Praktické tipy, programovací jazyky

IB113
Radek Pelánek

2024

Dnešní témata

mnoho témat, rychlý průchod, „pro představu“

- vývoj programů, dokumentace, testování
- dělení projektu do souborů
- styl, PEP8
- vývojová prostředí, správa verzí
- přehled programovacích jazyků

Návrh a vývoj softwaru

„softwarové inženýrství“

vývoj rozsáhlého softwaru nejen o zvládnutí „programování“:

- specifikace, ujasnění požadavků
- návrh
- dokumentace
- testování
- integrace, údržba

mnoho metodik celého procesu: vodopád, spirála, agilní přístupy, ...

Dokumentace

pro koho:

- pro sebe (při vývoji i později)
- pro ostatní

jak:

- názvy (modulů, funkcí, proměnných)
- dokumentace funkcí, tříd, rozhraní
- komentáře v kódu

Dokumentace: obecné postřehy

- neaktuální dokumentace je často horší než žádná dokumentace
- sebe-dokumentující se kód
  Nejlepší kód je takový, který se dokumentuje sám.
  $\Rightarrow$ názvy funkcí, parametrů, dodržování konvencí, ...
- u rozsáhlých projektů dokumentace nezbytnost
- psaní *dobré* dokumentace – trochu umění, nezbytnost empatie

Dokumentace v Pythonu

- dokumentační řetězec (*docstring*)
- první řetězec funkce (třídy, metody, modulu)
- konvenčně zapisován pomocí „trojitých uvozovek“ (povolují víceřádkové řetězce)

```
def add(a, b):
    """Add two numbers and return the result."""
    return a + b
```

Dokumentační řetězec víceřádkový

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """
    if imag == 0.0 and real == 0.0:
        return complex_zero
    ...
```

Dokumentační řetězce: ukázky

```
"""
This string, being the first statement in the file,
will become the module's docstring when the file is
imported.
"""

class MyClass(object):
    """The class's docstring"""

    def my_method(self):
        """The method's docstring"""

def my_function():
    """The function's docstring"""
```

Dokumentace vs. komentáře

- dokumentační řetězec
  - **co** kód dělá, jak se používá (volá), ...
  - brán v potaz při zpracování, dá se s ním dále pracovat: `__doc__` atribut, nástroje pro automatické zpracování, ...
- komentáře (#)
  - **jak** kód funguje, jak se udržuje, ...
  - při zpracování ignorovány

Testování: obecné postřehy

Východisko: lidé dělají chyby

Přístupy k testování:

- nevhodný: „přesvědčit se, že program je v pořádku“
- vhodný: „najít chyby v kódu“

obecný kontext: „konfirmační zkreslení“ (confirmation bias)

Testování

rozsáhlé téma:

- testování vs formální verifikace
- různé úrovně testování: unit, integration, component, system, ...
- různé styly testování: black box, white box, ...
- různé typy testování: regression, functional, usability, ...
- metodiky (test-driven development), automatizované nástroje

unit testing (jednotkové testování) – testování samostatných „jednotek“ (např. funkce, metoda, třída)

Testování: dílčí tipy

- cíleně testujte okrajové podmínky, netypické příklady
  - prázdný seznam (řetězec)
  - záporná čísla, desetinná čísla
  - pravoúhlý trojúhelník, rovnoběžné přímky
  - přechodný rok
- „pokrytí kódu testem“ (code coverage)
  - Jsou pomocí testu „vyzkoušeny“ všechny části kódu (funkce, příkazy, podmínky)?

Assert

```
assert expression
```

- pokud `expression` není splněn, program „spadne“ (přesněji: je vyvolána výjimka, kterou lze odchytit)
- pomůcka pro ladění, testování
- explicitní kontrola implicitních předpokladů

Assert: příklad

```
def factorial(n):
    assert n == int(n) and n >= 0
    f = 1
    for i in range(1, n+1):
        f = f * i
    return f

print(factorial(-2))
```

```
Traceback (most recent call last):
  File "/home/xpelanek/temp/test.py", line 8, in <module>
    print(factorial(-2))
  File "/home/xpelanek/temp/test.py", line 2, in factorial
    assert n == int(n) and n >= 0
AssertionError
```

Refaktorování

- složitější kód nikdy nenapíšeme ideálně na poprvé
- refaktorování (code refactoring) – úprava kódu bez změny funkčnosti
- dobře napsané testy usnadňují refaktorování

Dělení projektu do souborů

- univerzální princip: velký projekt nechceme mít v jednom souboru
- důvody: jako dělení programu na funkce, o úroveň abstrakce výš

Dělení projektu do souborů

- jazykově specifické
- programovací jazyky se liší technickými požadavky i konvencemi
  - hlavičkové soubory v C
  - „hodně malých souborů“ v Javě
- terminologie: knihovny, moduly, balíčky, frameworky, ...

Terminologie v kontextu Pythonu

pojmy s přesným významem:

- **modul** (module): soubor s příponou .py, funkce/třídy s příbuznou funkcionalitou
- **balík** (package): kolekce příbuzných modulů, společná inicializace, ...

související pojmy používané volněji:

- **knihovna** (library)
- **framework** (framework)

Moduly v Pythonu

- modul poskytuje rozšiřující funkcionalitu
- zdroje modulů:
  - standardní distribuce (např. math, turtle)
  - separátní instalace (např. numpy, Image)
  - vlastní implementace (základ: „.py soubor ve stejném adresáři“)
- použití:
  - `import module` – následná volání `module.function()`
  - `import module as m`
  - `from module import function`
  - `from module import *` (nedoporučeno)

Jmenné prostory

jmenný prostor (*namespace*) $\sim$ mapování jmen na objekty

- jmenné prostory umožňují použití stejného jména v různých kontextech bez toho, aby to způsobilo problémy
- jmenné prostory mají funkce, moduly, třídy...
- moduly – tečková notace (podobně jako objekty)
  - `random.randint`
  - `math.log`

Proč nepoužívat „import *“

