Programy a algoritmy pracující s čísly

IB111 Základy programování

Radek Pelánek

2018

Rozcvička

$$1^2 + 2^2 + 3^2 + \cdots + 99^2 + 100^2$$

Dnešní přednáška

- práce s čísly v Pythonu
- ukázky programů, ilustrace použití základních konstrukcí
- upozornění na záludnosti, ladění
- ukázky jednoduchých algoritmů, ilustrace rozdílu v efektivitě

Číselné typy

- `int` – „integer“, celá čísla
- `float`
  - „floating-point number“
  - čísla s plovoucí desetinnou čárkou
  - reprezentace: mantisa $\times$ báze$^{\text{exponent}}$
  - nepřesnosti, zaokrouhlování
- (`complex` – komplexní čísla)

Nepřesnosti

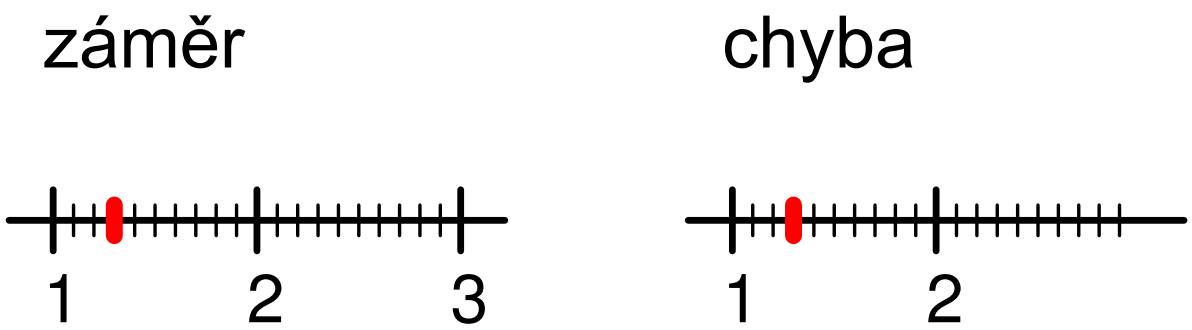
Přesná matematika:

$$((1 + \frac{1}{x}) - 1) \cdot x = 1$$

Nepřesné počítače:

```
>>> x = 2**50
>>> ((1 + 1 / x) - 1) * x
1.0
>>> x = 2**100
>>> ((1 + 1 / x) - 1) * x
0.0
```

Nepřesné výpočty – praktický případ




```
step = 0.1
while value <= bound:
    # draw line ...
    value += step
```

Číselné typy – poznámky

- explicitní přetypování: `int(x)`, `float(x)`
- automatické „nafukování“ typu `int`:
  - viz např. 2**100
  - pomalejší, ale korektní
  - rozdíl od většiny jiných prog. jazyků (běžné je „přetečení“)

v Python2.7 dělení: rozdíl `3/2` a `3/2.0`

v Python3 dělení intuitivní

Kvízové otázky

Co udělá program?

```
n = 1
while n > 0:
    print(n)
    n = n / 10
print("done")
```

Co když použijeme „`n = n * 10`“?
A co „`n = n * 10.0`“?
Co udělají analogické programy v jiných programovacích jazycích?

Pokročilejší operace s čísly

Některé operace v knihovně `math`:

- použití knihovny: `import math`
- zaokrouhlování: `round`, `math.ceil`, `math.floor`
- absolutní hodnota: `abs`
- `math.exp`, `math.log`, `math.sqrt`
- goniometrické funkce: `math.sin`, `math.cos`, `...`
- konstanty: `math.pi`, `math.e`

Záludné detaily

```
>>> round(2.5)
2
>>> round(3.5)
4
>>> round(2.675, 2)
2.67
```

https://docs.python.org/3/library/functions.html#round

The behavior of round() for floats can be surprising: for example, round(2.675, 2) gives 2.67 instead of the expected 2.68. This is not a bug: it's a result of the fact that most decimal fractions can't be represented exactly as a float.

Ciferný součet

- vstup: číslo x
- výstup: ciferný součet čísla x
- příklady:
  - $8 \to 8$
  - $15 \to 6$
  - $297 \to 18$
  - $11211 \to 6$

Ciferný součet: základní princip

opakovaně provádíme:

- dělení 10 se zbytkem – hodnota poslední cifry
- celočíselné dělení – „okrajování“ čísla

Ciferný součet – nevhodná pasáž

```
if n % 10 == 0:
    f = 0 + f
elif n % 10 == 1:
    f = 1 + f
elif n % 10 == 2:
    f = 2 + f
elif n % 10 == 3:
    f = 3 + f
elif n % 10 == 4:
    f = 4 + f
...
```

Ciferný součet – řešení

```
def digit_sum(n):
    result = 0
    while n > 0:
        result += n % 10
        n = n // 10
    return result
```

