad



# PEP 8

## Python Enhancement Proposal 8: [Style guide for Python code](https://www.python.org/dev/peps/pep-0008/)

- <https://www.python.org/dev/peps/pep-0008/>
- *Doporučení*, jak formátovat kód tak, aby formát nebránil v jeho snadném pochopení.

Konzistence s PEP 8 je důležitá, ale

- konzistence v rámci projektu je důležitější,
- konzistence v rámci jednoho modulu je nejdůležitější.

Dobré důvody ignorovat určité doporučení z PEP 8:

- když doporučení dělá kód méně čitelným;
- když je třeba zachovat konzistenci v rámci modulu, či projektu (např. z historických důvodů (ale možná je to příležitost jak vyčistit nečitelný kód?));
- když je kód starší než doporučení a není žádný jiný důvod daný kód modifikovat;
- když kód musí zůstat kompatibilní se starší verzí Pythonu, která ještě nepodporuje doporučovaný způsob.

Modul pep8:

- <https://pypi.python.org/pypi/pep8>
- Automatická kontrola formátování kódu podle doporučení PEP 8.

### Formátování kódu

- Proč?
- PEP 8
- Doporučení

### Clean Code

### Příklad



# Některá doporučení PEP 8

---

Rozložení a organizace kódu:

- Pro *odsazení* používejte 4 mezery (nikoli tabulátory).
- Omezte *délku řádků* na 79 znaků, 72 pro docstringy a komentáře.
- Používejte výchozí *kódování UTF-8*; jinak jej specifikujte v úvodu modulu např. takto:  

```
# -*- encoding: latin_1 -*-
```
- **importy** umístěte na začátek souboru, každý modul na zvláštní řádek
- Definice funkcí udržujte *pohromadě*.
- Definice tříd a funkcí nejvyšší úrovně *odděluje* 2 prázdnými řádky. Definice vnořených funkcí a metod odděluje 1 prázdným řádkem.
- Příkazy a volání funkcí na nejvyšší úrovni udržujte *pohromadě na konci programu*.

---

## Formátování kódu

- Proč?
- PEP 8
- Doporučení

---

## Clean Code

---

## Příklad





# Některá doporučení PEP 8

## Rozložení a organizace kódu:

- Pro *odsazení* používejte 4 mezery (nikoli tabulátory).
- Omezte *délku řádků* na 79 znaků, 72 pro docstringy a komentáře.
- Používejte výchozí *kódování UTF-8*; jinak jej specifikujte v úvodu modulu např. takto:  
`# -*- encoding: latin_1 -*-`
- **importy** umístěte na začátek souboru, každý modul na zvláštní řádek
- Definice funkcí udržujte *pohromadě*.
- Definice tříd a funkcí nejvyšší úrovně *odděluje* 2 prázdnými řádky. Definice vnořených funkcí a metod odděluje 1 prázdným řádkem.
- Příkazy a volání funkcí na nejvyšší úrovni udržujte *pohromadě na konci programu*.

## Komentáře a docstringy:

- Komentáře pište v angličtině. (Výjimkou jsou případy, kdy jste si na 120 % jistí, že váš kód nebude číst nikdo, kdo by nerozuměl vašemu jazyku.)

### Formátování kódu

- Proč?
- PEP 8
- Doporučení

### Clean Code

### Příklad



# Některá doporučení PEP 8

## Formátování kódu

- Proč?
- PEP 8
- Doporučení

## Clean Code

## Příklad

### Rozložení a organizace kódu:

- Pro *odsazení* používejte 4 mezery (nikoli tabulátory).
- Omezte *délku řádků* na 79 znaků, 72 pro docstringy a komentáře.
- Používejte výchozí *kódování UTF-8*; jinak jej specifikujte v úvodu modulu např. takto:  

```
# -*- encoding: latin_1 -*-
```
- **importy** umístěte na začátek souboru, každý modul na zvláštní řádek
- Definice funkcí udržujte *pohromadě*.
- Definice tříd a funkcí nejvyšší úrovně *odděluje* 2 prázdnými řádky. Definice vnořených funkcí a metod odděluje 1 prázdným řádkem.
- Příkazy a volání funkcí na nejvyšší úrovni udržujte *pohromadě na konci programu*.

### Komentáře a docstringy:

- Komentáře pište v angličtině. (Výjimkou jsou případy, kdy jste si na 120 % jistí, že váš kód nebude číst nikdo, kdo by nerozuměl vašemu jazyku.)

### Konvence pro pojmenování:

- lowercase\_with\_underscores pro proměnné, funkce, moduly a balíky;
- CamelCase pro názvy tříd a výjimek;
- CAPITAL\_LETTERS\_WITH\_UNDERSCORES pro “konstanty”.



# Některá doporučení PEP 8

## Rozložení a organizace kódu:

- Pro *odsazení* používejte 4 mezery (nikoli tabulátory).
- Omezte *délku řádků* na 79 znaků, 72 pro docstringy a komentáře.
- Používejte výchozí *kódování UTF-8*; jinak jej specifikujte v úvodu modulu např. takto:  

```
# -*- encoding: latin_1 -*-
```
- **importy** umístěte na začátek souboru, každý modul na zvláštní řádek
- Definice funkcí udržujte *pohromadě*.
- Definice tříd a funkcí nejvyšší úrovně *odděluje* 2 prázdnými řádky. Definice vnořených funkcí a metod odděluje 1 prázdným řádkem.
- Příkazy a volání funkcí na nejvyšší úrovni udržujte *pohromadě na konci programu*.

## Komentáře a docstringy:

- Komentáře pište v angličtině. (Výjimkou jsou případy, kdy jste si na 120 % jistí, že váš kód nebude číst nikdo, kdo by nerozuměl vašemu jazyku.)

## Konvence pro pojmenování:

- lowercase\_with\_underscores pro proměnné, funkce, moduly a balíky;
- CamelCase pro názvy tříd a výjimek;
- CAPITAL\_LETTERS\_WITH\_UNDERSCORES pro “konstanty”.

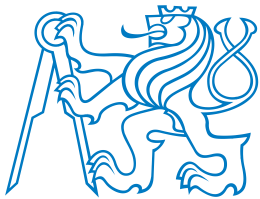
A to stačí, aby byl můj kód “čistý”???