```
from X import *
```

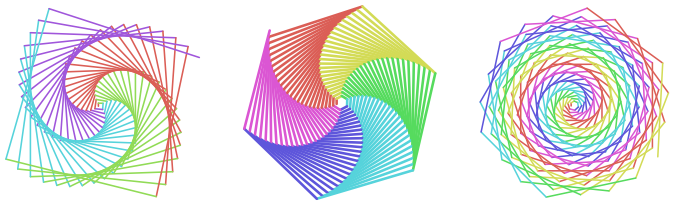
V malém programu nemusí vadit, ale ve větších projektech považováno za velmi špatnou praxi.

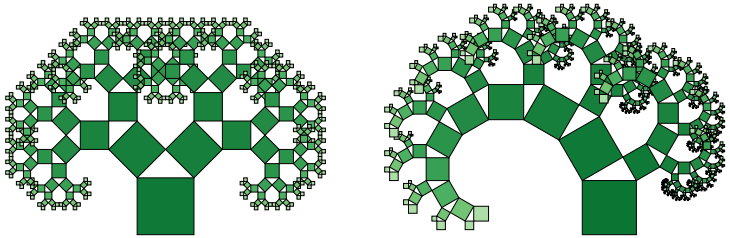
- „nepořádek“ v jmenném prostoru
- kolize, přepis
- překlepy se mohou chovat magicky
- složitější interpretace chybových hlášek
- Python Zen: Explicit is better than implicit. (import this)

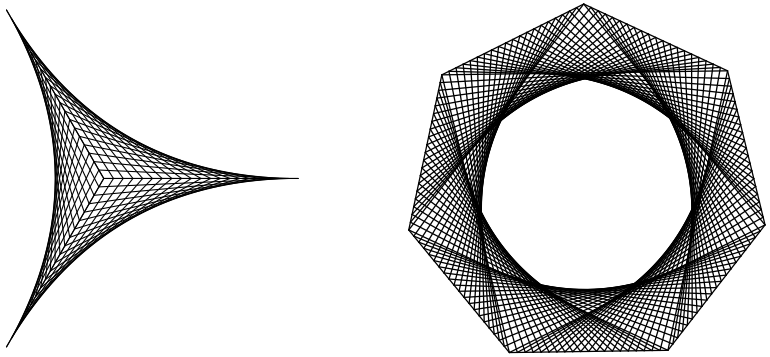
Moduly – praktický příklad

tvorba obrázků do knihy

`https://radekpelanek.cz/?zelva`





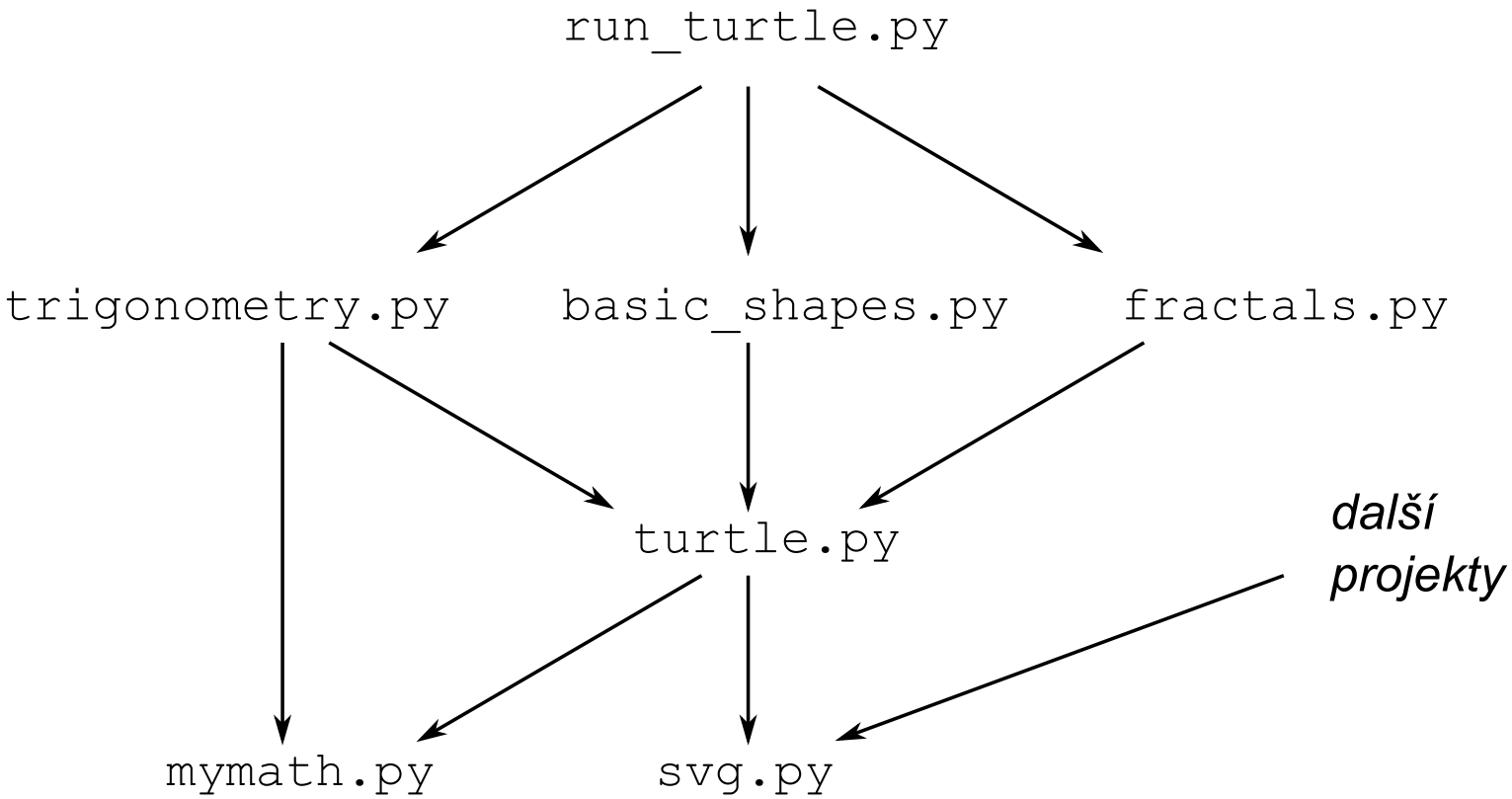


Moduly – praktický příklad

želví grafika, vykreslování obrázků do SVG (pro následnou manipulaci)

- `mymath.py` – trigonometrické funkce počítající ve stupních
- `svg.py` – generování SVG kódu, funkce typu:
  - `svg_header()`, `svg_line()`, `svg_circle()`
  - manipulace s celkovým obrázkem (posun, rámeček), uložení do souboru
- `turtle.py` – třída reprezentující želvu, podpora pro více želv

Moduly – praktický příklad



PEP8 – stylistická doporučení pro Python

`https://www.python.org/dev/peps/pep-0008/`

- výběr vybraných bodů
- obecný „duch“ doporučení celkem univerzální
- částečně však specifické pro Python (pojmenování, bílá místa, ...)

PEP8: Styl a konzistence

konzistence (rostoucí důležitost)

- s doporučeními
- v rámci projektu
- v rámci modulu či funkce

PEP8: Odsazování a délka řádků

- standardní odsazení: 4 mezery
- nepoužívat tabulátor
- maximální délka řádku 79 znaků
- rady k zalomení dlouhých řádků

PEP8: Prázdné řádky

- oddělení funkcí a tříd: 2 prázdné řádky
- oddělení metod: 1 prázdný řádek
- uvnitř funkce: 1 prázdný řádek pro oddělení logických celků (výjimečně)

PEP8: Bílé znaky ve výrazech

- mezera za čárkou
- mezera kolem přiřazení a binárních operátorů
  - zachování čitelnosti celkového výrazu
  - ne okolo = v definici defaultní hodnoty argumentu
- nepoužívat přebytečné mezery uvnitř závorek

PEP8: Bílé znaky – příklady