Ciferný součet: pokročilé řešení

Pro zajímavost:

```
def digit_sum(n):
    return sum(map(int, str(n)))
```

Return vs. print

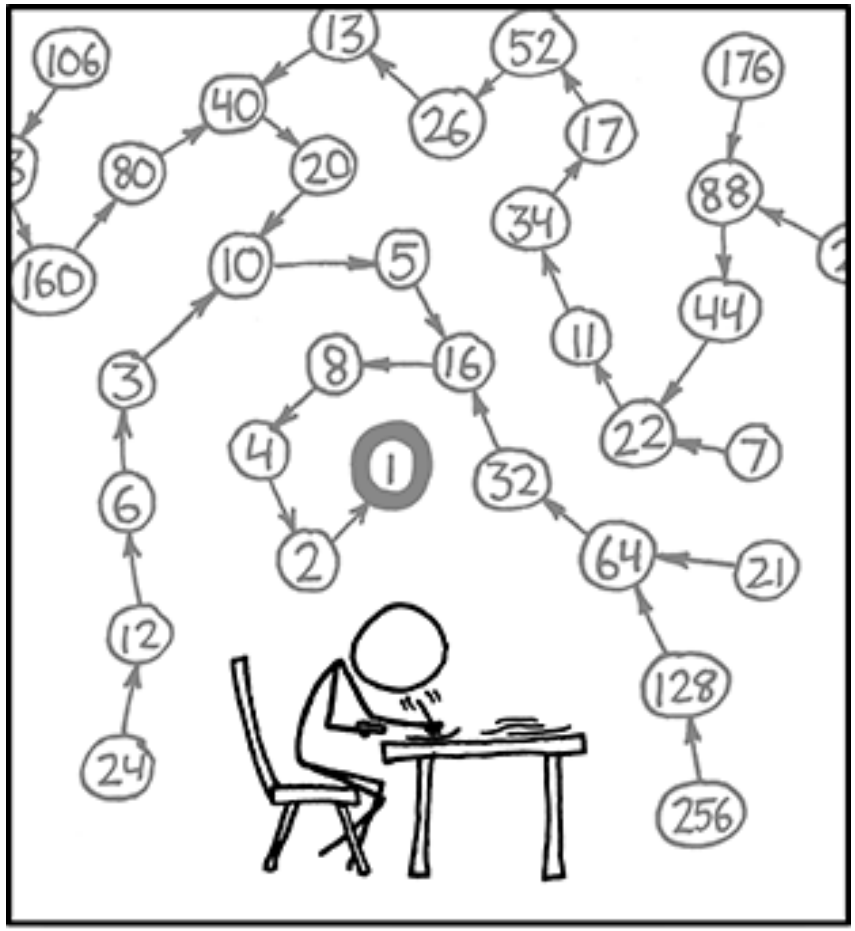
připomenutí:

- `print` = výpis
- `return` = návratová hodnota, se kterou můžeme dále pracovat
  - blízký vztah k matematickým funkcím

příklad:
výpis všech čísel menších jak 1000 s ciferným součtem 13

Collatzova posloupnost

- vezmi přirozené číslo:
  - pokud je sudé, vyděl jej dvěma
  - pokud je liché, vynásob jej třemi a přičti jedničku
- tento postup opakuj, dokud nedostaneš číslo jedna