### Formátování kódu

- Proč?
- PEP 8
- Doporučení

### Clean Code

### Příklad



# Clean Code

Zpracováno podle  
**Robert C. Martin: *Clean Code: A Handbook of Agile Software Craftsmanship*,  
Prentice Hall, 2008.**

# Který kód je čistší? A proč?

---

Eratostenovo síto

# Který kód je čistší? A proč?

## Eratostenovo síto

Dvě implementace téhož algoritmu:

```
1 def generate_primes_up_to(max_value):
2     """Find primes up to the max_value
3     using the Sieve of Eratosthenes.
4     """
5     if max_value >= 2: # There are some primes
6         # Initialize the list (incl. 0)
7         f = [True for i in range(max_value+1)]
8         # Get rid of the known non-primes
9         f[0] = f[1] = False
10        # Run the sieve
11        for i in range(2, len(f)):
12            if f[i]: # i is still a candidate
13                # mark its multiples as not prime
14                for j in range(2*i, len(f), i):
15                    f[j] = False
16        # Find the primes and put them in a list
17        primes = [i for i in range(len(f)) if f[i]]
18        return primes
19    else: # max_value < 2
20        # no primes, return empty list
21        return list()
```

```
1 PRIME = True
2 NONPRIME = False
3
4 def generate_primes_up_to(max_value):
5     """Find primes up to the max_value
6     using the Sieve of Eratosthenes.
7     """
8     if max_value < 2:
9         return []
10    else:
11        candidates = init_integers_up_to(max_value)
12        mark_non_primes(candidates)
13        return collect_remaining(candidates)
```

# Který kód je čistší? A proč?

## Eratostenovo síto

Dvě implementace téhož algoritmu:

```
1 def generate_primes_up_to(max_value):
2     """Find primes up to the max_value
3     using the Sieve of Eratosthenes.
4     """
5     if max_value >= 2: # There are some primes
6         # Initialize the list (incl. 0)
7         f = [True for i in range(max_value+1)]
8         # Get rid of the known non-primes
9         f[0] = f[1] = False
10        # Run the sieve
11        for i in range(2, len(f)):
12            if f[i]: # i is still a candidate
13                # mark its multiples as not prime
14                for j in range(2*i, len(f), i):
15                    f[j] = False
16        # Find the primes and put them in a list
17        primes = [i for i in range(len(f)) if f[i]]
18        return primes
19    else: # max_value < 2
20        # no primes, return empty list
21        return list()
```

```
1 PRIME = True
2 NONPRIME = False
3
4 def generate_primes_up_to(max_value):
5     """Find primes up to the max_value
6     using the Sieve of Eratosthenes.
7     """
8     if max_value < 2:
9         return []
10    else:
11        candidates = init_integers_up_to(max_value)
12        mark_non_primes(candidates)
13        return collect_remaining(candidates)
14
15 def init_integers_up_to(max_value):
16     return [PRIME for i in range(max_value+1)]
17
18 def mark_non_primes(candidates):
19     # Mark 0 and 1, they are not primes.
20     candidates[0] = candidates[1] = NONPRIME
21     for number in range(2, len(candidates)):
22         if candidates[number] == PRIME:
23             mark_multiples_of(number, candidates)
24
25 def mark_multiples_of(number, candidates):
26     for multiple in range(
27         2*number, len(candidates), number):
28         candidates[multiple] = NONPRIME
29
30 def collect_remaining(candidates):
31     primes = [i for i in range(len(candidates))
32              if candidates[i] == PRIME]
33     return primes
```



# Co je “clean code”?

---

Bjarne Stroustrup, autor jazyka C++ a knihy “The C++ Programming Language”:

I like my code to be **elegant and efficient**. The logic should be **straightforward** to make it hard for bugs to hide, the **dependencies minimal** to ease maintenance, error handling complete according to an articulated strategy, and **performance close to optimal** so as not to tempt people to make the code messy with unprincipled optimizations. **Clean code does one thing well.**

---

## Formátování kódu

### Clean Code

- Který kód je čistší?  
A proč?
- Co je “clean code”?
- Čistý kód v praxi
- Smysluplná jména
- Eratostenovo síto:  
smysluplná jména
- Komentáře
- Eratostenovo síto:  
komentáře
- Funkce a metody
- Eratostenovo síto:  
funkce
- Eratostenovo síto:  
pomocná třída?

---

## Příklad





# Co je “clean code”?

---

Bjarne Stroustrup, autor jazyka C++ a knihy “The C++ Programming Language”:

I like my code to be **elegant and efficient**. The logic should be **straightforward** to make it hard for bugs to hide, the **dependencies minimal** to ease maintenance, error handling complete according to an articulated strategy, and **performance close to optimal** so as not to tempt people to make the code messy with unprincipled optimizations. **Clean code does one thing well.**

Grady Booch, autor knihy “Object Oriented Analysis and Design with Applications”:

Clean code is **simple and direct**. Clean code **reads like well-written prose**. Clean code **never obscures the designer’s intent** but rather is full of **crisp abstractions** and **straightforward lines of control**.

---

## Formátování kódu

### Clean Code

- Který kód je čistší?  
A proč?
- Co je “clean code”?
- Čistý kód v praxi
- Smysluplná jména
- Eratostenovo síto:  
smysluplná jména
- Komentáře
- Eratostenovo síto:  
komentáře
- Funkce a metody
- Eratostenovo síto:  
funkce
- Eratostenovo síto:  
pomocná třída?

---

## Příklad



## Co je “clean code”?

---

Bjarne Stroustrup, autor jazyka C++ a knihy “The C++ Programming Language”:

I like my code to be **elegant and efficient**. The logic should be **straightforward** to make it hard for bugs to hide, the **dependencies minimal** to ease maintenance, error handling complete according to an articulated strategy, and **performance close to optimal** so as not to tempt people to make the code messy with unprincipled optimizations. **Clean code does one thing well.**

Grady Booch, autor knihy “Object Oriented Analysis and Design with Applications”:

Clean code is **simple and direct**. Clean code **reads like well-written prose**. Clean code **never obscures the designer’s intent** but rather is full of **crisp abstractions and straightforward lines of control**.

Dave Thomas, zakladatel firmy OTI (převzata firmou IBM v roce 1996), kmotr Eclipse:

Clean code can be read, and enhanced by a developer other than its **original author**. It has **unit and acceptance tests**. It has **meaningful names**. It provides one way rather than many ways for doing one thing. It has **minimal dependencies**, which are explicitly defined, and **provides a clear and minimal API**.

---

### Formátování kódu

#### Clean Code

- Který kód je čistší?  
A proč?
- Co je “clean code”?
- Čistý kód v praxi
- Smysluplná jména
- Eratostenovo síto:  
smysluplná jména
- Komentáře
- Eratostenovo síto:  
komentáře
- Funkce a metody
- Eratostenovo síto:  
funkce
- Eratostenovo síto:  
pomocná třída?

---

### Příklad



# Čistý kód v praxi

---

Code review:

## Formátování kódu

### Clean Code

- Který kód je čistší?  
A proč?
- Co je “clean code”?
- Čistý kód v praxi
- Smysluplná jména
- Eratostenovo síto:  
smysluplná jména
- Komentáře
- Eratostenovo síto:  
komentáře
- Funkce a metody
- Eratostenovo síto:  
funkce
- Eratostenovo síto:  
pomocná třída?

