IB113 Úvod do programování a algoritmizace
Přednáška 4

Náhodná čísla, simulace
Práce s řetězci

Nikola Beneš

9. říjen 2017

Co bude dnes?

Ladění a chyby

- doplnění k minulé přednášce

Náhodná čísla

- generování náhodných čísel
- využití náhodných čísel: simulace, hry, ...

Práce s řetězci

- základní operace s řetězci
- základy zpracování textu
- formátování výstupu

Ladění a chyby

Čtení chybových hlášek

- chyby při spuštění
- chyby za běhu

```
Traceback (most recent call last):
  File "chessboard.py", line 14, in <module>
    chessboard(8)
  File "chessboard.py", line 12, in chessboard
    line(n, i % 2)
  File "chessboard.py", line 3, in line
    if i % 2 == party:
NameError: name 'party' is not defined
```

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

- řadu chyb umí odhalit vývojové prostředí samo

Základní typy chyb

SyntaxError (chyba při spuštění)

- špatná syntax: chybějící dvojtečka nebo závorka, záměna = a ==

IndentationError (chyba při spuštění)

- špatné odsazení

NameError

- špatné jméno proměnné: překlep v názvu, chybějící inicializace

TypeError

- špatný typ pro danou operaci: sčítání čísla a řetězce, ...

IndexError

- chyba při indexování řetězce, seznamu apod. (uvidíme časem)

... a další

Časté chyby

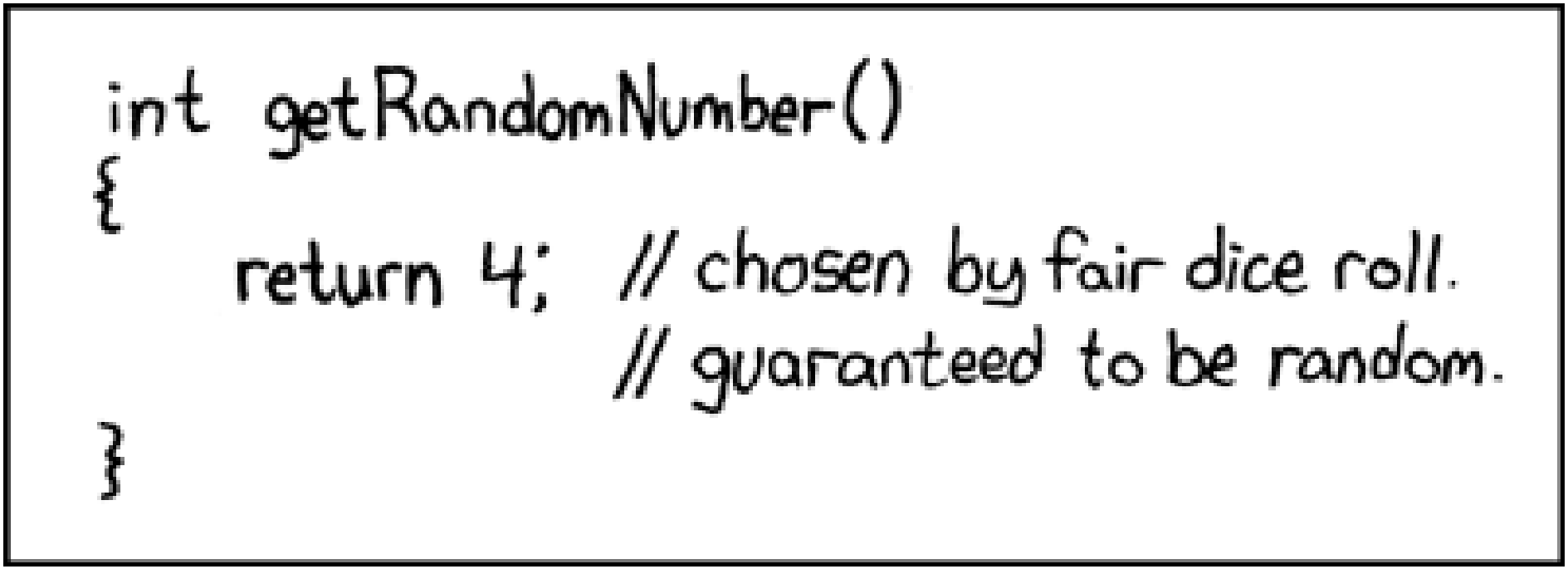
projeví se rychle:

- chyby syntaxe (zapomenutá dvojtečka, závorka, uvozovka, ...)
- překlepy (v lepším případě)
- použití `=` tam, kde mělo být `==`
- špatný počet parametrů při volání funkce

nemusí se projevit rychle

- použití `==` tam, kde mělo být `=`
- `"True"` místo `True`
- chybné indexování
- záměna `print` a `return`
- chyba v logice programu (špatný algoritmus)

Náhodná čísla



Zdroj: `https://xkcd.com/221/`

Náhodná čísla

Náhodná čísla v počítači

- pseudo-náhodná čísla (případ Pythonu i jiných jazyků)
- zdroj entropie (náhody)
- opravdová náhodná čísla? `http://www.random.org`

Využití náhodných čísel při programování

- náhodnostní algoritmy
  - některé problémy se snadněji řeší s použitím náhody
- simulace
- hry
- ...

