Úvod, organizace, základy Pythonu

IB111 Základy programování

Tomáš Vojnar (na základě slajdů Nikoly Beneše)

18. 9. 2025

Most of you are familiar with the virtues of a programmer. There are three, of course: laziness, impatience, and hubris. (Larry Wall)

https://thethreevirtues.com/

Cíle předmětu

- Zvládnutí základních programátorských konstrukcí (proměnné, funkce, větvení, cykly, vstup a výstup, rekurze).
- Seznámení se se základními datovými typy a strukturami (čísla, řetězce, seznamy, vícerozměrná pole, slovníky, základy vlastních datových struktur).
- Obecné principy použitelné v řadě programovacích jazyků.
- Kultura programování, dodržování konvencí.
- Úvod do programátorského a algoritmického stylu myšlení.
- Dostatečný základ pro další programovací a algoritmické předměty.

Předpoklady

- Základní počítačová gramotnost.
- Středoškolská matematika (faktoriál, prvočíslo, logaritmus, …).
- Logické spojky (konjunkce `and`, disjunkce `or`, negace `not`).

- Nepředpokládají se programátorské zkušenosti.
- Náročnost předmětu ovšem na vstupních dovednostech závisí:
  - zejména na schopnosti abstraktního uvažování.

Programovací jazyk Python

Základy programování, ne Programování v Pythonu!

- Používáme *omezenou část* jazyka Python:
  - jen určité povolené konstrukce/funkce (dle kapitol),
  - cokoli napíšeme, bude korektní Python (verze 3.10 nebo vyšší).
- Cílem je naučit se obecné koncepty programování,
  ne detailní zvládnutí jazyka Python.
- *Některá* specifika Pythonu součástí látky, byť ne hlavním cílem.
  - Nutné pro pochopení vytvářených programů.
  - Východisko pro srovnání s později probíranými jazyky.
  - Některá srovnání s jinými jazyky budou občas uvedena.

Programátorská kultura

Programy by měly být:

- korektní – mají dělat to, co od nich chceme;
- efektivní – mají fungovat rychle, nevyžadovat mnoho zdrojů;
- čitelné – měl by je být schopen přečíst jiný programátor.

Všechny tyto tři body jsou podstatné.

Programátorská kultura

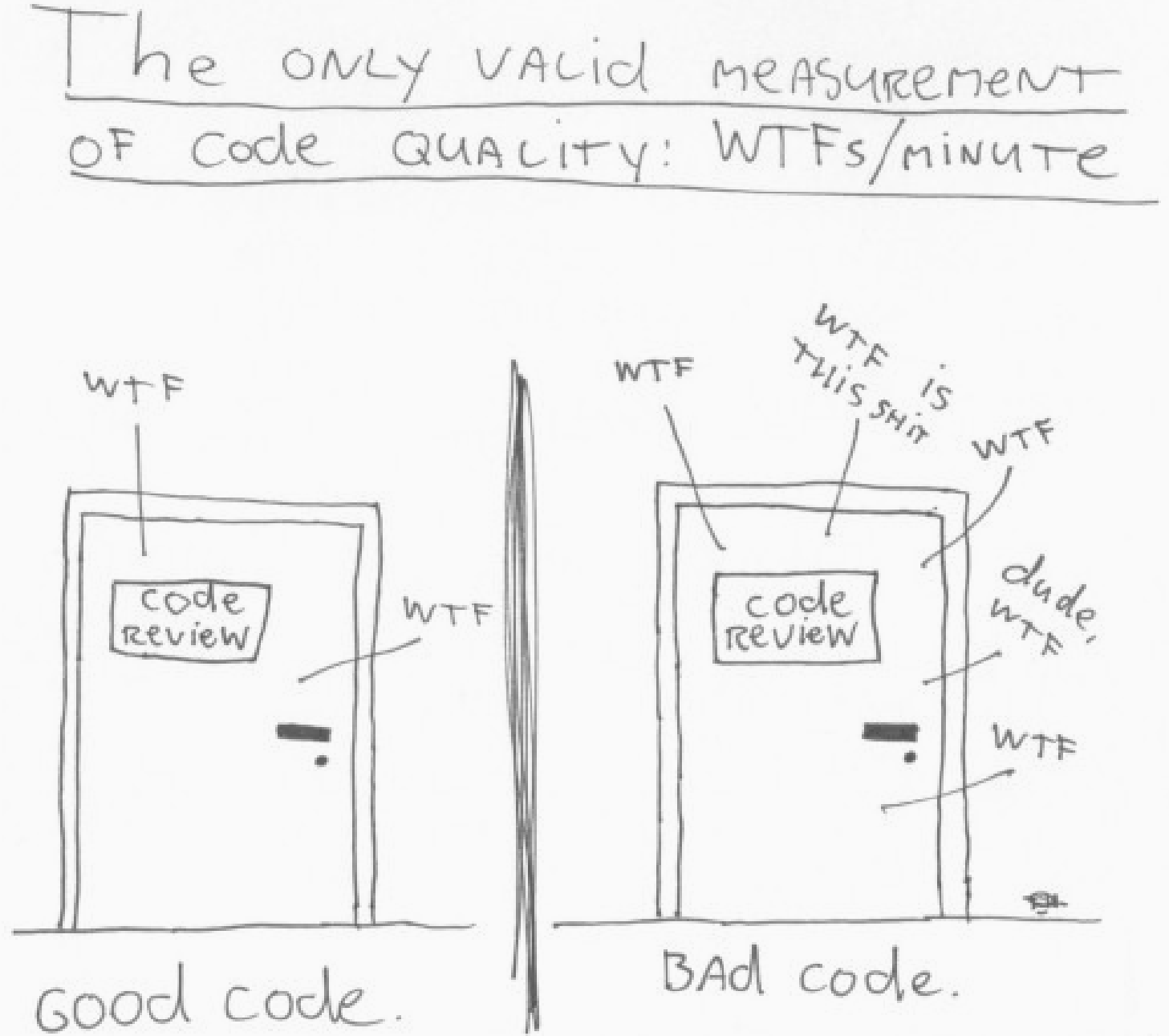
Programy by měly být:

- korektní – mají dělat to, co od nich chceme;
- efektivní – mají fungovat rychle, nevyžadovat mnoho zdrojů;
- čitelné – měl by je být schopen přečíst jiný programátor.

Všechny tyto tři body jsou podstatné.

Na čitelnost programů budeme klást důraz:

- názvy proměnných, podprogramů/funkcí,
- rozdělení větších programů do logických celků,
- dokumentace, komentáře,
- omezení duplikace kódu,
- ...


The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE
WTF
code review
WTF
Good code.
WTF
WTF IS THIS SHIT
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

Organizace předmětu

https://is.muni.cz/auth/el/fi/podzim2025/IB111/um/

(IS → Student → IB111 → Studijní materiály → Učební materiály)

- `text/ib111.seminar.pdf` (nebo `.html`)

Organizace předmětu

Bodování po blocích

- Bloky 1–3 jsou vždy čtyři týdny; čtvrtý blok je zkouška.
- Nutné minimum 50 bodů z každého bloku.
- Účast na cvičeních, aktivita na cvičeních:
  - na konci každého bloku vnitrosemestrální test
    (příprava na zkoušku: programování i teorie – bez internetu).
- *Týdenní přípravy* – odevzdávají se do soboty před cvičením.
- *Domácí úkoly* – dle harmonogramu.
- *Zkouška* – programování i teorie – bez internetu.
- Pamatujte: cílem je naučit se programovat,
  body jsou jen motivační prostředek.

Obecně k programování

- Na programování si vyhraďte čas, kdy vás nebude nic rušit
  - ani sociální sítě!
- Nejdříve si promyslete, co a jak budete řešit,
  - klidně s pomocí papíru a tužky.

Stručný návod k úspěšnému absolvování předmětu

Obecně k programování

- Na programování si vyhraďte čas, kdy vás nebude nic rušit
  - ani sociální sítě!
- Nejdříve si promyslete, co a jak budete řešit,
  - klidně s pomocí papíru a tužky.

Týdenní režim

- Podívejte se na přednášku.
- Přečtěte si úvodní text v příslušné kapitole.
  - včetně řešených ukázek (`d`).
- Rozcvičte se na elementárních příkladech (`e`).
- Samostatně řešte přípravy (`p`).
  - Nemusíte zdaleka vyřešit všechny;
  - pokud Vám nějaká nejde, zkuste jinou.
  - O přípravách bude prostor k diskusi na příštím cvičení.