### Příklad



# Čistý kód v praxi

Code review:

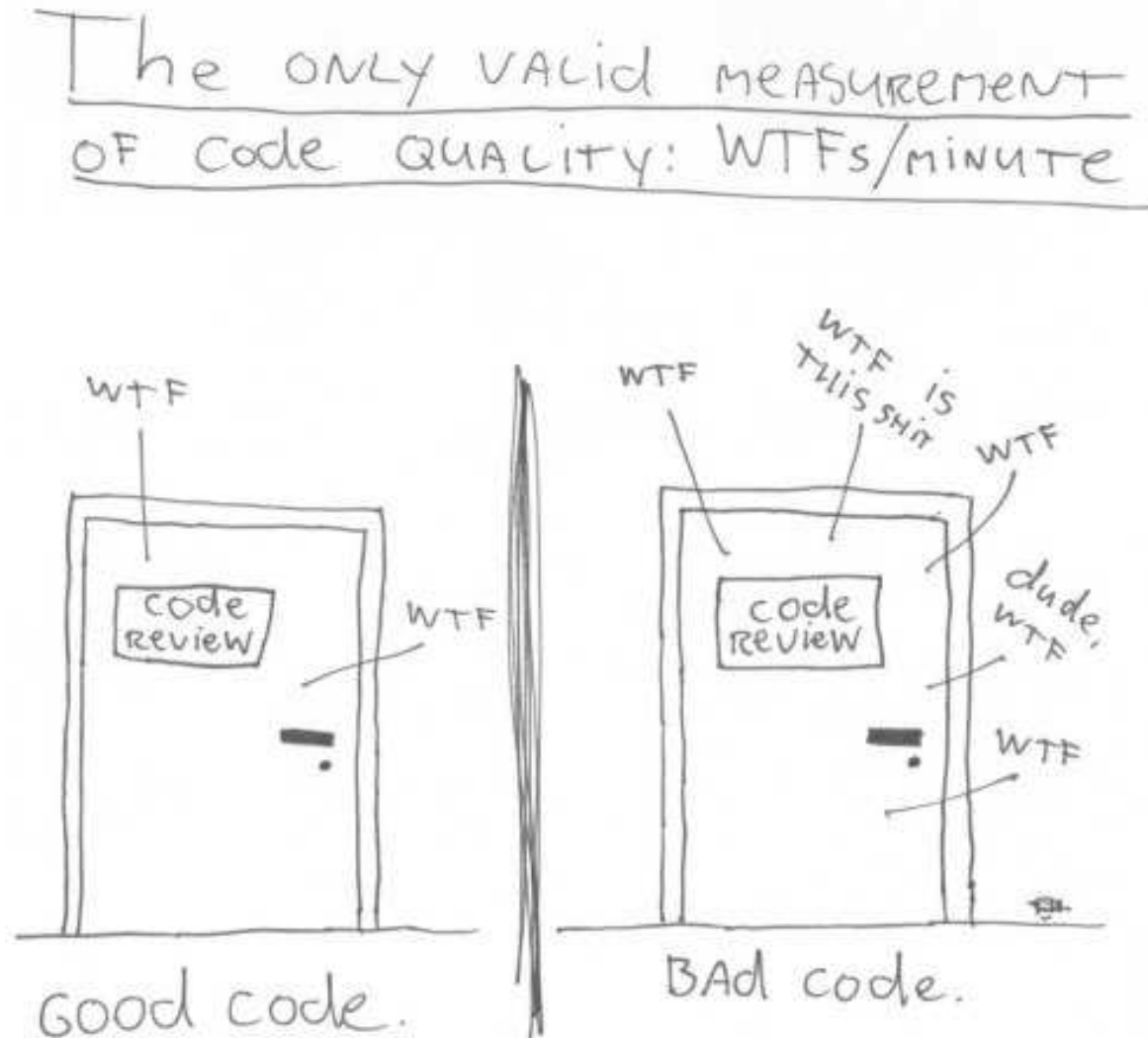
Jediné správné měřítko kvality kódu: Co-to-k-čerty za minutu

## Formátování kódu

### Clean Code

- Který kód je čistší?  
A proč?
- Co je "clean code"?
- Čistý kód v praxi
- Smysluplná jména
- Eratostenovo síto:  
smysluplná jména
- Komentáře
- Eratostenovo síto:  
komentáře
- Funkce a metody
- Eratostenovo síto:  
funkce
- Eratostenovo síto:  
pomocná třída?

### Příklad



(c) 2008 Focus Shift/OSNews/Thom Holwerda - <http://www.osnews.com/comics>



# Smysluplná jména

- Vymyslet dobrá jména je **velmi těžké**! Nebojte se jméno změnit, přijdete-li na lepší!
- Dobré jméno **odhaluje autorův záměr** (intention-revealing). Pokud jméno vyžaduje komentář, neodhaluje záměr. Porovnejte:

- `self.d = 0` *# Elapsed time in days*

- `self.elapsed_days = 0`

## Formátování kódu

### Clean Code

- Který kód je čistší? A proč?
- Co je “clean code”?
- Čistý kód v praxi
- **Smysluplná jména**
- Eratostenovo síto: smysluplná jména
- Komentáře
- Eratostenovo síto: komentáře
- Funkce a metody
- Eratostenovo síto: funkce
- Eratostenovo síto: pomocná třída?

## Příklad



# Smysluplná jména

- Vymyslet dobrá jména je **velmi těžké**! Nebojte se jméno změnit, přijdete-li na lepší!
- Dobré jméno **odhaluje autorův záměr** (intention-revealing). Pokud jméno vyžaduje komentář, neodhaluje záměr. Porovnejte:

- `self.d = 0` *# Elapsed time in days*

- `self.elapsed_days = 0`

Porovnejte (co když i ten komentář chybí?):

- `st("obama", 20, False, True)`

## Formátování kódu

### Clean Code

- Který kód je čistší? A proč?
- Co je “clean code”?
- Čistý kód v praxi
- **Smysluplná jména**
- Eratostenovo síto: smysluplná jména
- Komentáře
- Eratostenovo síto: komentáře
- Funkce a metody
- Eratostenovo síto: funkce
- Eratostenovo síto: pomocná třída?

## Příklad



# Smysluplná jména

- Vymyslet dobrá jména je **velmi těžké**! Nebojte se jméno změnit, přijdete-li na lepší!
- Dobré jméno **odhaluje autorův záměr** (intention-revealing). Pokud jméno vyžaduje komentář, neodhaluje záměr. Porovnejte:

- `self.d = 0` *# Elapsed time in days*

- `self.elapsed_days = 0`

Porovnejte (co když i ten komentář chybí?):

- `st("obama", 20, False, True)`

- `search_twitter("obama", 20, False, True)`

## Formátování kódu

### Clean Code

- Který kód je čistší? A proč?
- Co je “clean code”?
- Čistý kód v praxi
- **Smysluplná jména**
- Eratostenovo síto: smysluplná jména
- Komentáře
- Eratostenovo síto: komentáře
- Funkce a metody
- Eratostenovo síto: funkce
- Eratostenovo síto: pomocná třída?

## Příklad



# Smysluplná jména

## Formátování kódu

### Clean Code

- Který kód je čistší?  
A proč?
- Co je “clean code”?
- Čistý kód v praxi
- **Smysluplná jména**
- Eratostenovo síto:  
smysluplná jména
- Komentáře
- Eratostenovo síto:  
komentáře
- Funkce a metody
- Eratostenovo síto:  
funkce
- Eratostenovo síto:  
pomocná třída?