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

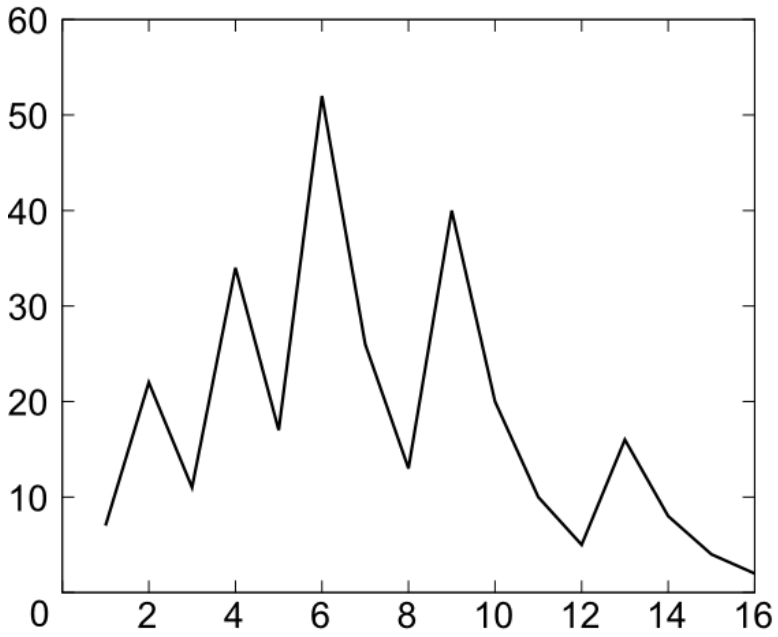
https://xkcd.com/710/

Collatzova posloupnost: výpis

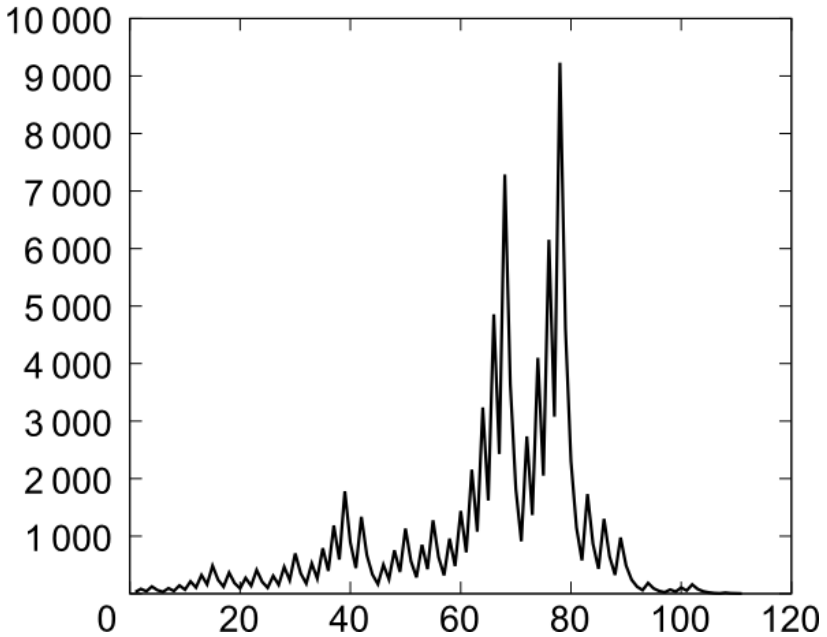
```
def collatz_sequence(n):
    while n != 1:
        print(n, end=", ")
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3*n + 1
    print(1)
```

Collatzova posloupnost: příklady graficky

začínající číslem 7



začínající číslem 27



Bonus: Vykreslení grafu v Pythonu

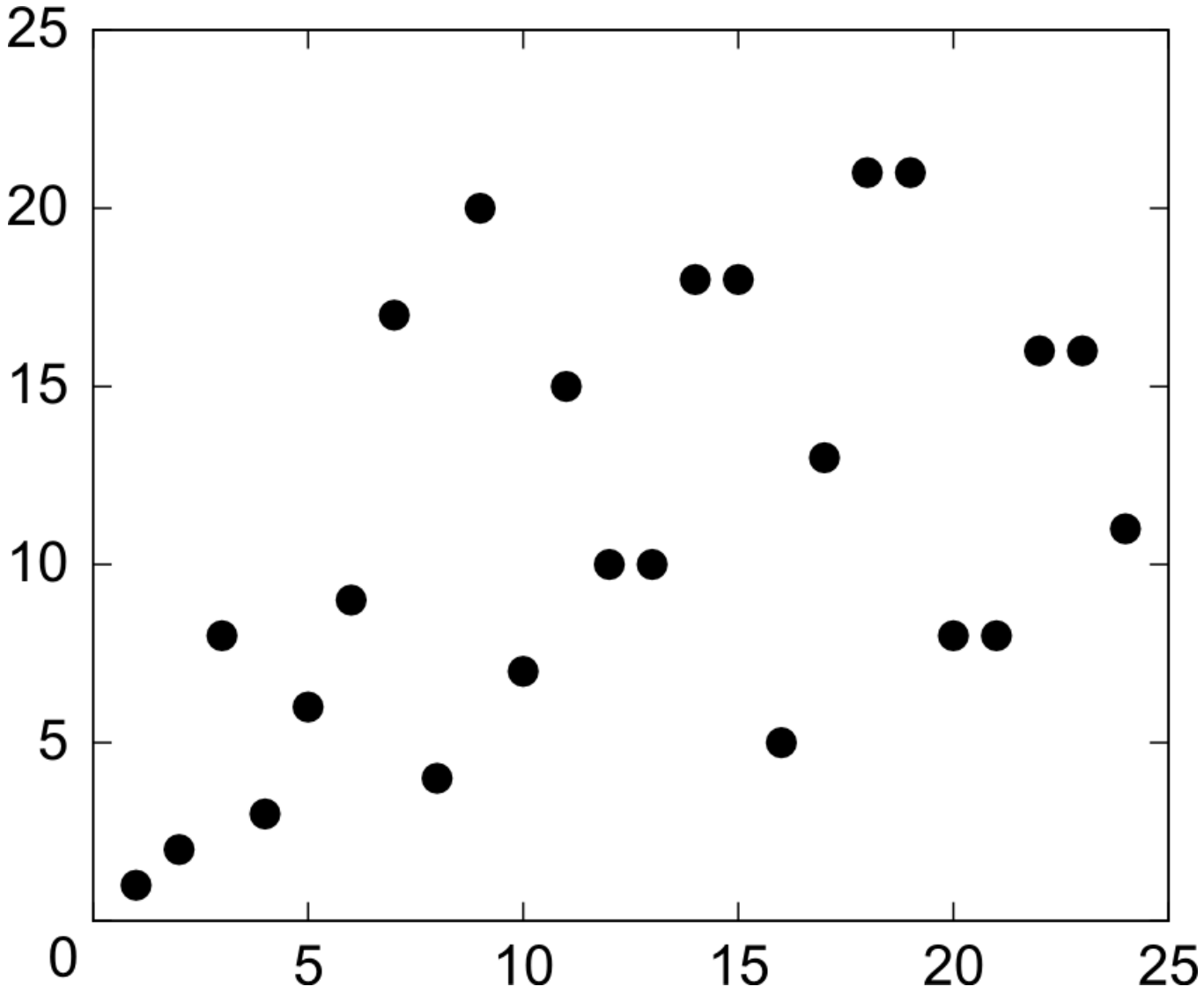
Využívá seznamy a knihovnu `pylab`

```
import pylab

def collatz(n):
    sequence = []
    while n != 1:
        sequence.append(n)
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3*n + 1
    sequence.append(1)
    return sequence
pylab.plot(collatz(27))
pylab.show()
```