Náhodná čísla v Pythonu

Knihovna `random`

- `import random` nebo `from random import jméno_funkce`
- `random.random()` – float mezi 0 a 1
- `random.randint(a, b)` – celé číslo mezi `a` a `b` (*včetně*)
- mnoho dalších funkcí
  - `random.choice`, `random.sample` – náhodný výběr z daného seznamu prvků
  - `random.shuffle` – náhodné zamíchání seznamu prvků
  - jiná rozdělení: `random.normalvariate`, ...

```
from random import randint

print(randint(1, 6))
```

Příklad: Průměr náhodných čísel

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

- jakou očekáváme hodnoty na výstupu?
- jak velký bude rozptyl hodnot?

Příklad: Simulace volebního průzkumu

- volební průzkumy se často liší
- jaká je jejich přesnost?
- přístupy:
  - matematické modely, statistika
  - simulace

- simulace volebního průzkumu
  - vstup: preference stran, velikost vzorku
  - výstup: preference zjištěné v náhodně vybraném vzorku

Příklad: 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:", 100.0 * count1 / size)
    print("Party 2:", 100.0 * count2 / size)
    print("Party 3:", 100.0 * count3 / size)
```

Příklad: Simulace volebního průzkumu

- řešení na předchozím slajdu není moc dobré; proč?
  - opakující se kód (není DRY)
  - funguje jen pro 3 strany
- lepší řešení: využití seznamů (později)

Generování náhodných čísel v Pythonu

Deterministické generování náhodných čísel

- Python i jiné programovací jazyky
- výchozí stav – „semínko“ (seed)
  - můžeme nastavit i sami
  - pokud nenastavíme, vezme se aktuální čas

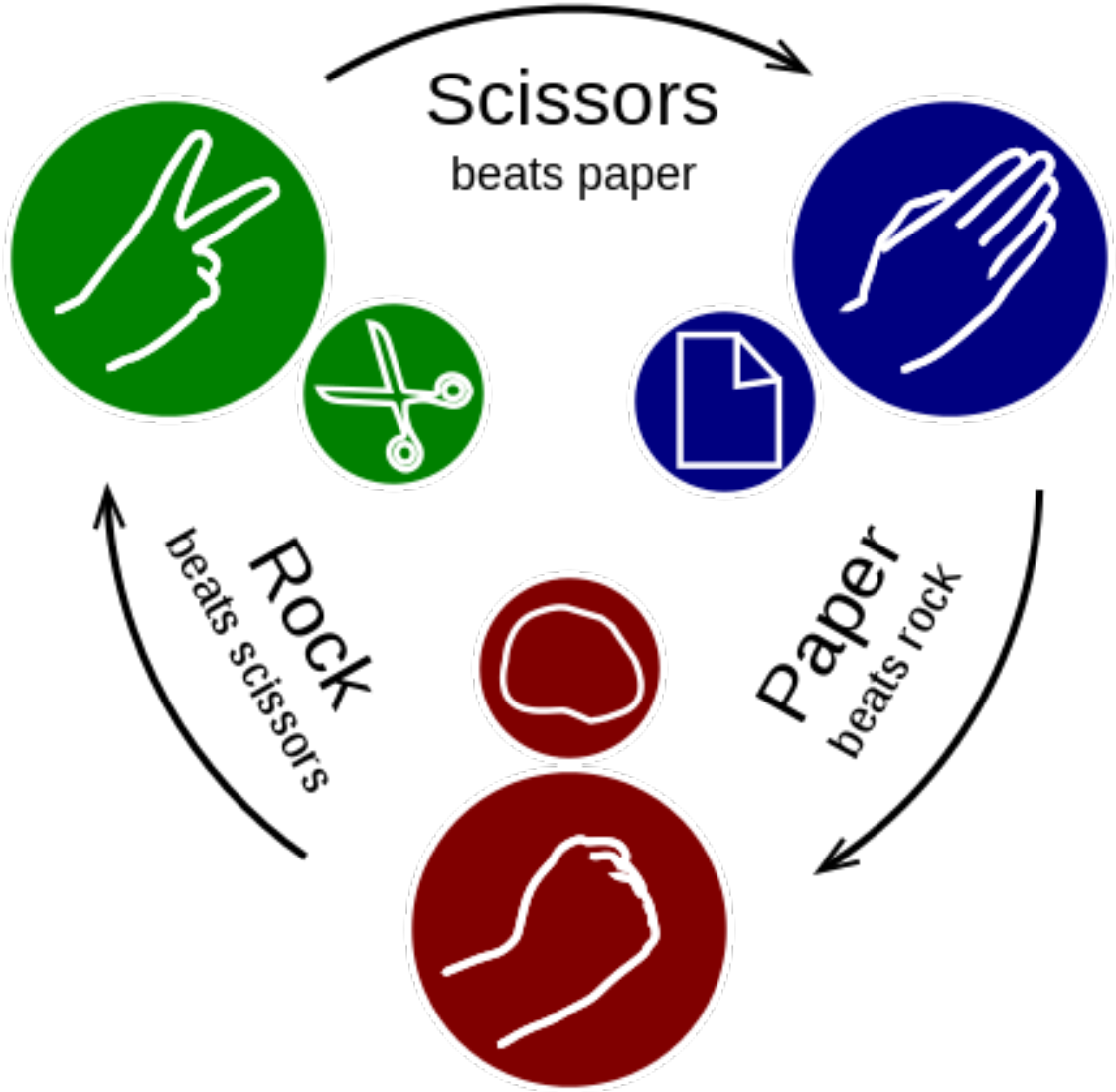
```
from random import randint, seed