### Příklad

- Vymyslet dobrá jména je **velmi těžké**! Nebojte se jméno změnit, přijdete-li na lepší!
  - Dobré jméno **odhaluje autorův záměr** (intention-revealing).  
Pokud jméno vyžaduje komentář, neodhaluje záměr. Porovnejte:
    - `self.d = 0 # Elapsed time in days`
    - `self.elapsed_days = 0`
- Porovnejte (co když i ten komentář chybí?):
- `st("obama", 20, False, True)`
  - `search_twitter("obama", 20, False, True)`
- Názvy tříd: **podstatná jména** (s přívlastky):
    - Customer, WikiPage, AddressParser, Filter, StupidFilter, Corpus, TrainingCorpus
  - Názvy funkcí/metod: **slovesa** (s předmětem):
    - post\_payment, delete\_page, save, train, test, get\_email
  - Jeden termín pro jeden koncept! Nepoužívejte stejné slovo k více účelům!
  - Nebojte se dlouhých jmen!
    - Dlouhé popisné jméno je lepší než dlouhý popisný komentář.
    - Čím delší oblast platnosti proměnné, tím popisnější jméno by měla mít.
  - Používejte **pojmenované konstanty** místo magických čísel v kódu! Porovnejte:
    - `if opponents_move == True:`
    - `if opponents_move == DEFECT:`



# Eratostenovo síto: smysluplná jména

```
1 def generate_primes_up_to(max_value):
2     """Find primes up to the max_value
3     using the Sieve of Eratosthenes.
4     """
5     if max_value >= 2: # There are some primes
6         # Initialize the list (incl. 0)
7         f = [True for i in range(max_value+1)]
8         # Get rid of the known non-primes
9         f[0] = f[1] = False
10        # Run the sieve
11        for i in range(2, len(f)):
12            if f[i]: # i is still a candidate
13                # mark its multiples as not prime
14                for j in range(2*i, len(f), i):
15                    f[j] = False
16        # Find the primes and put them in a list
17        primes = [i for i in range(len(f)) if f[i]]
18        return primes
19    else: # max_value < 2
20        # no primes, return empty list
21        return list()
```

# Eratostenovo síto: smysluplná jména

```
1 def generate_primes_up_to(max_value):
2     """Find primes up to the max_value
3     using the Sieve of Eratosthenes.
4     """
5     if max_value >= 2: # There are some primes
6         # Initialize the list (incl. 0)
7         f = [True for i in range(max_value+1)]
8         # Get rid of the known non-primes
9         f[0] = f[1] = False
10        # Run the sieve
11        for i in range(2, len(f)):
12            if f[i]: # i is still a candidate
13                # mark its multiples as not prime
14                for j in range(2*i, len(f), i):
15                    f[j] = False
16        # Find the primes and put them in a list
17        primes = [i for i in range(len(f)) if f[i]]
18        return primes
19    else: # max_value < 2
20        # no primes, return empty list
21        return list()
```

```
1 PRIME = True
2 NONPRIME = False
3
4 def generate_primes_up_to(max_value):
5     """Find primes up to the max_value
6     using the Sieve of Eratosthenes.
7     """
8     if max_value >= 2: # There are some primes
9         # Initialize the list (incl. 0)
10        candidates = [
11            PRIME for i in range(max_value+1)]
12        # Get rid of the known non-primes
13        candidates[0] = candidates[1] = NONPRIME
14        # Run the sieve
15        for number in range(2, len(candidates)):
16            if candidates[number] == PRIME:
17                # mark its multiples as not prime
18                for multiple in range(
19                    2*number, len(candidates), number):
20                    candidates[multiple] = NONPRIME
21        # Find the primes and put them in a list
22        primes = [i for i in range(len(candidates))
23                    if candidates[i] == PRIME]
24        return primes
25    else: # max_value < 2
26        # no primes, return empty list
27        return list()
```

Další smysluplná jména budou následovat!!!



# Komentáře

Čistý kód komentáře (skoro) nepotřebuje!

- Komentáře kompenzují naše selhání vyjádřit se v prog. jazyce. Porovnej:

```
1 # Check whether point lies inside the unit circle
2 if point[0]**2 + point[1]**2 <= 1:
```

versus

```
1 if is_inside_unit_circle(point):
```

- Komentáře lžou! Ne vždy a ne záměrně, ale až příliš často!
- Nepřesné komentáře jsou horší než žádné komentáře!
- Komentáře nenapraví špatný kód!
- Dobré komentáře:
  - (do)vysvětlení, (do)upřesnění
  - zdůraznění, varování před následky
  - TODOs
- Špatné komentáře:
  - staré (už neplatné), bezvýznamné, nevhodné, redundantní, nebo zavádějící komentáře
  - komentáře z povinnosti
  - zakomentovaný kód
  - nelokální nebo nadbytečné informace

## Formátování kódu

### Clean Code

- Který kód je čistší?  
A proč?
- Co je “clean code”?
- Čistý kód v praxi
- Smysluplná jména
- Eratostenovo síto:  
smysluplná jména
- **Komentáře**
- Eratostenovo síto:  
komentáře
- Funkce a metody
- Eratostenovo síto:  
funkce
- Eratostenovo síto:  
pomocná třída?

### Příklad

# Eratostenovo síto: komentáře

---

```
1 # This function generates prime numbers up to
2 # a user specified maximum. The algorithm
3 # used is the Sieve of Eratosthenes
4 #
5 # Eratosthenes of Cyrene, b. c. 276 BC,
6 # Cyrene, Libya -- d. c. 194 BC, Alexandria.
7 # The first man to calculate the circumference
8 # of the Earth. Also known for working on
9 # calendars with leap years and ran
10 # the library at Alexandria.
11 #
12 # The algorithm is quite simple.
13 # Given an array of integers starting at 2,
14 # cross out all multiples of 2.
15 # Find the next uncrossed integer,
16 # and cross out all of its multiples.
17 # Repeat until you have passed
18 # the maximum value.
19 #
20 # @author hugo
21 # @version 1
```

# Eratostenovo síto: komentáře

---

```
1 # This function generates prime numbers up to
2 # a user specified maximum. The algorithm
3 # used is the Sieve of Eratosthenes
4 #
5 # Eratosthenes of Cyrene, b. c. 276 BC,
6 # Cyrene, Libya -- d. c. 194 BC, Alexandria.
7 # The first man to calculate the circumference
8 # of the Earth. Also known for working on
9 # calendars with leap years and ran
10 # the library at Alexandria.
11 #
12 # The algorithm is quite simple.
13 # Given an array of integers starting at 2,
14 # cross out all multiples of 2.
15 # Find the next uncrossed integer,
16 # and cross out all of its multiples.
17 # Repeat until you have passed
18 # the maximum value.
19 #
20 # @author hugo
21 # @version 1
```

```
1 # This function generates prime numbers up to
2 # a user specified maximum. The algorithm
3 # used is the Sieve of Eratosthenes.
4 # Given an array of integers starting at 2,
5 # cross out all multiples of 2.
6 # Find the next uncrossed integer,
7 # and cross out all of its multiples.
8 # Repeat until you have passed
9 # the maximum value.
10 #
11 # @author hugo
12 # @version 1
```

Za chvíli se zbavíme dalších komentářů!