```
Ano: spam(ham[1], {eggs: 2})
Ne:  spam( ham[ 1 ], { eggs: 2 } )

Ano: if x == 4: print(x, y); x, y = y, x
Ne:  if x == 4 : print( x , y ); x , y = y , x

Yes: spam(1)
No:  spam (1)

Yes: dct['key'] = lst[index]
No:  dct ['key'] = lst [index]
```

PEP8: Bílé znaky – příklady

Ano:

```
x = 1
y = 2
long_variable = 3
```

Ne:

```
x             = 1
y             = 2
long_variable = 3
```

PEP8: Bílé znaky – příklady

Ano:

```
i = i + 1
submitted += 1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

Ne:

```
i=i+1
submitted +=1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

PEP8: Pojmenování – přehled stylů

```
lowercase
lower_case_with_underscores (snake_case)
UPPERCASE
UPPER_CASE_WITH_UNDERSCORES
CapitalizedWords (CapWords, CamelCase, StudlyCaps)
mixedCase
Capitalized_Words_With_Underscores
_single_leading_underscore
single_trailing_underscore_
__double_leading_underscore
__double_leading_and_trailing_underscore__
```

PEP8: Pojmenování – základní doporučení

- proměnné, funkce, moduly: `lowercase`, příp. `lower_case_with_underscores`
- konstanty: `UPPERCASE`
- třídy: `CapitalizedWords`

Pojmenování: další rady

- jednopísmenné proměnné – jen lokální pomocné proměnné, nejlépe s konvenčním významem:
  - `n` – počet prvků (např. délka seznamu)
  - `i`, `j` – index v cyklu
  - `x`, `y` – souřadnice
- nepoužívat: `l`, `O`, `I` (snadná záměna s jinými znaky)
- angličtina, ASCII kompatibilita

PEP8: Komentáře

Komentář, který protiřečí kódu, je horší než žádný komentář.

- „inline“ komentáře používat výjimečně, nekomentovat zřejmé věci
  - Ne: `x = x + 1 # Increment x`
- doporučení ke stylu psaní komentářů (# následované jednou mezerou)

Vývojová prostředí, editory

IDLE dostatečný pro jednoduché příklady, do budoucna chcete něco lepšího. . .

žádoucí vlastnosti editoru:

- syntax highlighting
- odsazování, párování závorek
- autocomplete, suggest
- PEP8 kontrola
- podpora ladění
- podpora refaktorizace
- . . .

Vývojová prostředí, editory

příklady různých typů editorů:

- **IDLE** základní editor používaný v tomto kurzu
- **emacs, vi** (+příkazová řádka) obecné editory, příp. se specifickou konfigurací
- **pyCharm** „silný“ editor speciálně pro Python, vhodné obzvlášť pro velké projekty
- **ipython, jupyter** interaktivní použití (prolínání programu a výsledků), v prohlížeči

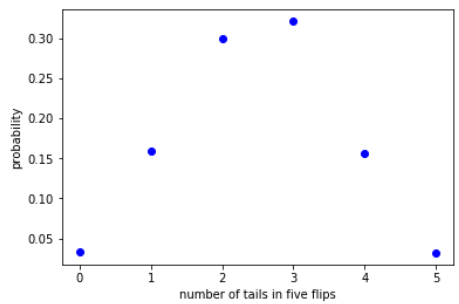
In [14]:
```
rnd.seed(33)
dicethrow = rnd.randint(1, 6 + 1, 100)
side = np.zeros(6, dtype='int')
for i in range(6):
    side[i] = np.count_nonzero(dicethrow == i + 1)
    print('number of times', i + 1, 'is', side[i])
print('total number of throws ', sum(side))
```