seed(42)  # zkuste zakomentovat

print(randint(1, 1000))
```

- opakované spuštění s pevným seedem povede ke stejnému výsledku

Příklad: Kámen, nůžky, papír



https://commons.wikimedia.org/w/index.php?curid=27958688

Příklad: Kámen, nůžky, papír

Strategie

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


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

Příklad: Kámen, nůžky, papír

Vyhodnocení jednoho kola

```
def eval_round(symbol1, symbol2):
    if symbol1 == symbol2:
        return 0  # remíza
    if (symbol1 == "R" and symbol2 == "S") or \
       (symbol1 == "S" and symbol2 == "P") or \
       (symbol1 == "P" and symbol2 == "R"):
        return 1  # vyhrál první hráč
    return 2 # vyhrál druhý hráč
```

Příklad: Kámen, nůžky, papír

Sehrání zápasu

```
def rsp_game(rounds):
    wins1 = 0
    wins2 = 0
    for i in range(rounds):
        print("Round", i + 1)
        symbol1 = strategy_uniform()
        symbol2 = strategy_uniform()
        print("Symbols:", symbol1, symbol2)
        winner = eval_round(symbol1, symbol2)
        if winner == 1:
            print("Player 1 wins this round.")
            wins1 += 1
        elif winner == 2:
            print("Player 2 wins this round.")
            wins2 += 1
```

Příklad: Kámen, nůžky, papír

Sehrání zápasu (pokračování)

```
if wins1 > wins2:
    print("Player 1 wins!")
elif wins1 < wins2:
    print("Player 2 wins!")
else
    print("It's a tie!")
```

- náměty pro další rozšíření
  - obecnější strategie
  - turnaj různých strategií
  - strategie s historií (kopírování posledního tahu soupeře, ...)
  - rozšíření na víc symbolů (kámen, nůžky, papír, tapír, Spock)

Příklad: Kámen, nůžky, papír

(pro zajímavost)

- parametry funkcí mohou být jiné funkce

```
def rsp_game(rounds, strategy1, strategy2):
    # ...
    for i in range(rounds):
        # ...
        symbol1 = strategy1()
        symbol2 = strategy2()
        # ...


rsp_game(100, strategy_uniform, strategy_bart_simpson)
```

Řetězce

Syntax

- uvozovky: `"Toto je text"`
- alternativně apostrofy: `'Toto je taky text.'`
- víceřádkové řetězce:

```
text = """Toto je text,
který pokračuje na dalším řádku.
A potom ještě na jednom."""
```

- speciální symboly – uvozené zpětným lomítkem \
  - konec řádku `\n`
  - tabulátor `\t`
  - uvozovka `\"`, apostrof `\'`
  - zpětné lomítko `\\`
  - … a různé jiné

