CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering
Department of Cybernetics

B4B33RPH: Řešení problémů a hry

3. úloha: Spam filtr

Petr Pošík a Tomáš Svoboda

CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering
Department of Cybernetics

B4B33RPH: Řešení problémů a hry

3. úloha: Spam filtr

Petr Pošík a Tomáš Svoboda

SPAM?

SPAM?

Hormel Foods Corporation, 1937

SPAM?

Hormel Foods Corporation, 1937

70. léta:

- **Monty Python's Flying Circus** (youtube):
  - Manželé si chtějí dát v bistru snídani.
  - Manželka nechce jídlo obsahující SPAM.
  - Obsluha i manžel jí přesto stále SPAM vnucují.
  - "I DON'T LIKE SPAM!!!"
- **SPAM** - nezajímavé, nechtěné, vnucované, opakující se zprávy
- **HAM** - korektní zprávy týkající se tématu

SPAM?

Hormel Foods Corporation, 1937

70. léta:

- **Monty Python's Flying Circus** (youtube):
  - Manželé si chtějí dát v bistru snídani.
  - Manželka nechce jídlo obsahující SPAM.
  - Obsluha i manžel jí přesto stále SPAM vnucují.
  - "I DON'T LIKE SPAM!!!"
- **SPAM** - nezajímavé, nechtěné, vnucované, opakující se zprávy
- **HAM** - korektní zprávy týkající se tématu

Definice?

Spam je

- *nevyžádaná* zpráva
- *hromadně rozesílaná* mnoha lidem *bez rozdílu*,
- velmi často *za komerčním účelem*.

Definice?

Spam je

- *nevyžádaná* zpráva
- *hromadně rozesílaná* mnoha lidem *bez rozdílu*,
- velmi často *za komerčním účelem*.

Zkuste si odpovědět:

1. Označili by všichni lidé za spam ty samé emaily?
2. Kdyby stejnému člověku přišla stejná sada emailů v současnosti a za pět let, označil by za spamy stejné emaily?

A 1. Ne. | 2. Ne.

B 1. Ne. | 2. Ano.

C 1. Ano. | 2. Ne.

D 1. Ano. | 2. Ano.

Definice?

Spam je

- *nevyžádaná* zpráva
- *hromadně rozesílaná* mnoha lidem *bez rozdílu*,
- velmi často *za komerčním účelem*.

Zkuste si odpovědět:

1. Označili by všichni lidé za spam ty samé emaily?
2. Kdyby stejnému člověku přišla stejná sada emailů v současnosti a za pět let, označil by za spamy stejné emaily?

A 1. Ne. | 2. Ne.

B 1. Ne. | 2. Ano.

C 1. Ano. | 2. Ne.

D 1. Ano. | 2. Ano.

Spam filtr

Spam filtr:

- Automat, který každému emailu umí přiřadit štítek SPAM nebo HAM (přičemž se může plést).

Spam filtr

Spam filtr:

- Automat, který každému emailu umí přiřadit štítek SPAM nebo HAM (přičemž se může plést).

Na základě čeho se spam filtr rozhoduje?

A Podle hlavičky emailu (odesílatel, adresát, datum odeslání, ...)

B Podle těla emailu (text zprávy).

C Podle počtu, typu a obsahu příloh.

D Podle formátu textu (barvy, tučné nebo blikající písmo, ...)

Spam filtr

Spam filtr:

- Automat, který každému emailu umí přiřadit štítek SPAM nebo HAM (přičemž se může plést).

Na základě čeho se spam filtr rozhoduje?

A Podle hlavičky emailu (odesílatel, adresát, datum odeslání, ...)

B Podle těla emailu (text zprávy).

C Podle počtu, typu a obsahu příloh.

D Podle formátu textu (barvy, tučné nebo blikající písmo, ...)

Spam filtr

Spam filtr:

- Automat, který každému emailu umí přiřadit štítek SPAM nebo HAM (přičemž se může plést).

Jaké má vstupy a jaké má výstupy?

- Vstup: všechny informace o emailu, které můžeme získat (tělo emailu, přílohy, hlavičky). Velmi často nestrukturovaný text.
- Výstup: štítek SPAM nebo HAM

Spam filtr

Spam filtr:

- Automat, který každému emailu umí přiřadit štítek SPAM nebo HAM (přičemž se může plést).