Stručný návod k úspěšnému absolvování předmětu

Sady domácích úkolů

- Začněte je řešit včas,
- řešte je samostatně.

Stručný návod k úspěšnému absolvování předmětu

Sady domácích úkolů

- Začněte je řešit včas,
- řešte je samostatně.

Co znamená „samostatně“?

- Bez pomoci jiných lidí,
- bez pomoci „umělé inteligence“,
- bez opisování kódů z internetu, knih a odjinud.
  - Je rozdíl mezi
    „pochopil jsem myšlenku a pak ji samostatně použiju“
    a *„vezmu kód odjinud a zkusím ho nějak upravit“*.

- Opisováním se nic nenaučíte,

Stručný návod k úspěšnému absolvování předmětu

Sady domácích úkolů

- Začněte je řešit včas,
- řešte je samostatně.

Co znamená „samostatně“?

- Bez pomoci jiných lidí,
- bez pomoci „umělé inteligence“,
- bez opisování kódů z internetu, knih a odjinud.
  - Je rozdíl mezi
    „pochopil jsem myšlenku a pak ji samostatně použiju“
    a *„vezmu kód odjinud a zkusím ho nějak upravit“*.

- Opisováním se nic nenaučíte,
- opisování citelně postihujeme (obě strany).

Algoritmus

- Návod/postup, jak „mechanicky“ vyřešit určitý typ úlohy/problému.

Algoritmy

Algoritmus

- Návod/postup, jak „mechanicky“ vyřešit určitý typ úlohy/problému.

Příklady algoritmů

- kuchařský recept – pokud je přesný,
- návod k sestavení nábytku,
- Euklidův algoritmus (největší společný dělitel dvou čísel),
- rozklad přirozeného čísla na součin prvočísel,
- řešení Sudoku,
- nalezení nejkratší cesty mezi dvěma městy,
- řešení nějaké logické úlohy, ...

Žádoucí vlastnosti algoritmů

- Jasný vstup a výstup.
- Obecnost:
  - není jen pro omezenou sadu instancí,
  - „spočítat 7×5“ vs. „spočítat součin dvou zadaných čísel“.
- Determinovanost:
  - každý krok je jasně a přesně definován,
  - v každé situaci je jednoznačné, jak dál postupovat.
    - Není vždy nutné – *pravděpodobnostní algoritmy*.
- Konečnost:
  - skončí po konečném počtu kroků.
    - Ne u tzv. *reaktivních systémů* – běží do povelu „ukonči se“.
- Elementárnost kroků:
  - každý krok je dostatečně jednoduchý.
- Efektivita:
  - časová náročnost, prostorová náročnost, ...

Program

- Algoritmus zapsaný v programovacím jazyce.
  - Ten definuje sadu základních instrukcí, jež můžeme použít.
  - Více o různých druzích jazyků koncem semestru.

Programy a programování

Program

- Algoritmus zapsaný v programovacím jazyce.
  - Ten definuje sadu základních instrukcí, jež můžeme použít.
  - Více o různých druzích jazyků koncem semestru.

Jak se učit programování?

- Tréninkem.
- Programování je dovednost; dovednost se získává praxí.
- Ukážeme Vám cestu, ale jít po ní už musíte sami.
  - Přednášky, cvičení i domácí úkoly „ukazují cestu“.
  - Je vhodné si občas jen tak zkusit něco naprogramovat:
  - k dispozici ukázky z přednášek – zkoušejte modifikovat.

Selhání programů

- Syntaktické chyby: program nelze přeložit a spustit.
- Neočekávané ukončení běhu z důvodů
  - externích – málo paměti, chyba HW, výpadek napájení, … či
  - interních – dělení nulou, práce s neexistující položkou dat, …
- Neočekávané neukončení běhu – „zacyklení“ se.
  - Něco jiného než *očekávané* cyklení u reaktivních systémů – reaktivní systémy zde ale nebudeme běžně uvažovat.
- Chybný výstup s ohledem na specifikaci žádoucího chování.
  - Specifikace může omezit očekávané vstupy programu: pro ostatní nemusí být chování definováno (viz dále).