```
number of times 1 is 17
number of times 2 is 17
number of times 3 is 15
number of times 4 is 24
number of times 5 is 19
number of times 6 is 8
total number of throws  100
```

Back to Exercise 1

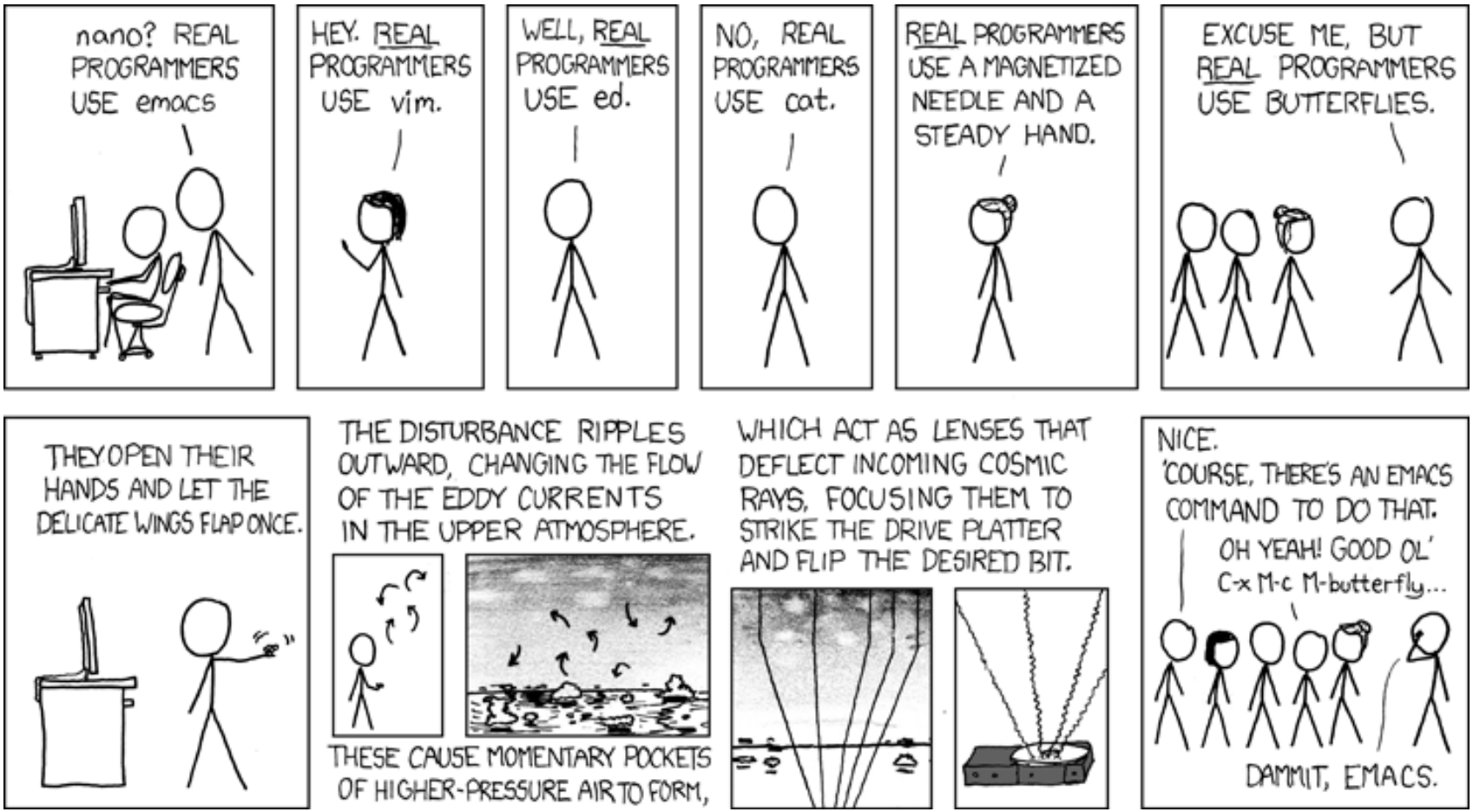
Answers to Exercise 2

In [15]:
```
N = 1000
tails = np.sum(rnd.randint(0, 1 + 1, (5, 1000)), axis=0)
counttails = np.zeros(6)
for i in range(6):
    counttails[i] = np.count_nonzero(tails == i)
plt.plot(range(0, 6), counttails / N, 'bo')
plt.xlabel('number of tails in five flips')
plt.ylabel('probability');
```



http://nbviewer.jupyter.org/github/mbakker7/exploratory_computing_with_python/blob/master/notebook9_discrete_random_variables/py_exploratory_comp_9_sol.ipynb

xkcd: Real Programmers



https://xkcd.com/378/

Správa verzí

- naivní přístup:
  - `myprogram.py`, `myprogram2.py`, `myprogram_oct_24.py`
  - `myprogram_zaloha.py`, `myprogram_pokus.py`
  - `myprogram_final.py`, `myprogram_really_final.py`
- sofistikovanější přístup – „version control“
  - automatizavaná správa verzí
  - podpora týmové práce
  - mnoho různých řešení: `git`, `cvs`, `svn`, ...

Git

- současné populární řešení
- distributed revision control
- GitHub – repozitář, (primárně) veřejné projekty
- `gitlab.fi.muni.cz` – repozitář na FI, umožňuje snadno vytvářet soukromé projekty

Python

další témata pro praktické použití:

- funkcionální prvky: map, filter, lambda funkce, ...
- obecnější předávání parametrů, využití objektů, práce se soubory, ...
- iterátory, výjimky, dekorátory, ...
- využívání modulů

Python moduly

užitečné moduly v základní distribuci

- **math**: matematické funkce
- **random**: náhodná čísla
- **sys**: „systémové“ funkce a proměnné
- **os**: spolupráce s operačním systémem
- **re**: regulární výrazy
- **datetime**: práce s časem
- **json**: práce se soubory ve formátu JSON

Python knihovny, frameworky

Příklady známých „externích“ knihoven:

- **Django**: webový framework
- **NumPy, SymPy, SciPy**: efektivní numerické výpočty, symbolické výpočty, vědecké výpočty, statistika
- **Pandas**: práce s daty (především „tabulkovými“), SQL-like operace
- **matplotlib**: tvorba grafů, vizualizace
- **pygame**: vývoj her
- **scrapy**: „scrapování“ dat z webu
- **Tensorflow**: strojové učení, deep learning

Praktický příklad vývoje v Pythonu

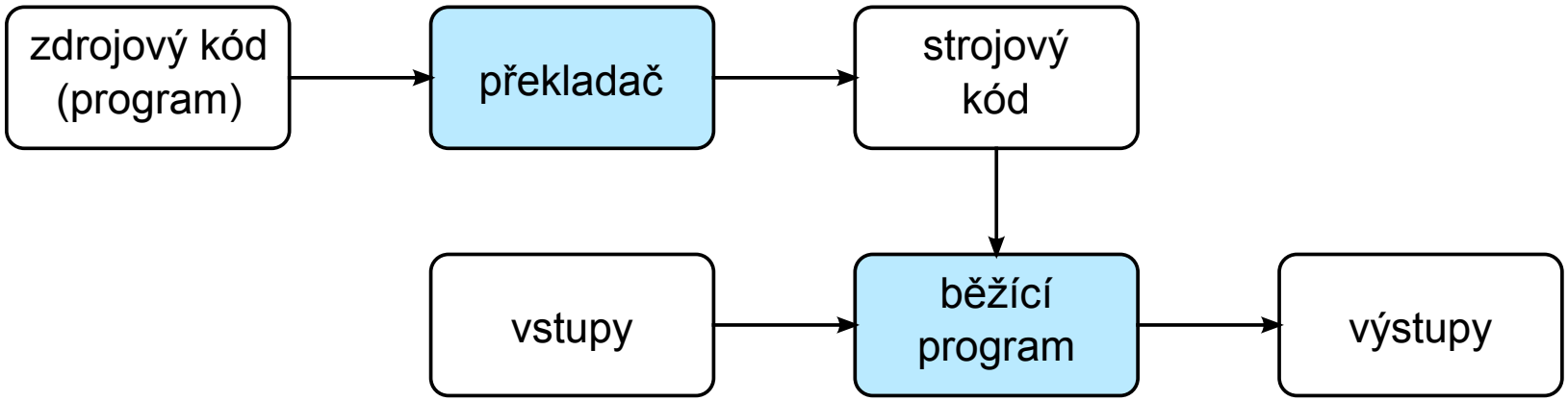
- vstup: data z výukového systému (CSV soubory)