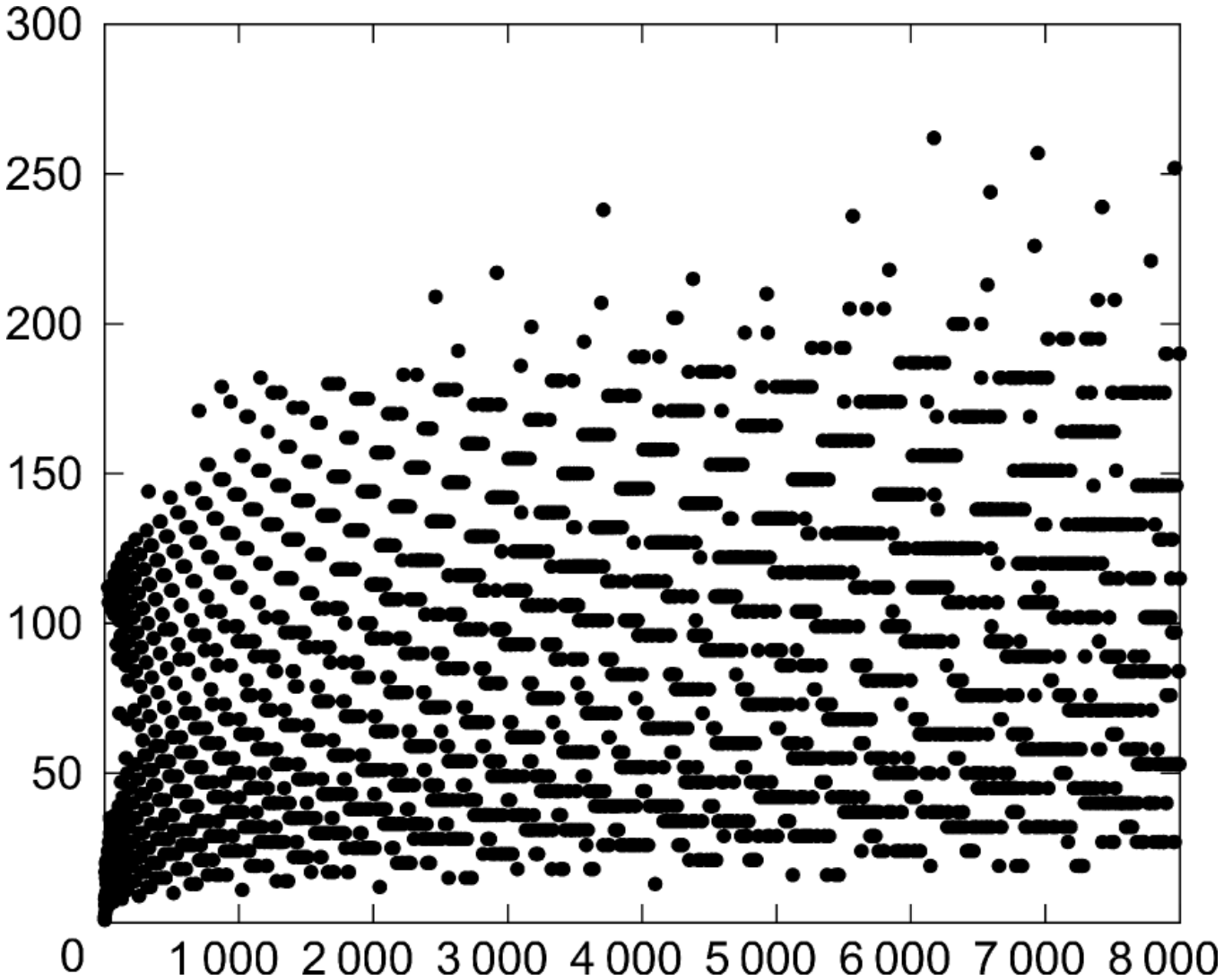
Collatzova posloupnost: délka posloupnosti

```
def collatz_length(n):
    length = 1
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3*n + 1
        length += 1
    return length

def collatz_table(count):
    for i in range(1, count+1):
        print(i, collatz_length(i))
```

Collatzova posloupnost: délka posloupnosti I



Collatzova posloupnost: délka posloupnosti II



Collatzova hypotéza

- Hypotéza: Pro každé počáteční číslo n, posloupnost narazí na číslo 1.
- experimentálně ověřeno pro velká n ($\sim 10^{18}$)
- důkaz není znám

Největší společný dělitel

- vstup: přirozená čísla a, b
- výstup: největší společný dělitel a, b
- příklad: 180, 504

Jak na to?