# Funkce a metody

## Formátování kódu

### Clean Code

- Který kód je čistší?  
A proč?
- Co je “clean code”?
- Čistý kód v praxi
- Smysluplná jména
- Eratostenovo síto:  
smysluplná jména
- Komentáře
- Eratostenovo síto:  
komentáře
- **Funkce a metody**
- Eratostenovo síto:  
funkce
- Eratostenovo síto:  
pomocná třída?

### Příklad

- Funkce by měly být **krátké!** (A ještě kratší!)
- Funkce by měla **dělat právě 1 věc** a měla by ji dělat dobře. (A bez vedlejších efektů.)
- Funkce dlouhé méně než 5 řádků
  - většinou dělají právě 1 věc,
  - mohou mít přesné a výstižné jméno,
  - nemohou obsahovat vnořené příkazy **if**, **for**, ..., a
  - bloky uvnitř příkazů **if**, **for**, ... jsou pouze 1-2 řádky dlouhé.
- Krátké funkce **umožňují testovat dílčí části** algoritmu!
- Sekce uvnitř funkcí/metod:
  - Jasná indikace toho, že funkce/metoda nedělá jen 1 věc a měla by být rozdělena.
- Parametry funkcí/metod:
  - Udržujte jejich počet malý! 0, 1, 2, výjimečně 3.
  - Vytvořte jméno tak, aby evokovalo pořadí argumentů.
  - Boolovské argumenty funkcí často značí, že funkce nedělá 1 věc! Rozdělte ji.
- Při volání používejte častěji **keyword arguments!** Porovnejte:
  - `st("obama", 20, False, True)`



# Funkce a metody

## Formátování kódu

### Clean Code

- Který kód je čistší?  
A proč?
- Co je “clean code”?
- Čistý kód v praxi
- Smysluplná jména
- Eratostenovo síto:  
smysluplná jména
- Komentáře
- Eratostenovo síto:  
komentáře
- **Funkce a metody**
- Eratostenovo síto:  
funkce
- Eratostenovo síto:  
pomocná třída?

### Příklad

- Funkce by měly být **krátké!** (A ještě kratší!)
- Funkce by měla **dělat právě 1 věc** a měla by ji dělat dobře. (A bez vedlejších efektů.)
- Funkce dlouhé méně než 5 řádků
  - většinou dělají právě 1 věc,
  - mohou mít přesné a výstižné jméno,
  - nemohou obsahovat vnořené příkazy **if**, **for**, ..., a
  - bloky uvnitř příkazů **if**, **for**, ... jsou pouze 1-2 řádky dlouhé.
- Krátké funkce **umožňují testovat dílčí části** algoritmu!
- Sekce uvnitř funkcí/metod:
  - Jasná indikace toho, že funkce/metoda nedělá jen 1 věc a měla by být rozdělena.
- Parametry funkcí/metod:
  - Udržujte jejich počet malý! 0, 1, 2, výjimečně 3.
  - Vytvořte jméno tak, aby evokovalo pořadí argumentů.
  - Boolovské argumenty funkcí často značí, že funkce nedělá 1 věc! Rozdělte ji.
- Při volání používejte častěji **keyword arguments!** Porovnejte:
  - `st("obama", 20, False, True)`
  - `search_twitter("obama", 20, False, True)`



# Funkce a metody

## Formátování kódu

### Clean Code

- Který kód je čistší?  
A proč?
- Co je “clean code”?
- Čistý kód v praxi
- Smysluplná jména
- Eratostenovo síto:  
smysluplná jména
- Komentáře
- Eratostenovo síto:  
komentáře
- **Funkce a metody**
- Eratostenovo síto:  
funkce
- Eratostenovo síto:  
pomocná třída?

### Příklad

- Funkce by měly být **krátké!** (A ještě kratší!)
- Funkce by měla **dělat právě 1 věc** a měla by ji dělat dobře. (A bez vedlejších efektů.)
- Funkce dlouhé méně než 5 řádků
  - většinou dělají právě 1 věc,
  - mohou mít přesné a výstižné jméno,
  - nemohou obsahovat vnořené příkazy **if**, **for**, ..., a
  - bloky uvnitř příkazů **if**, **for**, ... jsou pouze 1-2 řádky dlouhé.
- Krátké funkce **umožňují testovat dílčí části** algoritmu!
- Sekce uvnitř funkcí/metod:
  - Jasná indikace toho, že funkce/metoda nedělá jen 1 věc a měla by být rozdělena.
- Parametry funkcí/metod:
  - Udržujte jejich počet malý! 0, 1, 2, výjimečně 3.
  - Vytvořte jméno tak, aby evokovalo pořadí argumentů.
  - Boolovské argumenty funkcí často značí, že funkce nedělá 1 věc! Rozdělte ji.
- Při volání používejte častěji **keyword arguments**! Porovnejte:
  - `st("obama", 20, False, True)`
  - `search_twitter("obama", 20, False, True)`
  - `search_twitter("obama", numtweets=20, retweets=False, unicode=True)`



# Eratostenovo síto: funkce

```
1 PRIME = True
2 NONPRIME = False
3
4 def generate_primes_up_to(max_value):
5     """Find primes up to the max_value
6     using the Sieve of Eratosthenes.
7     """
8     if max_value >= 2: # There are some primes
9         # Initialize the list (incl. 0)
10        candidates = [
11            PRIME for i in range(max_value+1)]
12        # Get rid of the known non-primes
13        candidates[0] = candidates[1] = NONPRIME
14        # Run the sieve
15        for number in range(2, len(candidates)):
16            if candidates[number] == PRIME:
17                # mark its multiples as not prime
18                for multiple in range(
19                    2*number, len(candidates), number):
20                    candidates[multiple] = NONPRIME
21        # Find the primes and put them in a list
22        primes = [i for i in range(len(candidates))
23                  if candidates[i] == PRIME]
24        return primes
25    else: # max_value < 2
26        # no primes, return empty list
27        return list()
```