- výstup: analýzy obtížnosti, popularity (reporty složené z HTML tabulek a grafů)
- nástroje: PyCharm, git
- Python knihovny:
  - `os`, `sys`, `argparse` – práce se soubory a příkazovou řádkou
  - `json`, `pandas` – načítání, ukládání dat
  - `math`, `re`, `datetime`, `numpy` – výpočty, transformace dat
  - `matplotlib`, `seaborn`, `plotly` – vykreslování grafů

Programovací jazyky

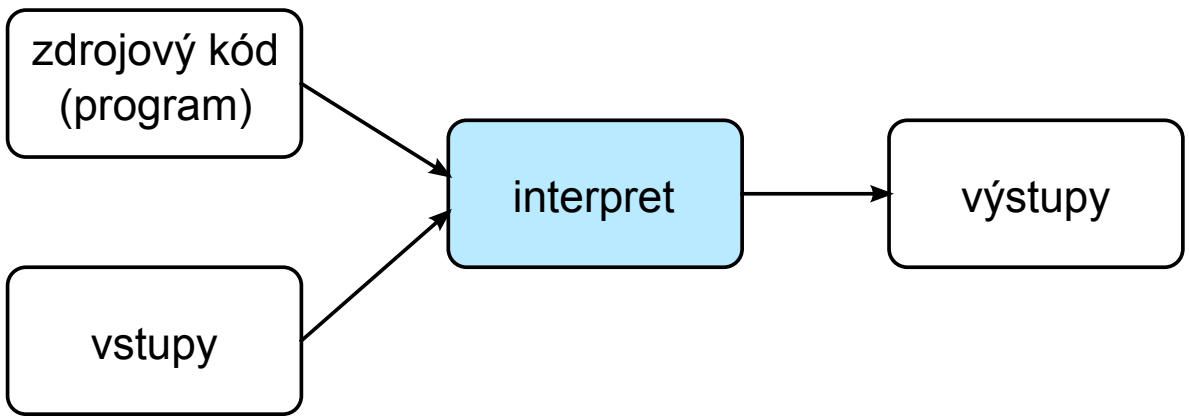
- přehled
- klasifikace, vlastnosti
- historie
- způsoby užití
- neseriózní postřehy

Interpretace, kompilace

kompilovaný program



interpretovaný program



Programovací jazyky: klasifikace I

nízko-úrovňové

- kompilované
- nutnost řešit specifika konkrétního systému
- explicitní práce s pamětí
- náročnější vývoj (nízká efektivita práce)
- vysoká efektivita programu

vysoko-úrovňové

- interpretované
- nezávislé na konkrétním systému
- využití abstraktních datových typů
- snadnější vývoj (vysoká efektivita práce)
- nižší efektivita programu

nikoliv dvě kategorie, ale plynulý přechod; zjednodušeno

Programovací jazyky: klasifikace II

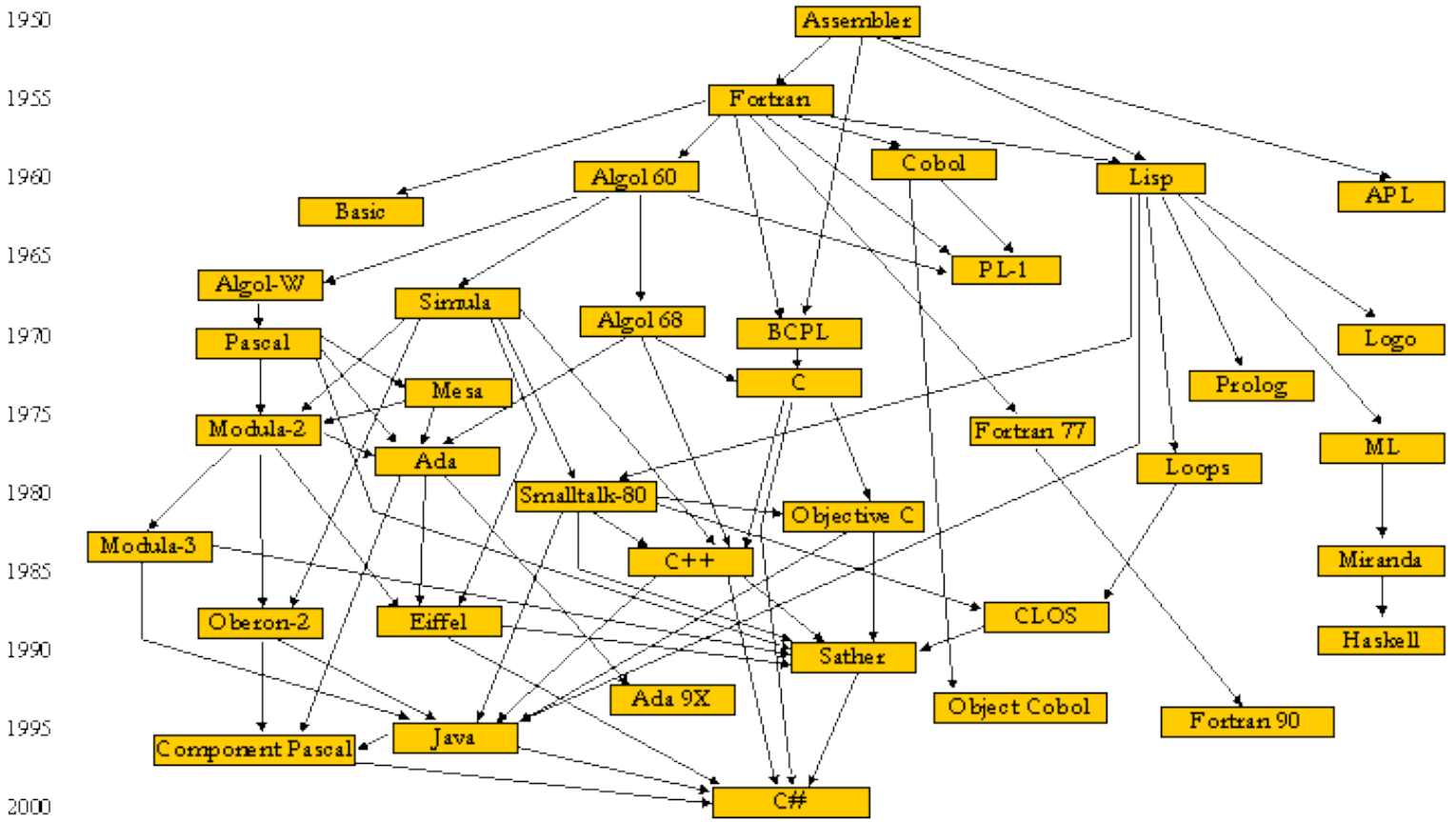
zjednodušená klasifikace a použití

nízko-úrovňové	C, FORTRAN, ...
	vestavěné systémy, rychlé výpočty
objektové	C++, Java, C#, ...
	klasické aplikace, rozsáhlé systémy
skriptovací	Python, PHP, JavaScript, Perl, ...
	programování pro web, skriptování, prototypy
deklarativní	Prolog, LISP, Haskell, ...
	umělá inteligence

Jazyky se liší v práci s typy

https://xkcd.com/1537/

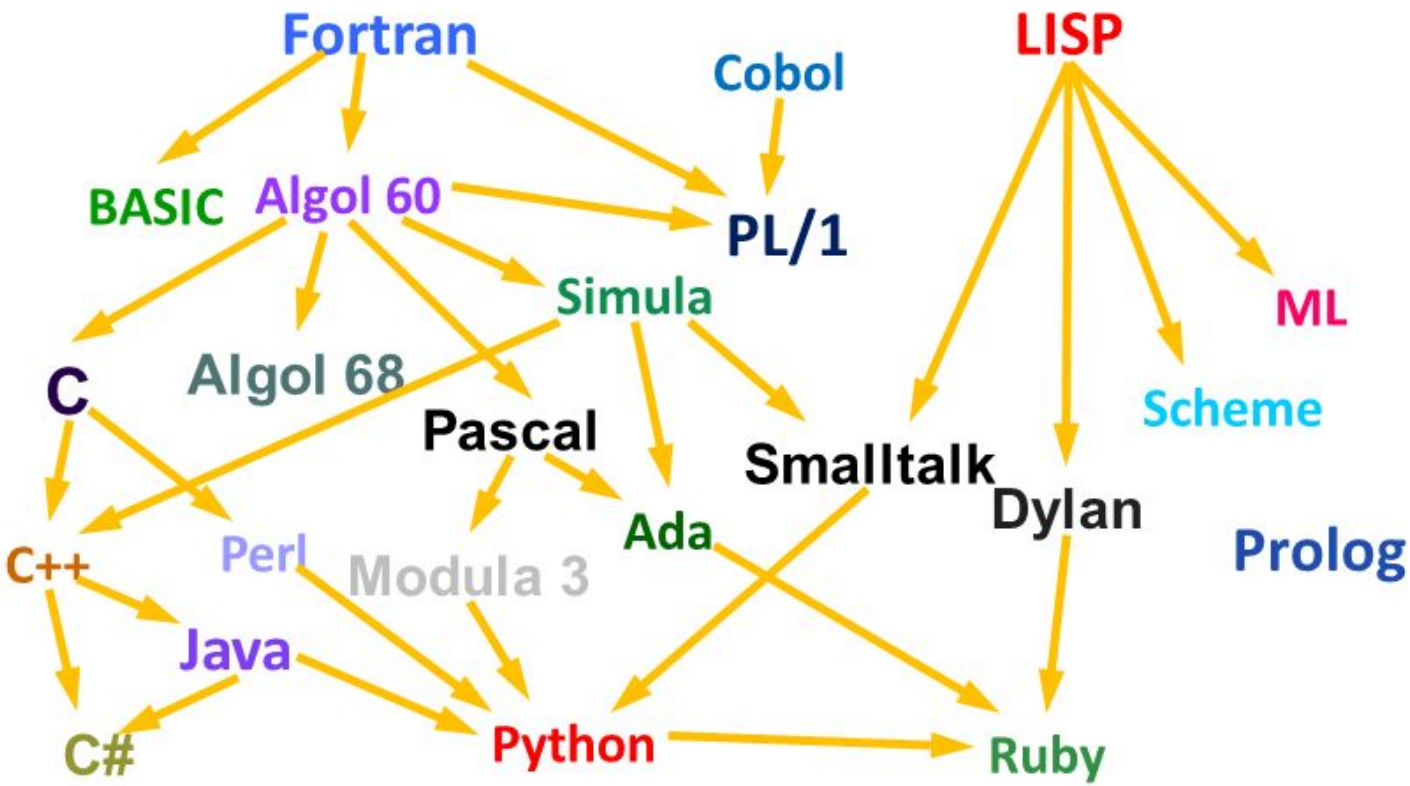
Programming Language Family Tree



(C) Fachhochschule Aargau für Technik, Wirtschaft und Gestaltung
Nordwestschweiz

A family tree of languages

Some of the 2400 + programming languages



Historie programovacích jazyků

prapočátky:

- 19. století: Charles Babbage, Ada Lovelace, děrné štítky, ...
- 30. léta: teoretické základy programování, Turingův stroj, lambda kalkul (Alonzo Church)
- 40. léta: první počítače, strojový kód, asembler

Historie programovacích jazyků

50. a 60. léta: první vysokoúrovňové jazyky (v některých aplikacích přežívají dodnes)

- ALGOL
- COBOL
- FORTRAN – vědecko-technické výpočty (užíván stále)
- BASIC

BASIC

- „jazyk pro začátečníky“
- Beginner's All-purpose Symbolic Instruction Code
- rozšířen v 70. a 80. letech na „mikropočítačích“
- *výborný jazyk pro vytvoření špatných programátorských návyků*

BASIC