Selhání programů

- Syntaktické chyby: program nelze přeložit a spustit.
- Neočekávané ukončení běhu z důvodů
  - externích – málo paměti, chyba HW, výpadek napájení, … či
  - interních – dělení nulou, práce s neexistující položkou dat, …
- Neočekávané neukončení běhu – „zacyklení“ se.
  - Něco jiného než *očekávané* cyklení u reaktivních systémů – reaktivní systémy zde ale nebudeme běžně uvažovat.
- Chybný výstup s ohledem na specifikaci žádoucího chování.
  - Specifikace může omezit očekávané vstupy programu: pro ostatní nemusí být chování definováno (viz dále).

Pravidla předmětu: *Nebude-li řečeno jinak, na přednáškách, v úkolech, ve zkouškách uvažujeme POUZE programy bez syntaktických chyb, spouštěné pro očekávané vstupy, přičemž nedochází k neočekávanému ukončení ani neukončení jejich běhu.*

Základy Pythonu

Jak a kam psát programy

- Program v Pythonu je textový soubor („plain text“).
  - Přípona `.py`.
  - Stačí libovolný textový editor.
- Spuštění programu z příkazové řádky:
  - `python jméno_programu.py`,
    - na některých platformách `python3`.
  - Program `python` je tzv. interpret Pythonu, postupně čte řádky programu a vykonává je.
- Spuštění interpretu bez souboru s programem:
  - tzv. interaktivní shell nebo též REPL (read-eval-print loop),
  - užitečné pro jednoduché zkoušení příkazů apod.
- Interaktivní mód se spuštěním programu
  - `python -i jméno_programu.py`.
- Užitečný web pro krokování programů:
  http://pythontutor.com/

Vývojová prostředí

IDE (integrated development environment)

- Usnadňují psaní programů,
  - automatické doplňování názvu funkcí apod. („napovídání“),
  - snadné spouštění programů klávesovou zkratkou/z menu,
  - zvýrazňování syntaxe (barvy).

IDE pro Python

- Základní součást instalace Pythonu: IDLE.
- Existují i mnohá jiná, např. pro výuku určené Thonny.
- Doporučujeme vyzkoušet Thonny:
  - interaktivní shell ve spodní části,
  - po spuštění programu jsou v interaktivním módu k dispozici definované proměnné, funkce apod.

Obecné poznámky k syntaxi

- „Case sensitive“ – na velikosti písmen záleží,
  - `print` není totéž, co `PRINT` nebo `Print`.
- Komentáře – cokoli od znaku # po konec řádku.
  - Python je ignoruje.
  - Užitečné pro dokumentaci, poznámky k fungování programu.
- Písmena s diakritikou:
  - Python 3 umožňuje (preferujte UTF-8).
  - Raději nepoužívejte v názvech proměnných, funkcí apod.
  - Obecně raději preferujte angličtinu.
    (Proč?

Obecné poznámky k syntaxi

- „Case sensitive“ – na velikosti písmen záleží,
  - `print` není totéž, co `PRINT` nebo `Print`.
- Komentáře – cokoli od znaku # po konec řádku.
  - Python je ignoruje.
  - Užitečné pro dokumentaci, poznámky k fungování programu.
- Písmena s diakritikou:
  - Python 3 umožňuje (preferujte UTF-8).
  - Raději nepoužívejte v názvech proměnných, funkcí apod.
  - Obecně raději preferujte angličtinu.
    (Proč? Jak byste rádi četli kód po někom, kdo si proměnné pojmenoval finsky? Hebrejsky? Klingonsky?)

Omezený Python pro IB111

```
from ib111 import week_01
```

- Vyžaduje soubory `ib111.py` a `ib111.pyi` ve stejném adresáři (přibaleny ke všem příkladům ve studijních materiálech).
- Omezí výrazové prostředky Pythonu na ty, které jsou potřeba v zadaném týdnu (zadané kapitole sbírky).
- Tento řádek nemažte, nemodifikujte ani nikam nepřesouvejte.