```
print("\N{grinning face}")
```

Řetězce

Základní operace s řetězci

- zřetězení: `"ahoj, " + "lidi"`
- „násobení“: `"kolo" * 3`
- porovnávání
- přetypování na řetězec: `str(3.14)`
- přiřazení do proměnné: `text = "Dnes je pěkně."`
- délka textu (počet znaků): `len(text)`
- indexování (výběr konkrétního znaku): `text[0]`, `text[2]`, `text[-1]`
- číslo znaku: `ord("b")`
- znak pro zadané číslo: `chr(99)`

Indexování od nuly

- částečně historicko-technické důvody
- ale i dobré „matematické“ důvody
- souvisí s oblibou polouzavřených intervalů $\langle od, do)$

Pro zajímavost:

- `https://en.wikipedia.org/wiki/Zero-based_numbering`
- `https://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD831.html`
- `https://softwareengineering.stackexchange.com/questions/110804/why-are-zero-based-arrays-the-norm`

Řetězce – kódování

Jak jsou znaky reprezentovány v počítači?

- ASCII, ISO8859-2, Windows-1250, Unicode (UTF-8, UTF-16, UTF-32), ...
- Python umí pracovat s různými kódováními
- zdrojový soubor programu je v UTF-8 (dá se změnit) (Pythonu 3)
- interní reprezentace řetězců v Pythonu:
  - složitější (PEP-393), ale pokrývá celý Unicode

- pro účely tohoto předmětu:
  - `ord`, `chr` – převod znaků na čísla a zpět
  - znaky anglické abecedy mají přiřazena po sobě jdoucí čísla

```
for i in range(26):
    print(chr(ord('A') + i))
```

Řetězce – pokročilejší indexování

(specifické pro Python)

```
text[0:3]     # první 3 znaky
text[:3]      # --- totéž ---
text[3:]      # od 4. znaku do konce
text[1:8:2]   # od 2. znaku po 7., jen každý druhý znak
text[::3]     # od začátku do konce, každý třetí
```

- všimněte si podobnosti s `range()` – není náhodná

Řetězce – změny

- řetězce v Pythonu jsou neměnné (*immutable*)
  - rozdíl proti řetězcům v některých jiných jazycích
  - rozdíl proti seznamům v Pythonu (příště)
- místo změny znaku musíme vytvořit nový řetězec

```
text = "holinky"
text[2] = "d"  # chyba!
text = text[:2] + "d" + text[3:]
```

Řetězce – další operace

```
text = "hello, world!"
print(text.upper())
print(text.lower())
print(text.capitalize())
print(text.rjust(30))
print(">" + text.center(30) + "<")
print(text.replace("world", "people"))
```

- procházení řetězce po znacích

```
for c in text:
    print("znak:", c)
```

... a mnoho dalších (viz dokumentace, pozdější přednáška)

Řetězce – formátování

Vkládání čísel (a jiných hodnot) do řetězců

- starý způsob:
  `"Jmenuji se %s a je mi %d let." % ("Metuzalém", 900)`
- nový způsob (od Pythonu 3.1): metoda `format()`

```
print("Dnes je {}. {}. ({})".format(9, "října", "pondělí"))
print("Je právě {:d}.{:02d}".format(13, 7))
print("X{:<10}X".format("text"))
print("X{:>10}X".format("text"))
print("X{:^10}X".format("text"))
print("{:.2f}".format(math.pi))
```

- `format` toho umí spoustu: `https://pyformat.info/`

Příklad: Caesarova šifra

- vstup: text, posun
- výstup: zašifrovaný text

```
def caesar_cipher(text, shift):
    result = ""
    text = text.upper()
    for char in text:
        if char.isalpha():
            c = ord(char) + shift
            if c > ord('Z'):
                c -= 26
            result += chr(c)
        else:
            result += char
    return result
```

- k zamyšlení: jak dešifrovat text, když neznáme posun?

Závěrečný kvíz

https://kahoot.it

Příklad: Monty Hallův problém

- americká televizní soutěž; moderátor Monty Hall
- troje dveře: za jedněmi je hlavní cena (auto), za zbývajícími prohra (koza)
- soutěžící si zvolí jedny dveře
- moderátor otevře *jiné* dveře, ale takové, že za nimi je koza
  - moderátor samozřejmě ví, kde je hlavní cena
- soutěžící má volbu:
  - *zůstat* u své původní volby dveří
  - *změnit* svou volbu na zbývající dveře

- je lepší *zůstat* nebo *změnit* volbu? nebo je to jedno?
  - jak bychom to mohli simulovat?

Závěr

Co jsme se dnes dozvěděli?

- existence knihovny pro generování náhodných čísel
- ve skutečnosti pseudonáhodná čísla
- použití náhodných čísel pro jednoduché simulace
- práce s řetězci – základní operace
- reprezentace znaků v počítači
- indexování řetězců
- formátování dat

Co bude příště?

- datové typy seznam, ntice