```
10 INPUT "What is your name: "; U$
20 PRINT "Hello "; U$
30 INPUT "How many stars do you want: "; N
40 S$ = ""
50 FOR I = 1 TO N
60 S$ = S$ + "*"
70 NEXT I
80 PRINT S$
90 INPUT "Do you want more stars? "; A$
100 IF LEN(A$) = 0 THEN GOTO 90
110 A$ = LEFT$(A$, 1)
120 IF A$ = "Y" OR A$ = "y" THEN GOTO 30
130 PRINT "Goodbye "; U$
140 END
```

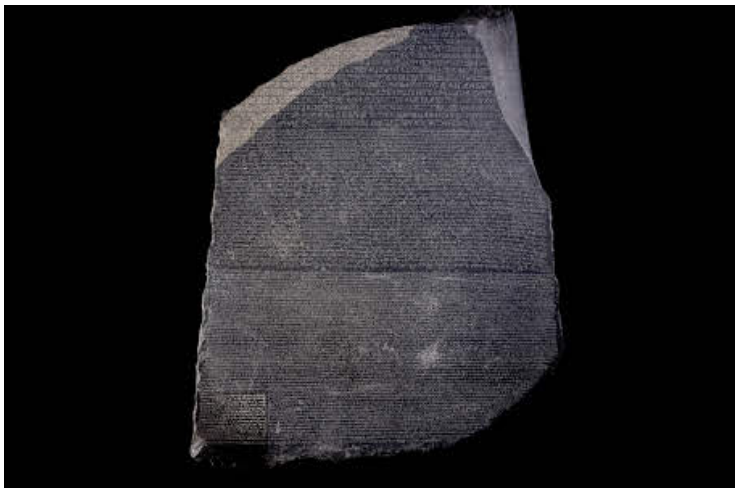
https://en.wikipedia.org/wiki/BASIC

Historie programovacích jazyků

- 70. léta
  - rozvoj základních paradigmat (imperativní, objektové, funkcionální, logické)
  - C, Pascal, Prolog
- 80. a 90. léta
  - další rozvoj jazyků, specializace, nové prvky související např. s nástupem internetu
  - C++, Perl, Haskell, Ruby, R, Java, JavaScript, PHP
- současnost
  - nové verze jazyků
  - vznik nových jazyků: Go, Dart, Kotlin, Julia

Rosseta Code

- `rosettacode.org`
- stejné problémy řešené v mnoha programovacích jazycích



If programming languages were ...

- weapons

  https://9gag.com/gag/anXEbe0

- religions

  http://blog.aegisub.org/2008/12/if-programming-languages-were-religions.html

- boats

  http://compsci.ca/blog/if-a-programming-language-was-a-boat/

- vehicles

  http://crashworks.org/if_programming_languages_were_vehicles/

Přehled programovacích jazyků

důležitý aspekt přehledu: různé jazyky mají různé rysy, (ne)výhody a aplikační domény

nedůležitý aspekt přehledu: volba citátů a přirovnání (značně subjektivní, pro zpestření, ...)

C

C would be Judaism – it's old and restrictive, but most of the world is familiar with its laws and respects them. The catch is, you can't convert into it – you're either into it from the start, or you will think that it's insanity. Also, when things go wrong, many people are willing to blame the problems of the world on it.

—

C is a nuclear submarine. The instructions are probably in a foreign language, but all of the hardware itself is optimized for performance.

C: užití

- nízkoúrovňové programování
- „blízko hardwaru“
- optimalizace rychlosti výpočtu

C syntax

mnoho jazyků staví na syntaxi C

základní rozdíly oproti Pythonu:

- vyznačování bloků kódu, (ne)významnost bílých znaků
- explicitně typovaný jazyk

C syntax: ukázka ciferný součet