Naivní algoritmus I

- projít všechny čísla od 1 do $min(a, b)$
- pro každé vyzkoušet, zda dělí a i b
- vzít největší

Naivní algoritmus II

- „školní“ algoritmus
- najít všechny dělitele čísel a, b
- projít dělitele, vybrat společné, vynásobit
- příklad:
  - $180 = 2^2 \cdot 3^2 \cdot 5$
  - $504 = 2^3 \cdot 3^2 \cdot 7$
  - $NSD = 2^2 \cdot 3^2 = 36$

Euklidův algoritmus: základ

základní myšlenka: pokud $a > b$, pak:

$$NSD(a, b) = NSD(a - b, b)$$

příklad:

krok	a	b
1	504	180
2	324	180
3	180	144
4	144	36
5	108	36
6	72	36
7	36	36
8	36	0

Neefektivita

- uvedená verze je neefektivní
- může být pomalejší než naivní algoritmus I
- kdy?

Euklidův algoritmus: vylepšení

vylepšená základní myšlenka: pokud $a > b$, pak:

$$NSD(a, b) = NSD(a \ mod \ b, b)$$

krok	a	b
1	504	180
2	180	144
3	144	36
4	36	0

Euklidův algoritmus: program

bez rekurze

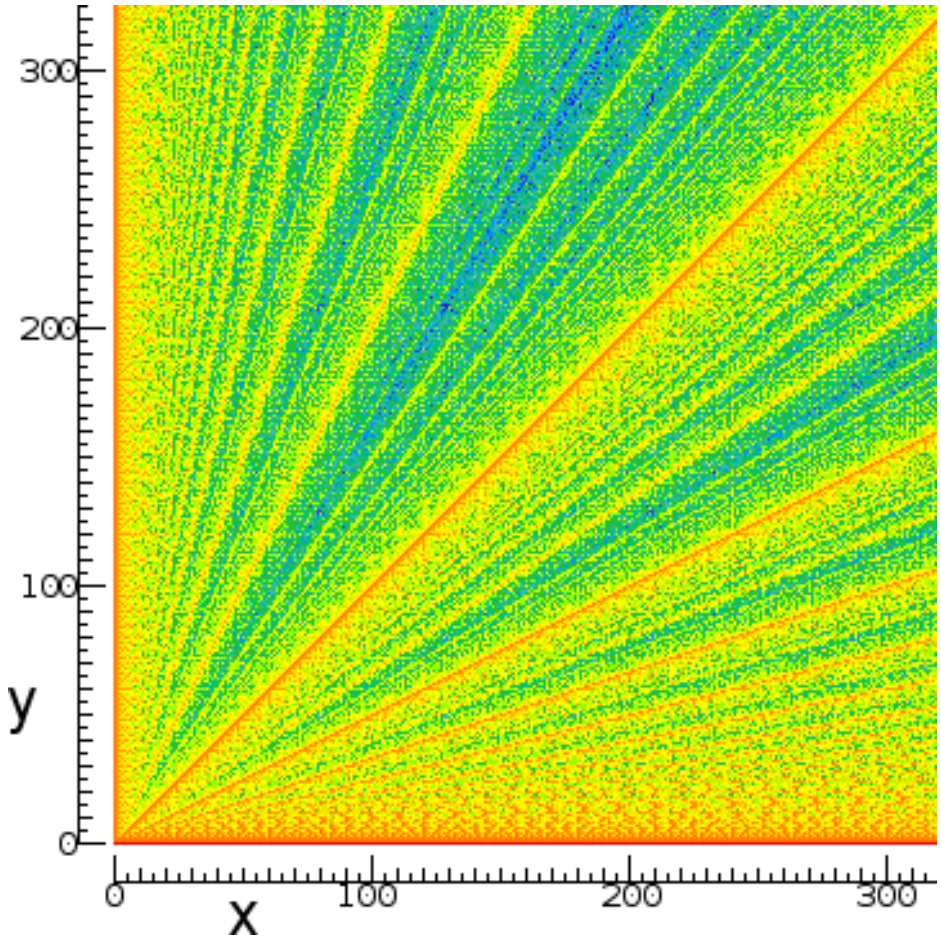
```
def gcd_euclid(a, b):
    if a == 0:
        return b
    while b != 0:
        if a > b:
            a = a % b
        else:
            b = b % a
    return a
```

Euklidův algoritmus: program

modulo varianta, rekurzivně

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

Euklidův algoritmus – vizualizace



http://en.wikipedia.org/wiki/Euclidean_algorithm

Efektivita algoritmů

- proč byly první dva algoritmy označeny jako „naivní“?
- časová náročnost algoritmu:
  - naivní: exponenciální vůči počtu cifer
  - Euklidův: lineární vůči počtu cifer
- různé algoritmy se mohou výrazně lišit svou efektivností
- často rozdíl použitelné vs nepoužitelné
- více později (a v dalších předmětech)

Kahoot

Kahoot program A

```
i = 1
while i < 3:
    print("x", end=" ")
    i = i + 1
print(i)
```

Kahoot program B

```
def increase(x):
    return x + 1

a = 5
increase(a)
print(a)
```

Výpočet odmocniny

- vstup: číslo x
- výstup: přibližná hodnota $\sqrt{x}$

Jak na to?