# Eratostenovo síto: funkce

```
1 PRIME = True
2 NONPRIME = False
3
4 def generate_primes_up_to(max_value):
5     """Find primes up to the max_value
6     using the Sieve of Eratosthenes.
7     """
8     if max_value >= 2: # There are some primes
9         # Initialize the list (incl. 0)
10         candidates = [
11             PRIME for i in range(max_value+1)]
12         # Get rid of the known non-primes
13         candidates[0] = candidates[1] = NONPRIME
14         # Run the sieve
15         for number in range(2, len(candidates)):
16             if candidates[number] == PRIME:
17                 # mark its multiples as not prime
18                 for multiple in range(
19                     2*number, len(candidates), number):
20                     candidates[multiple] = NONPRIME
21         # Find the primes and put them in a list
22         primes = [i for i in range(len(candidates))
23                   if candidates[i] == PRIME]
24         return primes
25     else: # max_value < 2
26         # no primes, return empty list
27         return list()
```

```
1 PRIME = True
2 NONPRIME = False
3
4 def generate_primes_up_to(max_value):
5     """Find primes up to the max_value
6     using the Sieve of Eratosthenes.
7     """
8     if max_value < 2:
9         return []
10    else:
11        candidates = init_integers_up_to(max_value)
12        mark_non_primes(candidates)
13        return collect_remaining(candidates)
14
15 def init_integers_up_to(max_value):
16     return [PRIME for i in range(max_value+1)]
17
18 def mark_non_primes(candidates):
19     # Mark 0 and 1, they are not primes.
20     candidates[0] = candidates[1] = NONPRIME
21     for number in range(2, len(candidates)):
22         if candidates[number] == PRIME:
23             mark_multiples_of(number, candidates)
24
25 def mark_multiples_of(number, candidates):
26     for multiple in range(
27         2*number, len(candidates), number):
28         candidates[multiple] = NONPRIME
29
30 def collect_remaining(candidates):
31     primes = [i for i in range(len(candidates))
32               if candidates[i] == PRIME]
33     return primes
```

# Eratostenovo síto: pomocná třída?

```
1 PRIME = True
2 NONPRIME = False
3
4 def generate_primes_up_to(max_value):
5     """Find primes up to the max_value
6     using the Sieve of Eratosthenes.
7     """
8     if max_value < 2:
9         return []
10    else:
11        candidates = init_integers_up_to(max_value)
12        mark_non_primes(candidates)
13        return collect_remaining(candidates)
14
15 def init_integers_up_to(max_value):
16     return [PRIME for i in range(max_value+1)]
17
18 def mark_non_primes(candidates):
19     # Mark 0 and 1, they are not primes.
20     candidates[0] = candidates[1] = NONPRIME
21     for number in range(2, len(candidates)):
22         if candidates[number] == PRIME:
23             mark_multiples_of(number, candidates)
24
25 def mark_multiples_of(number, candidates):
26     for multiple in range(
27         2*number, len(candidates), number):
28         candidates[multiple] = NONPRIME
29
30 def collect_remaining(candidates):
31     primes = [i for i in range(len(candidates))
32               if candidates[i] == PRIME]
33     return primes
```

# Eratostenovo síto: pomocná třída?

```
1 PRIME = True
2 NONPRIME = False
3
4 def generate_primes_up_to(max_value):
5     """Find primes up to the max_value
6     using the Sieve of Eratosthenes.
7     """
8     if max_value < 2:
9         return []
10    else:
11        candidates = init_integers_up_to(max_value)
12        mark_non_primes(candidates)
13        return collect_remaining(candidates)
14
15 def init_integers_up_to(max_value):
16     return [PRIME for i in range(max_value+1)]
17
18 def mark_non_primes(candidates):
19     # Mark 0 and 1, they are not primes.
20     candidates[0] = candidates[1] = NONPRIME
21     for number in range(2, len(candidates)):
22         if candidates[number] == PRIME:
23             mark_multiples_of(number, candidates)
24
25 def mark_multiples_of(number, candidates):
26     for multiple in range(
27         2*number, len(candidates), number):
28         candidates[multiple] = NONPRIME
29
30 def collect_remaining(candidates):
31     primes = [i for i in range(len(candidates))
32               if candidates[i] == PRIME]
33     return primes
```

```
1 PRIME = True
2 NONPRIME = False
3
4 def generate_primes_up_to(max_value):
5     """Return a list of primes up to the max_value."""
6     if max_value < 2: return []
7     candidates = CandidateNumbers(max_value)
8     candidates.checkout_multiples()
9     return candidates.collect_remaining()
10
11 class CandidateNumbers:
12     """List of boolean values for use in the Sieve of Erato
13     Shall be used with the generate_primes_up_to function.
14     """
15     def __init__(self, max_value):
16         self.max = max_value + 1
17         self.candidates = [PRIME for i in range(self.max)]
18         self.candidates[0] = self.candidates[1] = NONPRIME
19
20     def checkout_multiples(self):
21         """Mark multiples of all prime numbers as not prime
22         for number in range(2, int(self.max**0.5)+1):
23             if self.candidates[number] == PRIME:
24                 self.checkout_multiples_of(number)
25
26     def checkout_multiples_of(self, number):
27         """Mark multiples of number as not prime."""
28         for multiple in range(2*number, self.max, number):
29             self.candidates[multiple] = NONPRIME
30
31     def collect_remaining(self):
32         """Return a list of remaining candidates, they are
33         return [i for i in range(self.max)
34               if self.candidates[i] == PRIME]
```



# Příklad

Podle přednášky

**Raymond Hettinger: *Beyond PEP 8 — Best Practices for Beautiful Intelligible Code*. PyCon 2015**



# Původní kód

Formátování kódu

Clean Code

Příklad

- Původní kód
- Co kód dělá?
- Lepší kód
- Modul nettools
- Závěr

```
1 import jnettool.tools.elements.NetworkElement, \  
2       jnettool.tools.Routing, \  
3       jnettool.tools.RouteInspector  
4  
5 ne=jnettool.tools.elements.NetworkElement( '171.0.2.45' )  
6 try:  
7     routing_table=ne.getRoutingTable()  # fetch table  
8  
9 except jnettool.tools.elements.MissingVar:  
10     # Record table fault  
11     logging.exception( '''No routing table found''' )  
12     # Undo partial changes  
13     ne.cleanup( '''rollback''' )  
14  
15 else:  
16     num_routes=routing_table.getSize()  # determine table size  
17     for RToffset in range( num_routes ):  
18         route=routing_table.getRouteByIndex( RToffset )  
19         name=route.getName()  # route name  
20         ipaddr=route.getIPAddr()  # ip address  
21         print('{:15s} -> {:s}'.format(name, ipaddr))  # format nicely  
22 finally:  
23     ne.cleanup( '''commit''' )  # lockin changes  
24     ne.disconnect()
```



# Kód podle PEP 8

Formátování kódu

Clean Code

Příklad

- Původní kód
- Co kód dělá?
- Lepší kód
- Modul nettools
- Závěr

```
1 import jnettool.tools.elements.NetworkElement
2 import jnettool.tools.Routing
3 import jnettool.tools.RouteInspector
4
5
6 ne = jnettool.tools.elements.NetworkElement('171.0.2.45')
7
8 try:
9     routing_table = ne.getRoutingTable()
10 except jnettool.tools.elements.MissingVar:
11     logging.exception('No routing table found')
12     ne.cleanup('rollback')
13 else:
14     num_routes = routing_table.getSize()
15     for RToffset in range(num_routes):
16         route = routing_table.getRouteByIndex(RToffset)
17         name = route.getName()
18         ipaddr = route.getIPAddr()
19         print('{:15s} -> {}'.format(name, ipaddr))
20 finally:
21     ne.cleanup('commit')
22     ne.disconnect()
```