```c
#include <stdio.h>
int SumDigits(unsigned long long n,
              const int base) {
    int sum = 0;
    for (; n; n /= base)
        sum += n % base;
    return sum;
}
int main() {
    printf("%d %d %d\n",
        SumDigits(1, 10),
        SumDigits(12345, 10),
        SumDigits(123045, 10));
    return 0;
}
```

Objektové jazyky odvozené od C

C++, C#, Java, ...

- kompilované (Java – bytecode)
- (většinou) explicitně typované
- typicky „silně objektové“
- vhodné pro „velké projekty“

C++

C++ is a set of nunchuks, powerful and impressive when wielded but takes many years of pain to master and often you probably wish you were using something else.

Java

Java is a cargo ship. It's very bulky. It's very enterprise∼y. Though it can also carry a lot of weight. Will carry a project, but not very fun to drive.

Java: Hello World

```
public class HelloWorld
{
 public static void main(String[] args)
 {
  System.out.println("Hello world!");
 }
}
```

Interpretované, „skriptovací“ jazyky

- JavaScript
- Python
- Perl
- PHP
- Ruby

typické užití: vývoj webu (front-end, back-end), zpracování dat, skriptování, prototypování, ...

Eval

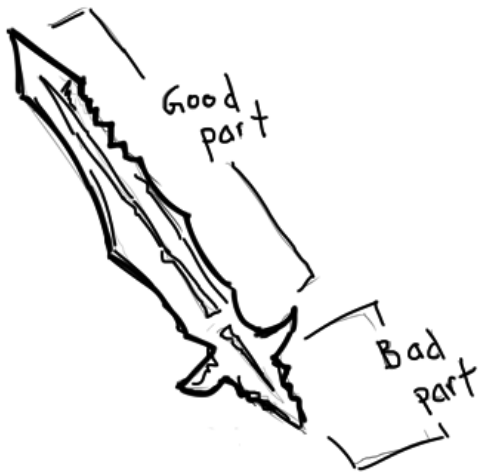
interpretované jazyky flexibilnější než kompilované

typická ukázka: příkaz `eval`

- „vyhodnocení výrazu v řetězci“
- může usnadnit práci
- ale nebezpečné (zejména nad uživatelským vstupem)

JavaScript

JavaScript is a sword without a hilt.



JavaScript

- i přes podobnost názvu nemá s Javou mnoho společného
- interpretovaný jazyk
- „jazyk webového front-endu“

PHP

PHP is a bamboo raft. A series of hacks held together by string. Still keeps afloat though.

Ruby

Ruby is difficult to describe. It's sleek, sexy, and very fun to drive. Here's a picture. Very trendy.

Perl

Perl would be Voodoo – An incomprehensible series of arcane incantations that involve the blood of goats and permanently corrupt your soul. Often used when your boss requires you to do an urgent task at 21:00 on friday night.

—

Perl used to serve the same purpose as Python, but now only bearded ex-hippies use it.

Deklarativní jazyky

- imperativní programování: program je posloupnost instrukcí („jak“ má počítač počítat)
- deklarativní programování: program je popis toho, co se má udělat

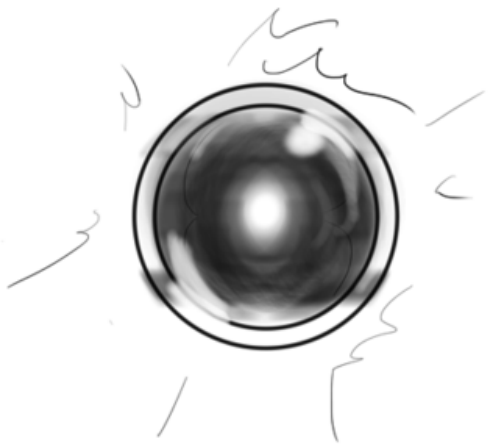
Deklarativní jazyky

- logické programování: Prolog
- funkcionální programování: Lisp, Haskell

typické užití: výpočty, „výuka principů, které využijete jinde“ (funkcionální prvky dnes v mnoha dalších jazycích), umělá inteligence (přesněji GOFAI $\sim$ *good old fashioned artificial intelligence*)

Prolog

Prolog is an AI weapon, you tell it what to do, which it does but then it also builds some terminators to go back in time and kill your mom.



Prolog

```
mother_child(trude, sally).

father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

Lisp

Lisp is a shiv which comes in many forms. Anyone who uses this is probably crazy and dangerous.