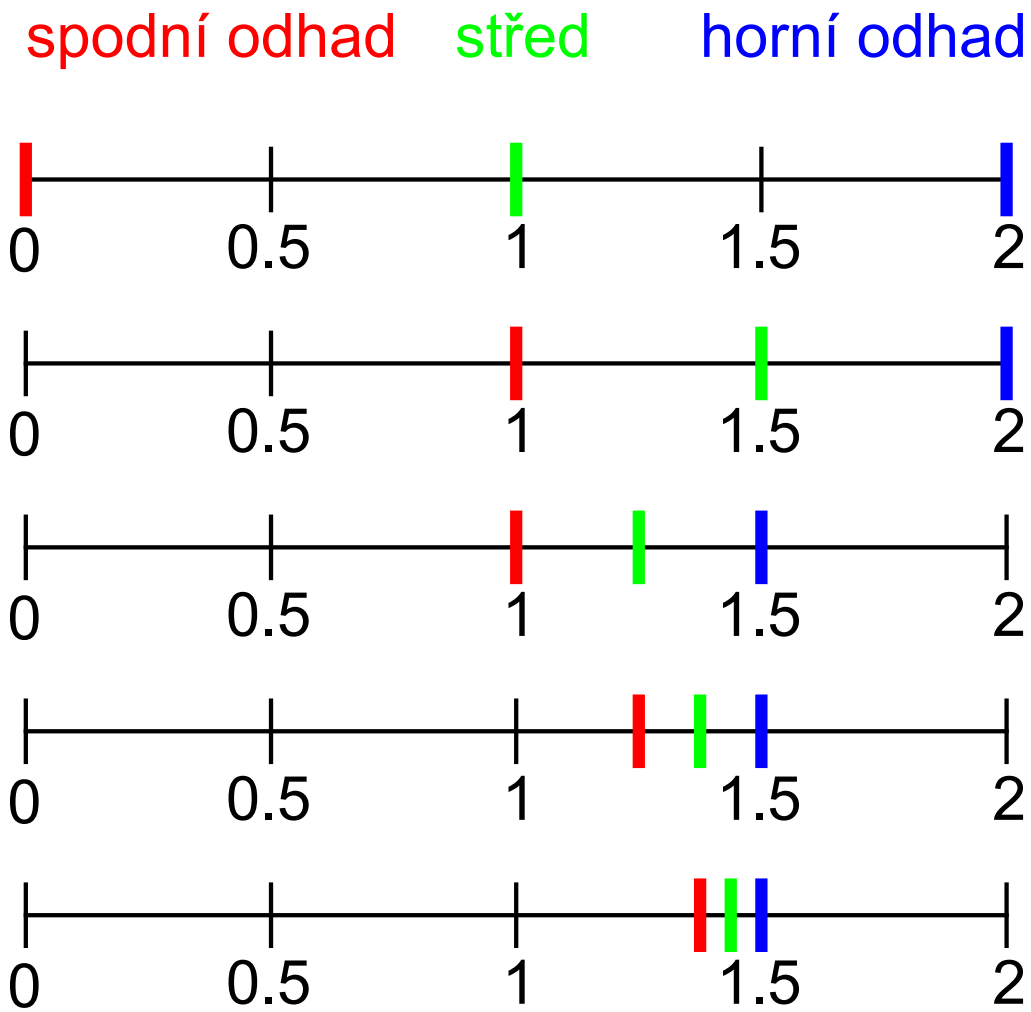
Výpočet odmocniny

- vstup: číslo x
- výstup: přibližná hodnota $\sqrt{x}$

Jak na to?

Mnoho metod, ukázka jedné z nich (rozhodně ne nejvíce efektivní)

Výpočet odmocniny: binární půlení



Výpočet odmocniny: binární půlení

```
def square_root(x, precision=0.01):
    upper = x
    lower = 0
    middle = (upper + lower) / 2
    while abs(middle**2 - x) > precision:
        print(lower, upper, sep="\t")
        if middle**2 > x:
            upper = middle
        if middle**2 < x:
            lower = middle
        middle = (upper + lower) / 2
    return middle
```

Výpočet odmocniny – chyba

Drobný problém: Program není korektní.

Kde je chyba?

Výpočet odmocniny – poznámky

- Funguje korektně jen pro čísla ≥ 1.
- Co program udělá pro čísla < 1?
- Proč?
- Jak to opravit?

Vsuvka: Obecný kontext

problém

$\Downarrow$

algoritmus

$\Downarrow$

program

$\Downarrow$

ladění

Poznámka o ladění

- laděním se nebudeme (na přednáškách) příliš zabývat
- to ale neznamená, že není důležité...

Ladění je dvakrát tak náročné, jak psaní vlastního kódu. Takže pokud napíšete program tak chytře, jak jen umíte, nebudete schopni jej odladit. (Brian W. Kernighan)

Postřeh k ladění

Do průšvihu nás nikdy nedostane to, co nevíme. Dostane nás tam to, co víme příliš jistě a ono to tak prostě není. (Y. Berry)

Ladění

- ladící výpisy
  - např. v každé iteraci cyklu vypisujeme stav proměnných
  - doporučeno vyzkoušet na ukázkových programech ze slidů
- použití debuggeru
  - dostupný přímo v IDLE
  - sledování hodnot proměnných, spuštěných příkazů, breakpointy, ...