Uložení dat

Objekt

- Abstrakce paměti.
- Úložiště/kontejner pro hodnoty.
- Objekty mají
  - identitu („adresu“ objektu),
  - typ[1] (celé číslo, řetězec, …),
  - hodnotu.
- Hodnota objektu se (povoluje-li to typ) může měnit, typ a identita objektu se po dobu jeho existence nemění.

https://docs.python.org/3/reference/datamodel.html

[1] Více o typech později.

Proměnné

- Umožňují srozumitelně pojmenovat objekty.
- Každá proměnná:
  - má jméno – ideálně smysluplné :) a
  - existuje v nějakém kontextu.
- Jméno je statický koncept v textu programu,
  kontext a proměnná existují za běhu programu.
- Kontext: spuštěný program, zavedený modul,
  běžící podprogram (resp. jedna z případně více rozběhnutých instancí podprogramu), ...
- Jméno a kontext určují identitu proměnné a nemění se.

Proměnné

- V Pythonu proměnné mohou, ale nemusí mít typ.
- Proměnná může, ale nemusí mít vazbu na objekt.
- Vazba proměnné na objekt se může v Pythonu měnit.

- Zjednodušené vyjadřování:
  - „Hodnota proměnné“: hodnota objektu, na který má daná proměnná vazbu.
  - „Stejná proměnná“ např. při ladění apod.: stejné jméno, stejný kontext, případně s různými vazbami.
  - Někdy se také mluví o „proměnné“ a myslí se tím jen její jméno.
    - Zejména pokud se bavíme o syntaxi.
    - Nutná opatrnost.

Proměnné

variables.py

- V Pythonu se proměnná vytváří (definuje) prvním přiřazením v určitém kontextu.
- Příkaz přiřazení: jméno = výraz, např. `x = 1 - y`.
- Výraz se vyhodnotí na objekt (existující nebo nově vzniklý).
- Proměnná se sváže s tímto objektem.

```
answer = 6 * 7
name = "John"

# what is the value of answer?
# what is the value of name?
# what is the value of (answer == 42)?
```

Názvy proměnných

- Posloupnost písmen, číslic a znaků _ (podtržítko).
- Víceslovné názvy `snake_case`, `CamelCase`, `mixedCase`.
- Nelze použít klíčová slova (`if`, `while`, …).
- Lze použít některé jiné prvky jazyka (názvy typů, standardních podprogramů), ale není to vůbec dobrý nápad.

Pravidla předmětu

- Všechny názvy anglicky;
- tak, aby bylo jasné, co reprezentují;
- malá písmena, více slov pomocí `snake_case` (PEP8).

Výrazy

Výraz: kombinace konstant, (jmen) proměnných, operátorů, závorek a volání funkcí.

Každý výraz má hodnotu – její výpočet popisuje.

Aritmetické operátory

- sčítání `+`, odčítání `-`, násobení `*`,
- umocňování `**` (`17 ** 4` je `83521`),
- zbytek po dělení `%` (`17 % 4` je `1`),
- celočíselné dělení `//` (`17 // 4` je `4`),
- reálné dělení `/` (`17.0 / 4.0` je `4.25`), …

- Zkrácený zápis: `x += 5` znamená `x = x + 5` (apod. pro ostatní operace).

Vestavěné funkce

- `abs`, `max`, `min`, `round`, …

Výrazy

Celá vs. reálná čísla

- Celá čísla: typ `int`, zápis bez tečky (např. `17`, `42`).
- „Reálná“ čísla: typ `float`, zápis s tečkou (např. `17.0`, `4.2`).
- Typová konverze z celého čísla na „reálné“: `float(num)`.

Dělení a zbytek

- Celočíselné dělení `//` v Pythonu zaokrouhluje *vždy dolů*:
  - `17 // 4` je `4`,
  - `-17 // 4` je `-5`.
- Zbytek po dělení: `(x // y) * y + x % y` je vždy `x`.
- Reálné dělení `/` vrací vždy číslo typu `float`.

Omezený režim pro IB111

- Celočíselné dělení `//` a zbytek `%` jen pro celá čísla.
- Reálné dělení `/` jen pro alespoň jeden operand typu `float`.

Výrazy

Porovnávání

- Rovnost `==`, nerovnost `!=`, `<`, `<=`, `>`, `>=`,
  - výsledkem je *pravdivostní hodnota*: `True` nebo `False`.
- Specialita Pythonu – řetězení: `a < b < c`,
  - jako `a < b and b < c`, ale `b` se vyhodnotí jen jednou.