# Co kód dělá?

---

```
1 import jnettool.tools.elements.NetworkElement
2 import jnettool.tools.Routing
3 import jnettool.tools.RouteInspector
4
5
6 ne = jnettool.tools.elements.NetworkElement('171.0.2.45')
7
8 try:
9     routing_table = ne.getRoutingTable()
10 except jnettool.tools.elements.MissingVar:
11     logging.exception('No routing table found')
12     ne.cleanup('rollback')
13 else:
14     num_routes = routing_table.getSize()
15     for RTOffset in range(num_routes):
16         route = routing_table.getRouteByIndex(RTOffset)
17         name = route.getName()
18         ipaddr = route.getIPAddr()
19         print('{:15s} -> {:s}'.format(name, ipaddr))
20 finally:
21     ne.cleanup('commit')
22     ne.disconnect()
```



# Co kód dělá?

---

```
1 import jnettool.tools.elements.NetworkElement
2 import jnettool.tools.Routing
3 import jnettool.tools.RouteInspector
4
5
6 ne = jnettool.tools.elements.NetworkElement('171.0.2.45')
7
8 try:
9     routing_table = ne.getRoutingTable()
10 except jnettool.tools.elements.MissingVar:
11     logging.exception('No routing table found')
12     ne.cleanup('rollback')
13 else:
14     num_routes = routing_table.getSize()
15     for RToffset in range(num_routes):
16         route = routing_table.getRouteByIndex(RToffset)
17         name = route.getName()
18         ipaddr = route.getIPAddr()
19         print('{:15s} -> {:s}'.format(name, ipaddr))
20 finally:
21     ne.cleanup('commit')
22     ne.disconnect()
```

Je toto čistý kód?

# Co kód dělá?

Co ten kód vlastně dělá?

```
1 import jnettool.tools.elements.NetworkElement
2 import jnettool.tools.Routing
3 import jnettool.tools.RouteInspector
4
5
6 ne = jnettool.tools.elements.NetworkElement('171.0.2.45')
7
8 try:
9     routing_table = ne.getRoutingTable()
10 except jnettool.tools.elements.MissingVar:
11     logging.exception('No routing table found')
12     ne.cleanup('rollback')
13 else:
14     num_routes = routing_table.getSize()
15     for RTOffset in range(num_routes):
16         route = routing_table.getRouteByIndex(RTOffset)
17         name = route.getName()
18         ipaddr = route.getIPAddr()
19         print('{:15s} -> {:s}'.format(name, ipaddr))
20 finally:
21     ne.cleanup('commit')
22     ne.disconnect()
```

Je toto čistý kód?

# Co kód dělá?

```
1 import jnettool.tools.elements.NetworkElement
2 import jnettool.tools.Routing
3 import jnettool.tools.RouteInspector
4
5
6 ne = jnettool.tools.elements.NetworkElement('171.0.2.45')
7
8 try:
9     routing_table = ne.getRoutingTable()
10 except jnettool.tools.elements.MissingVar:
11     logging.exception('No routing table found')
12     ne.cleanup('rollback')
13 else:
14     num_routes = routing_table.getSize()
15     for RTOffset in range(num_routes):
16         route = routing_table.getRouteByIndex(RTOffset)
17         name = route.getName()
18         ipaddr = route.getIPAddr()
19         print('{:15s} -> {:s}'.format(name, ipaddr))
20 finally:
21     ne.cleanup('commit')
22     ne.disconnect()
```

Je toto čistý kód?

Co ten kód vlastně dělá?

1. Importuje potřebné věci z knihovny pro práci se síťovými prvky.
2. Připojuje se k síťovému zařízení (routeru).
3. Pokouší se stáhnout směrovací tabulku.
4. Pokud tabulka chybí, zaznamená chybu a uklidí po sobě a skončí. (Pokud byste neuklidili, mohlo byto router zanechat ve stavu, kdy se k němu nelze připojit.)
5. Zjistí velikost tabulky.
6. Následně iteruje přes indexy ve směrovací tabulce.
7. Najde směrovací pravidlo podle indexu.
8. Zjistí název pravidla, IP adresu a vytiskne je.
9. Nakonec uloží všechny změny a odpojí se od směrovače.

# Co kód dělá?

```
1 import jnettool.tools.elements.NetworkElement
2 import jnettool.tools.Routing
3 import jnettool.tools.RouteInspector
4
5
6 ne = jnettool.tools.elements.NetworkElement('171.0.2.45')
7
8 try:
9     routing_table = ne.getRoutingTable()
10 except jnettool.tools.elements.MissingVar:
11     logging.exception('No routing table found')
12     ne.cleanup('rollback')
13 else:
14     num_routes = routing_table.getSize()
15     for RTOffset in range(num_routes):
16         route = routing_table.getRouteByIndex(RTOffset)
17         name = route.getName()
18         ipaddr = route.getIPAddr()
19         print('{:15s} -> {:s}'.format(name, ipaddr))
20 finally:
21     ne.cleanup('commit')
22     ne.disconnect()
```

Je toto čistý kód?

Co ten kód vlastně dělá?

1. Importuje potřebné věci z knihovny pro práci se síťovými prvky.
2. Připojuje se k síťovému zařízení (routeru).
3. Pokouší se stáhnout směrovací tabulku.
4. Pokud tabulka chybí, zaznamená chybu a uklidí po sobě a skončí. (Pokud byste neuklidili, mohlo byto router zanechat ve stavu, kdy se k němu nelze připojit.)
5. Zjistí velikost tabulky.
6. Následně iteruje přes indexy ve směrovací tabulce.
7. Najde směrovací pravidlo podle indexu.
8. Zjistí název pravidla, IP adresu a vytiskne je.
9. Nakonec uloží všechny změny a odpojí se od směrovače.

Čeho chce programátor tímto kódem dosáhnout?

# Co kód dělá?

```
1 import jnettool.tools.elements.NetworkElement
2 import jnettool.tools.Routing
3 import jnettool.tools.RouteInspector
4
5
6 ne = jnettool.tools.elements.NetworkElement('171.0.2.45')
7
8 try:
9     routing_table = ne.getRoutingTable()
10 except jnettool.tools.elements.MissingVar:
11     logging.exception('No routing table found')
12     ne.cleanup('rollback')
13 else:
14     num_routes = routing_table.getSize()
15     for RTOffset in range(num_routes):
16         route = routing_table.getRouteByIndex(RTOffset)
17         name = route.getName()
18         ipaddr = route.getIPAddr()
19         print('{:15s} -> {:s}'.format(name, ipaddr))
20 finally:
21     ne.cleanup('commit')
22     ne.disconnect()
```

Je toto čistý kód?

Co ten kód vlastně dělá?