Jaké má vstupy a jaké má výstupy?

- Vstup: všechny informace o emailu, které můžeme získat (tělo emailu, přílohy, hlavičky). Velmi často nestrukturovaný text.
- Výstup: štítek SPAM nebo HAM

Spam filtr

Spam filtr:

- Automat, který každému emailu umí přiřadit štítek SPAM nebo HAM (přičemž se může plést).

Jaké má vstupy a jaké má výstupy?

- Vstup: všechny informace o emailu, které můžeme získat (tělo emailu, přílohy, hlavičky). Velmi často nestrukturovaný text.
- Výstup: štítek SPAM nebo HAM

Zkuste si odpovědět:

1. Bude jeden a tentýž spam filter vyhovovat všem lidem?
2. Co musí spam filtr umět, aby si udržel svou efektivitu v průběhu času?
3. Co musí umět, aby byl efektivní pro různé lidi?
4. Může se filtr dopustit chyb? Jakých?
5. Budou všechny typy chyb stejně vážné?
6. Jak poznám, že je jeden filtr lepší než druhý?

Úloha

Proč byl spam filtr vybrán jako úloha v předmětu RPH?

- Praktická netriviální úloha, přesto řešitelná s našimi znalostmi Pythonu.
- Prohloubení schopnosti zpracovat text, pracovat se soubory a datovými strukturami.
- Ukázka učení z dat (strojového učení).

Na práci na spam filtru máme cca 4 týdny.

- 2 týdny: **individuální** práce na "zázemí", které nám umožní filtry
  - jednoduše psát,
  - snadno testovat a porovnávat a
  - vyjasnit si, jak že vlastně ten spam filtr bude vypadat.
- 2 týdny: **týmová** práce na vlastním filtru.
  - Vyzkoušíte si vámi zvolenou metodu filtrování spamu.

Dostanete testy k jednotlivým dílčím podúlohám.

Odevzdávat budete

- implementaci funkce pro hodnocení kvality filtru (každý sám),
- implementaci vašeho spam filtru (ve dvojicích) a
- report + prezentaci (ve dvojicích, bude doupřesněno).

Bodování: nápad, čistota kódu, účinnost na skutečných datech

Úloha

Proč byl spam filtr vybrán jako úloha v předmětu RPH?

- Praktická netriviální úloha, přesto řešitelná s našimi znalostmi Pythonu.
- Prohloubení schopnosti zpracovat text, pracovat se soubory a datovými strukturami.
- Ukázka učení z dat (strojového učení).

Na práci na spam filtru máme cca 4 týdny.

- 2 týdny: **individuální** práce na "zázemí", které nám umožní filtry
  - jednoduše psát,
  - snadno testovat a porovnávat a
  - vyjasnit si, jak že vlastně ten spam filtr bude vypadat.
- 2 týdny: **týmová** práce na vlastním filtru.
  - Vyzkoušíte si vámi zvolenou metodu filtrování spamu.

Dostanete testy k jednotlivým dílčím podúlohám.

Odevzdávat budete

- implementaci funkce pro hodnocení kvality filtru (každý sám),
- implementaci vašeho spam filtru (ve dvojicích) a
- report + prezentaci (ve dvojicích, bude doupřesněno).

Bodování: nápad, čistota kódu, účinnost na skutečných datech

CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering
Department of Cybernetics

B4B33RPH: Řešení problémů a hry
PEP 8. Čistý kód.

Petr Pošík

Katedra kybernetiky
ČVUT FEL

Formátování kódu

Proč je důležité formátování?

- 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.

Proč je důležité formátování?

- 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.

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 `pycodestyle`:

- `https://pypi.python.org/pypi/pycodestyle`
- Automatická kontrola formátování kódu podle doporučení PEP 8.

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.

Modul `pycodestyle`:

- `https://pypi.python.org/pypi/pycodestyle`
- Automatická kontrola formátování kódu podle doporučení PEP 8.

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 -*-`
- `import`y umístěte na začátek souboru, každý modul na zvláštní řádek
- Definice funkcí udržujte *pohromadě*.
- Definice tříd a funkcí nejvyšší úrovně *oddělujte* 2 prázdnými řádky. Definice vnořených funkcí a metod oddělujte 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”???