Logické operace

- Kombinují logické podmínky (pravdivostní hodnoty):
  - `and`, `or`, `not`.
  - Zkrácené vyhodnocování `and` a `or`:
    nejprve levý operand, je-li výsledek jasný, nepokračuje se.

Pozor: operátory `=` a `==` mají jiný význam!

Výrazy

Porovnávání

- Rovnost `==`, nerovnost `!=`, `<`, `<=`, `>`, `>=`,
  - výsledkem je *pravdivostní hodnota*: `True` nebo `False`.
- Specialita Pythonu – řetězení: `a < b < c`,
  - jako `a < b and b < c`, ale `b` se vyhodnotí jen jednou.

Logické operace

- Kombinují logické podmínky (pravdivostní hodnoty):
  - `and`, `or`, `not`.
  - Zkrácené vyhodnocování `and` a `or`:
    nejprve levý operand, je-li výsledek jasný, nepokračuje se.

Pozor: operátory `=` a `==` mají jiný význam!

- Jednoduché `=` je přiřazení (vytváří vazbu proměnné na objekt).
- Dvojité `==` je porovnání (test, zda jsou dvě hodnoty stejné).

Pořadí vyhodnocování výrazů

- Většinou „intuitivní“:
  - operace se vyhodnocují zleva doprava,
  - $*$ má přednost před $+$ apod.
- Jste-li na pochybách:
  - podívejte se do dokumentace,
  - použijte závorky.

https://docs.python.org/3/reference/expressions.html#operator-precedence

for_example.py

- Provedení části programu opakovaně – *iterace*.
- Zpracování sekvence vstupů, procházení složitějších datových struktur, ...

Cyklus s daným počtem opakování: `for`

```
total = 0
for i in range(10):
    total = total + i
# what is the value of total here?
```

- Všimněte si, že „tělo“ cyklu v Pythonu vyznačujeme odsazením
  - standardně 4 mezery, IDE typicky odsazuje automaticky.

Poznámka: `range` může mít i více parametrů (viz učební text, příští přednášku).

while_example.py

Cyklus s podmínkou: `while`

```
x = 10000
d = 0

while x > 0:
    x = x // 10
    d += 1

# what is the value of x, d here?
```

Další příkazy pro řízení cyklů

break_continue.py

- **break** ukončí provádění cyklu.
- **continue** ukončí aktuální iteraci cyklu a přejde na další.

```
total = 0
for i in range(10):
    if i == 3:
        continue
    if i == 7:
        break
    total += i
```

Větvení

if1.py

- Provedení části programu jen v případě, že platí určitá podmínka.

```
answer = -6 * 7
if answer < 0:
    answer = -answer

# what is the value of answer here?
```

Složitější větvení

if2.py

```
password = "STANFORD"

if password == "STANFORD":
    message = "Imminent threat!"
else:
    message = "Access denied!"
```

- Je-li podmínka pravdivá, provede se blok příkazů za `if`.
- Není-li pravdivá, provede se blok příkazů za `else`.

Ještě složitější větvení

if3.py

```
x = -17
if x > 0:
    status = "kladné"
elif x < 0:
    status = "záporné"
else:
    status = "nula (není ani kladná ani záporná)"
```

- Podmínky se vyhodnocují postupně shora;
- provede se blok příkazů u první podmínky, která je pravdivá.

Pravdivostní hodnoty

conditions.py

- Podmínky nemusí nutně být jen u příkazů `if` a `while`.
- Podmínky mají pravdivostní hodnotu.
  - Pravda: `True`.
  - Nepravda: `False`.

```
answer = 6 * 7
universe_ok = answer == 42
answer_positive = answer > 0
answer_natural = answer_positive or answer == 0
```

Příklad: Řešení hádanky hrubou silou

Hádanka: Farmář chová prasata a slepice. Celkem je na dvoře 20 hlav a 56 noh. Kolik má slepic a kolik prasat?

- Jak vyřešit pro konkrétní zadání?
- Jak vyřešit pro obecné zadání (H hlav a N noh)?

Příklad: Řešení hádanky hrubou silou

Hádanka: Farmář chová prasata a slepice. Celkem je na dvoře 20 hlav a 56 noh. Kolik má slepic a kolik prasat?