LISP

Lisp: ideální jazyk pro milovníky závorek.

```
(defun factorial (n)
(if (= n 0) 1
    (* n (factorial (- n 1)))))
```

Haskell

https://xkcd.com/1312/

Haskell

```
bsort :: Ord a => [a] -> [a]
bsort s = case _bsort s of
                t | t == s    -> t
                  | otherwise -> bsort t
 where _bsort (x:x2:xs) | x > x2    = x2:(_bsort (x:xs)
                        | otherwise = x:(_bsort (x2:xs)
       _bsort s = s
```

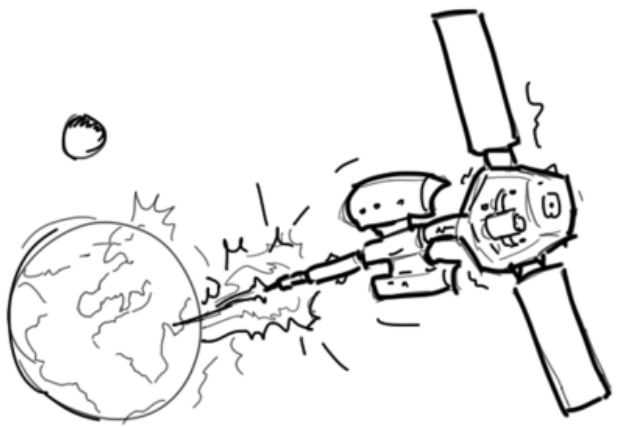
Matematický software

nástroje vyvinuté primárně jako „matematický software“, ale obsahují obecný programovací jazyk

- Mathematica
- MATLAB
- R

Mathematica

Mathematica is a low earth orbit projectile cannon, it could probably do amazing things if only anyone could actually afford one.



Esoterické programovací jazyky

„karikatury“ programovacích jazyků – dobrá karikatura umožňuje uvědomit si důležité aspekty originálu

- Brainfuck
- AsciiDots
- Whitespace
- LOLCODE
- OSTRAJava
- ...

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.+++++++..+++.>++.
<<++++++++++++++.>.+++.------.--------.>+.>.
```

```
/#$<.
*-[+]
\#1/
```

```
HAI
CAN HAS STDIO?
VISIBLE "HAI WORLD!"
KTHXBYE
```

```
banik pyco
tryda Ostrava{
    rynek(){
        Konzola.pravit("Toz vitaj") pyco
    }
}
fajront pyco
```

Jaký programovací jazyk je nejlepší?

Jaký programovací jazyk je nejlepší?

- Jaký přirozený jazyk je nejlepší? (čeština? angličtina? portugalština? ...)
- Jaký typ mapy je nejlepší? (automapa? turistická mapa? mapa města?)

Nesmyslné otázky, záleží na účelu a situaci.

Jaký programovací jazyk je nejlepší?

- neexistuje „univerzální“ jazyk, každý má své (ne)výhody
- lepší otázka: „Jaký jazyk je nejlepší pro danou situaci?“
  - problém, který řešíme
  - tým, který problém řeší
  - „legacy code“
- vyplatí se umět různé jazyky

Popularita jazyků

neexistuje nejlepší jazyk, ale také nejsou všechny jazyky stejně užitečné a rozšířené...

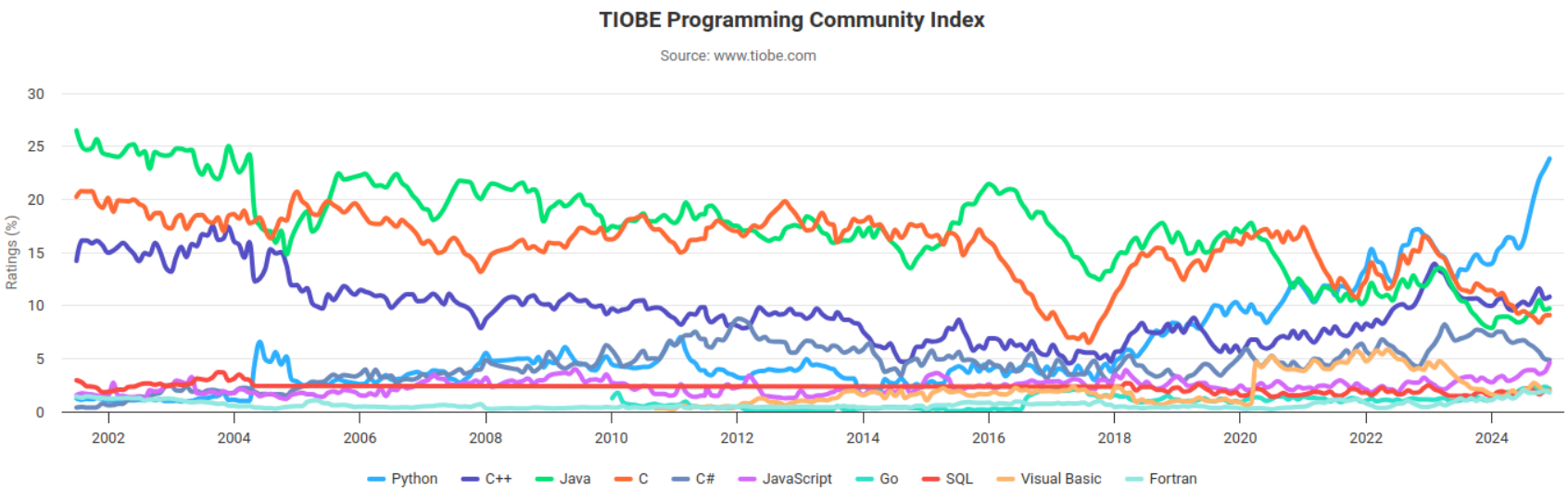
popularita těžko měřitelná, existuje řada pokusů různými metrikami:

- vyhledávání na webu
- počty knížek o jazyku
- výskyty v inzerátech
- dotazy na StackOverflow (a podobných stránkách)
- projekty na GitHubu (a podobných repozitářích)

Popularita jazyků

na vršku se vesměs vyskytují (abecedně): **C, C++, Java, JavaScript, Python**

konkrétní příklad indexu popularity:



Co dál?

navazující předměty na FI:

- **IB114 Úvod do programování a algoritmizace II**
- algoritmy, datové struktury (Python): IB002
- objektově orientované programování (Java, C++): PB161, PB162
- nízkoúrovňové programování (C): PB071
- principy prog. jazyků: PB006
- softwarové inženýrství: PB007

Závěrečné přání

Ať vás programování baví!