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 -*-`
- `import`y umístěte na začátek souboru, každý modul na zvláštní řádek
- Definice funkcí udržujte *pohromadě*.
- Definice tříd a funkcí nejvyšší úrovně *oddělujte* 2 prázdnými řádky. Definice vnořených funkcí a metod oddělujte 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".

Page

A to stačí, aby byl můj kód "čistý"???

Clean Code

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

Clean Code

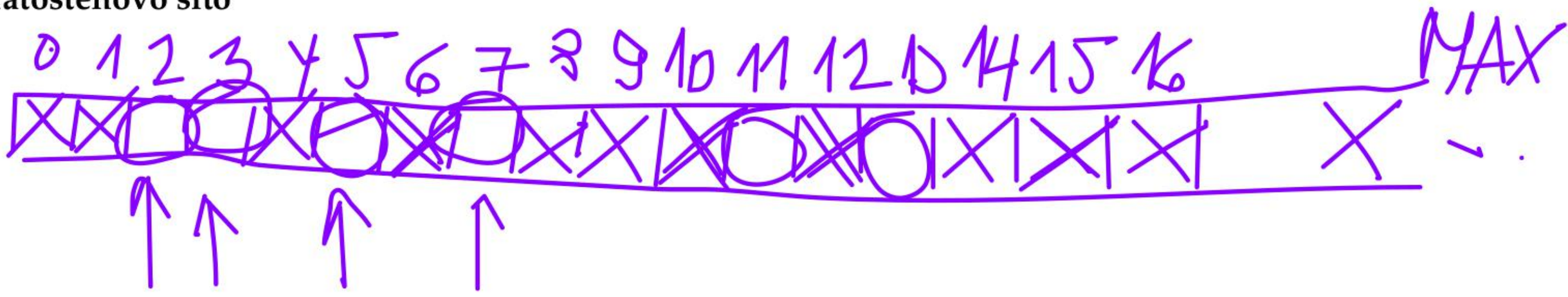
Zpracováno podle
Robert C. Martin: *Clean Code: A Handbook of Agile Software Craftsmanship*,
Prentice Hall, 2008.

Který kód je čistší? A proč?

Eratostenovo síto

Který kód je čistší? A proč?

Eratostenovo síto



Který kód je čistší? A proč?

Eratostenovo síto

Dvě implementace téhož algoritmu:

```python
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value >= 2:  # There are some primes
        # Initialize the list (incl. 0)
        f = [True for i in range(max_value+1)]
        # Get rid of the known non-primes
        f[0] = f[1] = False
        # Run the sieve
        for i in range(2, len(f)):
            if f[i]:   # i is still a candidate
                # mark its multiples as not prime
                for j in range(2*i, len(f), i):
                    f[j] = False
        # Find the primes and put them in a list
        primes = [i for i in range(len(f)) if f[i]]
        return primes
    else:   # max_value < 2
        # no primes, return empty list
        return list()
```

```python
PRIME = True
NONPRIME = False

def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value < 2:
        return []
    else:
        candidates = init_integers_up_to(max_value)
        mark_non_primes(candidates)
        return collect_remaining(candidates)
```

Který kód je čistší? A proč?

Eratostenovo síto

Dvě implementace téhož algoritmu:

```python
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value >= 2:  # There are some primes
        # Initialize the list (incl. 0)
        f = [True for i in range(max_value+1)]
        # Get rid of the known non-primes
        f[0] = f[1] = False
        # Run the sieve
        for i in range(2, len(f)):
            if f[i]:   # i is still a candidate
                # mark its multiples as not prime
                for j in range(2*i, len(f), i):
                    f[j] = False
        # Find the primes and put them in a list
        primes = [i for i in range(len(f)) if f[i]]
        return primes
    else:   # max_value < 2
        # no primes, return empty list
        return list()
```

```python
PRIME = True
NONPRIME = False

def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value < 2:
        return []
    else:
        candidates = init_integers_up_to(max_value)
        mark_non_primes(candidates)
        return collect_remaining(candidates)
```

Který kód je čistší? A proč?