Ladění a funkce

- dobrá dekompozice na funkce usnadňuje ladění
- „hledání chyby v celém programu“ vs. „hledání chyby v dílčí funkci“
- „unit testing“, „test driven development“

Čtení chybových hlášek

```
Traceback (most recent call last):
  File "sorting.py", line 63, in <module>
    test_sorts()
  File "sorting.py", line 59, in test_sorts
    sort(a)
  File "sorting.py", line 52, in insert_sort
    a[j] = curent
NameError: name 'curent' is not defined
```

- kde je problém? (identifikace funkce, číslo řádku)
- co je za problém (typ chyby)

Základní typy chyb

- SyntaxError
  - invalid syntax: zapomenutá dvojtečka či závorka, záměna = a ==, ...
  - EOL while scanning string literal: zapomenutá uvozovka
- NameError – špatné jméno proměnné (překlep v názvu, chybějící inicializace)
- IndentationError – špatné odsazení
- TypeError – nepovolená operace (sčítání čísla a řetězce, přiřazení do řetězce, ...)
- IndexError – chyba při indexování řetězce, seznamu a podobně („out of range“)

Časté chyby

projeví se „rychle“ (program spadne hned):

- zapomenutá dvojtečka, závorka, uvozovka
- překlepy
- použití $=$ tam, kde mělo být $==$
- špatný počet argumentů při volání funkce
- zapomenuté `len` v “`for i in range(alist)`”

Časté chyby

nemusí se projevit rychle / vždy:

- použití == tam, kde mělo být =
- `"True"` místo `True`
- záměna `print` a `return`
- dělení nulou
- chybné indexování (řetězce, seznamy)

Součet druhých mocnin

- Lze zapsat zadané číslo jako součet druhých mocnin?
- Příklad: $13 = 2^2 + 3^2$
- Která čísla lze zapsat jako součet druhých mocnin?

Součet druhých mocnin: řešení I

```
def sum_of_squares_test(n):
    for i in range(n+1):
        for j in range(n+1):
            if i**2 + j**2 == n:
                print(n, "=", i**2, "+", j**2)
```

- Program je zbytečně neefektivní. Proč?
- Výpis čísel, která lze zapsat jako součet čtverců

Testování druhé mocniny: nevhodný if

```
def is_square(n):
    square_root = int(n**0.5)
    if square_root**2 == n:
        return True
    else:
        return False
```

Součet druhých mocnin: řešení II

```
def is_square(n):
    square_root = int(n**0.5)
    return square_root**2 == n

def is_sum_of_squares(n):
    for i in range(int(n**0.5) + 1):
        rest = n - i**2
        if is_square(rest):
            return True
    return False

def print_sums_of_squares(count):
    for i in range(count):
        if is_sum_of_squares(i):
            print(i, end=", ")
```

Podobné náměty

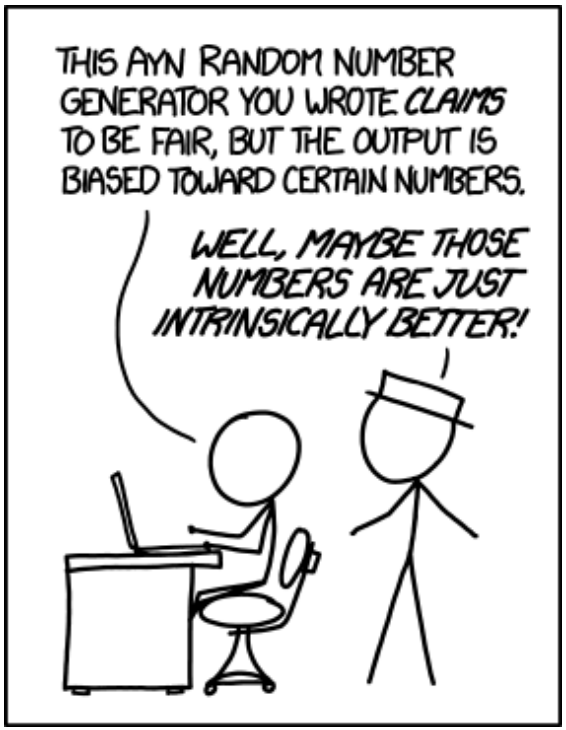
- variace: součet tří druhých mocnin, součet dvou třetích mocnin, ...
- další náměty na posloupnosti: *The On-Line Encyclopedia of Integer Sequences*, `http://oeis.org/`

Náhodná čísla

- přesněji: *pseudo-náhodná* čísla
- opravdová náhodná čísla: `https://www.random.org/`
- bohaté využití v programování: výpočty, simulace, hry, ...
- Python
  - `import random`
  - `random.random()` – float od 0 do 1
  - `random.randint(a, b)` – celé číslo mezi `a`, `b`
  - mnoho dalších funkcí

Náhodná čísla: xkcd

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```



https://xkcd.com/221/
https://xkcd.com/1277/

Náhodná čísla: průměr vzorku

Vygenerujeme náhodná čísla a vypočítáme průměrnou hodnotu:

```
def random_average(count, maximum=100):
    total = 0
    for i in range(count):
        total += random.randint(0, maximum)
    return total / count
```

Jakou očekáváme hodnotu na výstupu? Jak velký bude rozptyl hodnot? (Názorná ukázka *centrální limitní věty*)