1. Importuje potřebné věci z knihovny pro práci se síťovými prvky.
2. Připojuje se k síťovému zařízení (routeru).
3. Pokouší se stáhnout směrovací tabulku.
4. Pokud tabulka chybí, zaznamená chybu a uklidí po sobě a skončí. (Pokud byste neuklidili, mohlo byto router zanechat ve stavu, kdy se k němu nelze připojit.)
5. Zjistí velikost tabulky.
6. Následně iteruje přes indexy ve směrovací tabulce.
7. Najde směrovací pravidlo podle indexu.
8. Zjistí název pravidla, IP adresu a vytiskne je.
9. Nakonec uloží všechny změny a odpojí se od směrovače.

Čeho chce programátor tímto kódem dosáhnout?

Je to čistý kód?

# Co kód dělá?

```
1 import jnettool.tools.elements.NetworkElement
2 import jnettool.tools.Routing
3 import jnettool.tools.RouteInspector
4
5
6 ne = jnettool.tools.elements.NetworkElement('171.0.2.45')
7
8 try:
9     routing_table = ne.getRoutingTable()
10 except jnettool.tools.elements.MissingVar:
11     logging.exception('No routing table found')
12     ne.cleanup('rollback')
13 else:
14     num_routes = routing_table.getSize()
15     for RTOffset in range(num_routes):
16         route = routing_table.getRouteByIndex(RTOffset)
17         name = route.getName()
18         ipaddr = route.getIPAddr()
19         print('{:15s} -> {:s}'.format(name, ipaddr))
20 finally:
21     ne.cleanup('commit')
22     ne.disconnect()
```

Je toto čistý kód?

Co ten kód vlastně dělá?

1. Importuje potřebné věci z knihovny pro práci se síťovými prvky.
2. Připojuje se k síťovému zařízení (routeru).
3. Pokouší se stáhnout směrovací tabulku.
4. Pokud tabulka chybí, zaznamená chybu a uklidí po sobě a skončí. (Pokud byste neuklidili, mohlo byto router zanechat ve stavu, kdy se k němu nelze připojit.)
5. Zjistí velikost tabulky.
6. Následně iteruje přes indexy ve směrovací tabulce.
7. Najde směrovací pravidlo podle indexu.
8. Zjistí název pravidla, IP adresu a vytiskne je.
9. Nakonec uloží všechny změny a odpojí se od směrovače.

Čeho chce programátor tímto kódem dosáhnout?

Je to čistý kód?

- Nehezky kombinuje ošetření chyb s vlastní logikou.
- Není zřejmé, co kód vlastně dělá.



# Lepší kód

---

Jak to udělat lépe?

Formátování kódu

Clean Code

Příklad

- Původní kód
- Co kód dělá?
- Lepší kód
- Modul nettools
- Závěr



# Lepší kód

---

Jak to udělat lépe?

Formátování kódu

Clean Code

Příklad

- Původní kód
- Co kód dělá?
- Lepší kód
- Modul nettools
- Závěr

```
1 from nettools import NetworkElement
2
3 with NetworkElement('171.0.2.45') as ne:
4     for route in ne.routing_table:
5         print('{:15s} -> {}'.format(route.name, route.ipaddr))
```





# Lepší kód

Jak to udělat lépe?

Formátování kódu

Clean Code

Příklad

- Původní kód
- Co kód dělá?
- Lepší kód
- Modul nettools
- Závěr

```
1 from nettools import NetworkElement
2
3 with NetworkElement('171.0.2.45') as ne:
4     for route in ne.routing_table:
5         print('{:15s} -> {}'.format(route.name, route.ipaddr))
```

- Krátký, přehledný kód.
- Jasně sděluje, co je jeho hlavním účelem.
- Má jedinou vadu. Nefunguje. (Zatím.)



# Lepší kód

Jak to udělat lépe?

Formátování kódu

Clean Code

Příklad

- Původní kód
- Co kód dělá?
- Lepší kód
- Modul nettools
- Závěr

```
1 from nettools import NetworkElement
2
3 with NetworkElement('171.0.2.45') as ne:
4     for route in ne.routing_table:
5         print('{:15s} -> {:s}'.format(route.name, route.ipaddr))
```

- Krátký, přehledný kód.
- Jasně sděluje, co je jeho hlavním účelem.
- Má jedinou vadu. Nefunguje. (Zatím.)

Aby kód fungoval,

- musíme vytvořit nový modul nettools, který bude “překládat” náš krásný kód na volání původní knihovny jnettools.
- To není zcela triviální a my to ještě neumíme.
- Nicméně Python to umožňuje a podporuje.
- Modul nettools schovává detaily ošetření chyb, atd. Váš hlavní kód se může soustředit na logiku aplikace.
- Je to nezanedbatelná práce navíc. Ale uděláte ji jedinkrát, a v budoucnu z toho budete profitovat vy i vaši spolupracovníci.

# Modul nettools

Pro pokročilejší programátory, které zajímá, jak se takový modul dá udělat:

```
1 import jnettool.tools.elements.NetworkElement
2 import jnettool.tools.Routing
3
4 class NetworkElementError(Exception):
5     pass
6
7 class NetworkElement:
8
9     def __init__(self, ipaddr):
10         self.ipaddr = ipaddr
11         self.oldne = jnettool.tools.elements.NetworkElement(ipaddr)
12
13     @property
14     def routing_table(self):
15         try:
16             return RoutingTable(self.oldne.getRoutingTable())
17         except jnettool.tools.elements.MissingVar:
18             raise NetworkElementError('No routing table found')
19
20     def __enter__(self):
21         return self
22
23     def __exit__(self, exctype, excinst, exctb):
24         if exctype == NetworkElementError:
25             logging.exception('No routing table found')
26             self.oldne.cleanup('rollback')
27         else:
28             self.oldne.cleanup('commit')
29         self.oldne.disconnect()
30
31     def __repr__(self):
32         return '{:s}({:s})'.format(self.__class__.name, self.ipaddr)
33
34 class RoutingTable:
35
36     def __init__(self, oldrt):
37         self.oldrt = oldrt
38
39     def __len__(self):
40         return self.oldrt.getSize()
41
42     def __getitem__(self, index):
43         if index >= len(self):
44             raise IndexError
45         return Route(
46             self.oldrt.getRouteByIndex(index))
47
48 class Route:
49
50     def __init__(self, old_route):
51         self.old_route = old_route
52
53     @property
54     def name(self):
55         return self.old_route.getName()
56
57     @property
58     def ipaddr(self):
59         return self.old_route.getIPAddr()
60
```



# Závěr

---

Formátování kódu

---

Clean Code

---

Příklad

---

- Původní kód
- Co kód dělá?
- Lepší kód
- Modul nettools
- Závěr

- Čistý kód je subjektivní pojem, přesto by se o něj měl každý programátor snažit.
- Čistý kód by měl být především čitelný (skoro jako v přirozeném jazyce).
- 80 % čistého kódu jsou správně zvolená jména!
- Vhodná jména lze volit, jsou-li funkce/metody dostatečně krátké!
- Opakují-li se ve vašem programu stejné nebo podobné kusy kódu, prakticky vždy je možné takový kód definovat jako samostatnou funkci/metodu.