Eratostenovo síto

Dvě implementace téhož algoritmu:

```python
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value >= 2:  # There are some primes
        # Initialize the list (incl. 0)
        f = [True for i in range(max_value+1)]
        # Get rid of the known non-primes
        f[0] = f[1] = False
        # Run the sieve
        for i in range(2, len(f)):
            if f[i]:   # i is still a candidate
                # mark its multiples as not prime
                for j in range(2*i, len(f), i):
                    f[j] = False
        # Find the primes and put them in a list
        primes = [i for i in range(len(f)) if f[i]]
        return primes
    else:   # max_value < 2
        # no primes, return empty list
        return list()
```

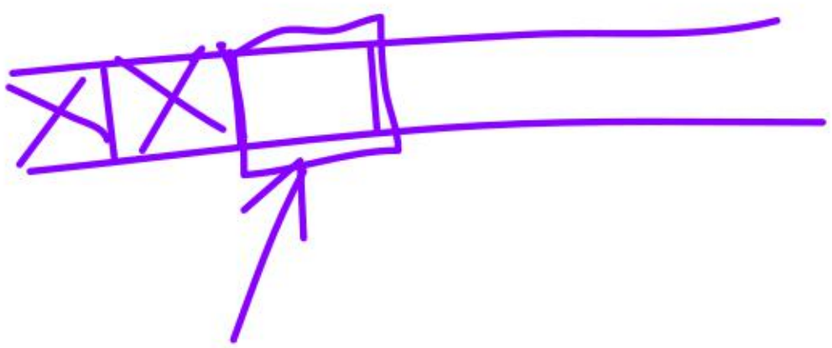
```python
PRIME = True
NONPRIME = False

def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value < 2:
        return []
    else:
        candidates = init_integers_up_to(max_value)
        mark_non_primes(candidates)
        return collect_remaining(candidates)

def init_integers_up_to(max_value):
    return [PRIME for i in range(max_value+1)]

def mark_non_primes(candidates):
    # Mark 0 and 1, they are not primes.
    candidates[0] = candidates[1] = NONPRIME
    for number in range(2, len(candidates)):
        if candidates[number] == PRIME:
            mark_multiples_of(number, candidates)

def mark_multiples_of(number, candidates):
    for multiple in range(
            2*number, len(candidates), number):
        candidates[multiple] = NONPRIME

def collect_remaining(candidates):
    primes = [i for i in range(len(candidates))
              if candidates[i] == PRIME]
    return primes
```

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**.

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**.

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:

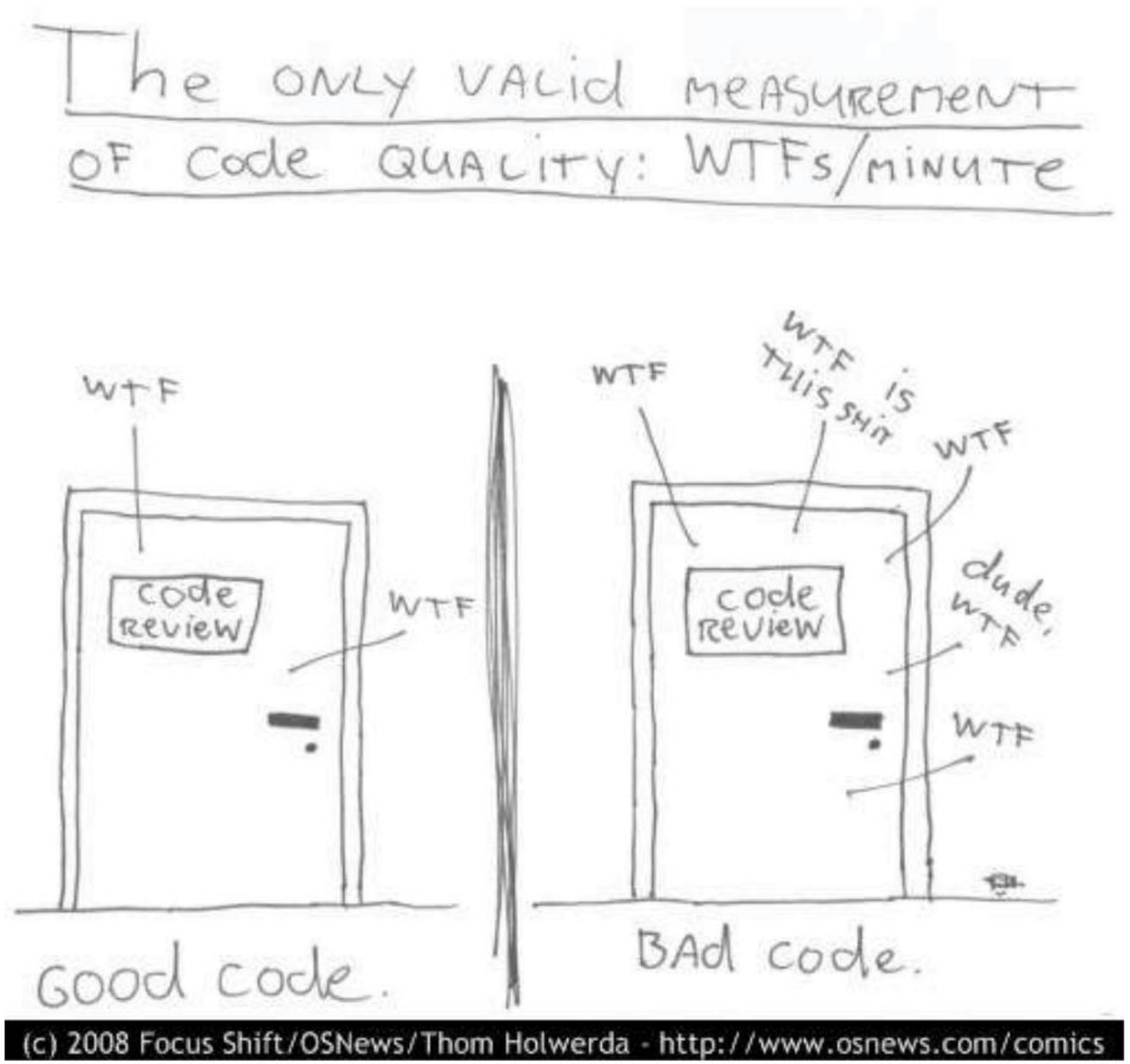
> **Clean code can be read, and enhanced by a developer other than its original author**. It has **unit and acceptance tests**. It has **meaningful names**. It provides one way rather than many ways for doing one thing. It has **minimal dependencies**, which are explicitly defined, and **provides a clear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

Čistý kód v praxi

Code review:

Čistý kód v praxi

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



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`

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`

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)`

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)`

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)`

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)`

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)`
- 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:`

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)`
- 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

```python
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value >= 2:  # There are some primes
        # Initialize the list (incl. 0)
        f = [True for i in range(max_value+1)]
        # Get rid of the known non-primes
        f[0] = f[1] = False
        # Run the sieve
        for i in range(2, len(f)):
            if f[i]:   # i is still a candidate
                # mark its multiples as not prime
                for j in range(2*i, len(f), i):
                    f[j] = False
        # Find the primes and put them in a list
        primes = [i for i in range(len(f)) if f[i]]
        return primes
    else:   # max_value < 2
        # no primes, return empty list
        return list()
```

```python
PRIME = True
NONPRIME = False

def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value >= 2:  # There are some primes
        # Initialize the list (incl. 0)
        candidates = [
            PRIME for i in range(max_value+1)]
        # Get rid of the known non-primes
        candidates[0] = candidates[1] = NONPRIME
        # Run the sieve
        for number in range(2, len(candidates)):
            if candidates[number] == PRIME:
                # mark its multiples as not prime
                for multiple in range(
                  2*number, len(candidates), number):
                    candidates[multiple] = NONPRIME
        # Find the primes and put them in a list
        primes = [i for i in range(len(candidates))
                  if candidates[i] == PRIME]
        return primes
    else:   # max_value < 2
        # no primes, return empty list
        return list()
```

Další smyslupná jména budou následovat!!!