Simulace volebního průzkumu

- volební průzkumy se často liší; jaká je jejich přesnost?
- přístup 1: matematické modely, statistika
- přístup 2: simulace
- program:
  - vstup: preference stran, velikost vzorku
  - výstup: preference zjištěné v náhodně vybraném vzorku

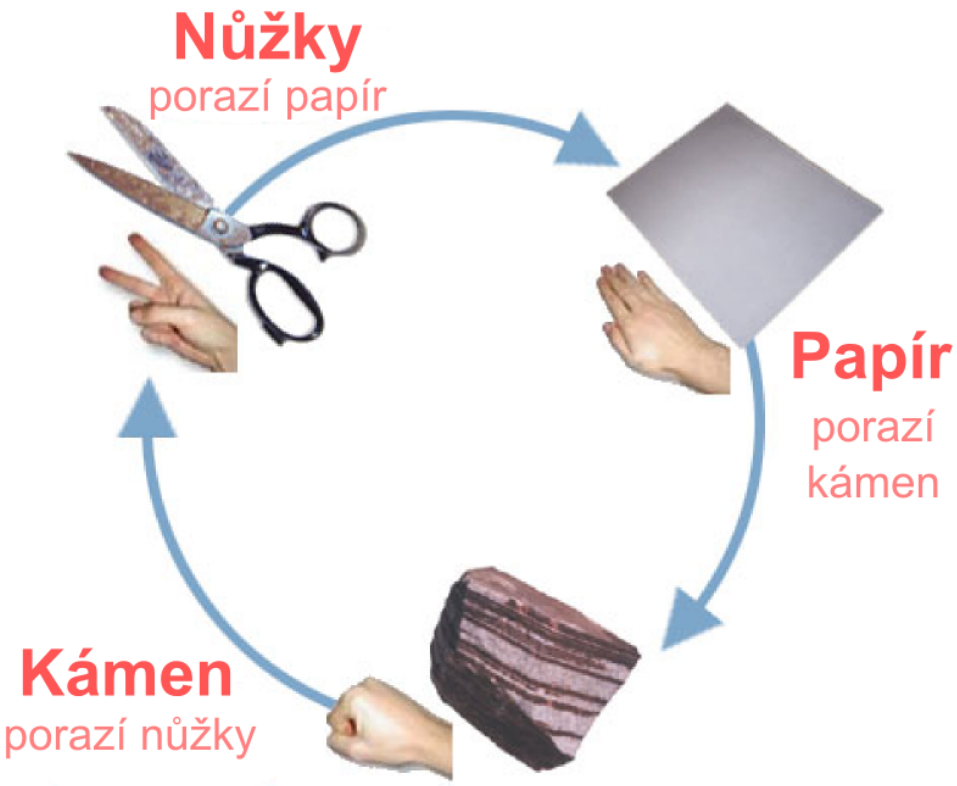
Simulace volebního průzkumu

```
def survey(size, pref1, pref2, pref3):
    count1 = 0
    count2 = 0
    count3 = 0
    for i in range(size):
        r = random.randint(1, 100)
        if r <= pref1: count1 += 1
        elif r <= pref1 + pref2: count2 += 1
        elif r <= pref1 + pref2 + pref3: count3 += 1
    print("Party 1:", 100 * count1 / size)
    print("Party 2:", 100 * count2 / size)
    print("Party 3:", 100 * count3 / size)
```

Poznámky ke zdrojovému kódu

- uvedené řešení není dobré:
  - „copy & paste“ kód
  - funguje jen pro 3 strany
- lepší řešení – využití seznamů

Kámen, nůžky, papír



Zdroj: Wikipedia

KNP: strategie

```
def strategy_uniform():
    r = random.randint(1, 3)
    if r == 1:
        return "R"
    elif r == 2:
        return "S"
    else:
        return "P"

def strategy_rock():
    return "R"
```

KNP: vyhodnocení tahu

```
def evaluate(symbol1, symbol2):
    if symbol1 == symbol2:
        return 0
    if symbol1 == "R" and symbol2 == "S" or \
       symbol1 == "S" and symbol2 == "P" or \
       symbol1 == "P" and symbol2 == "R":
        return 1
    return -1
```

KNP: sehrání západu

```
def rsp_game(rounds):
    points = 0
    for i in range(1, rounds+1):
        print("Round ", i)
        symbol1 = strategy_uniform()
        symbol2 = strategy_uniform()
        print("Symbols:", symbol1, symbol2)
        points += evaluate(symbol1, symbol2)
        print("Player 1 points:", points)
```

KNP: obecnější strategie

```
def strategy(weightR, weightS, weightP):
    r = random.randint(1, weightR + weightS + weightP)
    if r <= weightR:
        return "R"
    elif r <= weightR + weightS:
        return "S"
    else:
        return "P"
```

KNP: rozšiřující náměty

- turnaj různých strategií
- strategie pracující s historií
  - kopírování posledního tahu soupeře
  - analýza historie soupeře (hraje vždy kámen? $\rightarrow$ hraj papír)
- rozšíření na více symbolů (Kámen, nůžky, papír, ještěr, Spock)

Shrnutí

- operace s čísly, náhoda
- ukázky programů
- ukázky algoritmů, efektivita

Příště: Seznamy, řetězce a trocha šifer