Eratostenovo síto: smysluplná jména

```python
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value >= 2:  # There are some primes
        # Initialize the list (incl. 0)
        f = [True for i in range(max_value+1)]
        # Get rid of the known non-primes
        f[0] = f[1] = False
        # Run the sieve
        for i in range(2, len(f)):
            if f[i]:   # i is still a candidate
                # mark its multiples as not prime
                for j in range(2*i, len(f), i):
                    f[j] = False
        # Find the primes and put them in a list
        primes = [i for i in range(len(f)) if f[i]]
        return primes
    else:   # max_value < 2
        # no primes, return empty list
        return list()
```

```python
PRIME = True
NONPRIME = False

def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value >= 2:  # There are some primes
        # Initialize the list (incl. 0)
        candidates = [
            PRIME for i in range(max_value+1)]
        # Get rid of the known non-primes
        candidates[0] = candidates[1] = NONPRIME
        # Run the sieve
        for number in range(2, len(candidates)):
            if candidates[number] == PRIME:
                # mark its multiples as not prime
                for multiple in range(
                  2*number, len(candidates), number):
                    candidates[multiple] = NONPRIME
        # Find the primes and put them in a list
        primes = [i for i in range(len(candidates))
                  if candidates[i] == PRIME]
        return primes
    else:   # max_value < 2
        # no primes, return empty list
        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

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

Eratostenovo síto: komentáře

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

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

Eratostenovo síto: komentáře

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

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

Funkce a metody

- 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

- 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

- 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

- 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

- 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)`

Funkce a metody

- 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

```python
PRIME = True
NONPRIME = False

def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value >= 2:  # There are some primes
        # Initialize the list (incl. 0)
        candidates = [
            PRIME for i in range(max_value+1)]
        # Get rid of the known non-primes
        candidates[0] = candidates[1] = NONPRIME
        # Run the sieve
        for number in range(2, len(candidates)):
            if candidates[number] == PRIME:
                # mark its multiples as not prime
                for multiple in range(
                  2*number, len(candidates), number):
                    candidates[multiple] = NONPRIME
        # Find the primes and put them in a list
        primes = [i for i in range(len(candidates))
                  if candidates[i] == PRIME]
        return primes
    else:   # max_value < 2
        # no primes, return empty list
        return list()
```

```python
PRIME = True
NONPRIME = False

def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value < 2:
        return []
    else:
        candidates = init_integers_up_to(max_value)
        mark_non_primes(candidates)
        return collect_remaining(candidates)

def init_integers_up_to(max_value):
    return [PRIME for i in range(max_value+1)]

def mark_non_primes(candidates):
    # Mark 0 and 1, they are not primes.
    candidates[0] = candidates[1] = NONPRIME
    for number in range(2, len(candidates)):
        if candidates[number] == PRIME:
            mark_multiples_of(number, candidates)

def mark_multiples_of(number, candidates):
    for multiple in range(
            2*number, len(candidates), number):
        candidates[multiple] = NONPRIME

def collect_remaining(candidates):
    primes = [i for i in range(len(candidates))
              if candidates[i] == PRIME]
    return primes
```

Eratostenovo síto: funkce

```python
PRIME = True
NONPRIME = False

def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value >= 2:  # There are some primes
        # Initialize the list (incl. 0)
        candidates = [
            PRIME for i in range(max_value+1)]
        # Get rid of the known non-primes
        candidates[0] = candidates[1] = NONPRIME
        # Run the sieve
        for number in range(2, len(candidates)):
            if candidates[number] == PRIME:
                # mark its multiples as not prime
                for multiple in range(
                  2*number, len(candidates), number):
                    candidates[multiple] = NONPRIME
        # Find the primes and put them in a list
        primes = [i for i in range(len(candidates))
                  if candidates[i] == PRIME]
        return primes
    else:   # max_value < 2
        # no primes, return empty list
        return list()
```

```python
PRIME = True
NONPRIME = False

def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.
    """
    if max_value < 2:
        return []
    else:
        candidates = init_integers_up_to(max_value)
        mark_non_primes(candidates)
        return collect_remaining(candidates)

def init_integers_up_to(max_value):
    return [PRIME for i in range(max_value+1)]

def mark_non_primes(candidates):
    # Mark 0 and 1, they are not primes.
    candidates[0] = candidates[1] = NONPRIME
    for number in range(2, len(candidates)):
        if candidates[number] == PRIME:
            mark_multiples_of(number, candidates)

def mark_multiples_of(number, candidates):
    for multiple in range(
            2*number, len(candidates), number):
        candidates[multiple] = NONPRIME

def collect_remaining(candidates):
    primes = [i for i in range(len(candidates))
              if candidates[i] == PRIME]
    return primes
```

Příklad: Refactoring + čistý kód

Rosemary's Grocery Store!!!