- Jak vyřešit pro konkrétní zadání?
- Jak vyřešit pro obecné zadání (H hlav a N noh)?

Možné přístupy:

- „Chytré řešení“: řešení systému lineárních rovnic.
- „Hrubou silou“: vyzkoušíme všechny možnosti.

Příklad: Řešení hádanky hrubou silou

Hádanka: Farmář chová prasata a slepice. Celkem je na dvoře 20 hlav a 56 noh. Kolik má slepic a kolik prasat?

- Jak vyřešit pro konkrétní zadání?
- Jak vyřešit pro obecné zadání (H hlav a N noh)?

Možné přístupy:

- „Chytré řešení“: řešení systému lineárních rovnic.
- „Hrubou silou“: vyzkoušíme všechny možnosti.

K zamyšlení: Co kdyby farmář ještě navíc choval osminohé pavouky?

Příklad: Řešení hádanky hrubou silou

farm1.py

```
heads = 20
legs = 56
# -1: a special value to represent «no solution»
sol_hens = -1
sol_pigs = -1

for hens in range(heads + 1):
    pigs = heads - hens
    if 2 * hens + 4 * pigs == legs:
        sol_hens = hens
        sol_pigs = pigs
        break

# what is the value of sol_hens, sol_pigs?
```

Podprogramy (funkce, *subroutines*)

fun.py

- Způsob, jak rozložit program na menší logické celky.
- Úsek kódu, jenž má navíc název, parametry a návratovou hodnotu (podrobněji na další přednášce).
- V Pythonu[2] se podprogramy realizují pomocí tzv. *funkcí*.

```
def deg_to_rad(degrees):
    pi = 3.14
    return degrees / 180.0 * pi
```

Proč chceme rozkládat programy na podprogramy?

[2] A v mnoha jiných jazycích, i když ne ve všech

Podprogramy (funkce, *subroutines*)

fun.py

- Způsob, jak rozložit program na menší logické celky.
- Úsek kódu, jenž má navíc název, parametry a návratovou hodnotu (podrobněji na další přednášce).
- V Pythonu[2] se podprogramy realizují pomocí tzv. *funkcí*.

```
def deg_to_rad(degrees):
    pi = 3.14
    return degrees / 180.0 * pi
```

Proč chceme rozkládat programy na podprogramy?

- vyhneme se opakování stejného/podobného kódu,
- modularita (Lego), znovupoužitelnost,
- snazší přemýšlení o problému, „dělba práce“.

[2] A v mnoha jiných jazycích, i když ne ve všech

Příklad: Řešení hádanky s použitím funkce

farm2.py

```
def number_of_hens(heads, legs):
    for hens in range(heads + 1):
        pigs = heads - hens
        if 2 * hens + 4 * pigs == legs:
            return hens

    # a special value to represent «no solution»
    return -1

# what is the value of number_of_hens(20, 56)?
# what is the value of number_of_hens(40, 100)?
```

Knihovny

libs.py

- Předpřipravené sady funkcí, konstant, ...

- `import jméno_knihovny`
  - Funkce pak používáme takto: `jméno_knihovny.jméno_funkce(...)`.
- `from jméno_knihovny import jméno_funkce, ...`
  - Funkce pak můžeme použít přímo: `jméno_funkce(...)`.

```
from turtle import forward, done
import math

forward(150 * math.sin(30.0 / 180.0 * math.pi))
done()
```

Knihovny

libs.py

- Předpřipravené sady funkcí, konstant, ...

- `import jméno_knihovny`
  - Funkce pak používáme takto: `jméno_knihovny.jméno_funkce(...)`.
- `from jméno_knihovny import jméno_funkce, ...`
  - Funkce pak můžeme použít přímo: `jméno_funkce(...)`.

```
from turtle import forward, done
import math

forward(150 * math.sin(30.0 / 180.0 * math.pi))
done()
```

Varování: nepojmenovávejte vlastní soubory stejně jako standardní knihovny (např. `turtle`), ty Vám pak můžou přestat fungovat.

Zásady čitelnosti kódu

Obecné

- Pište tak, aby byl jasný záměr:
  - smysluplné pojmenování proměnných, funkcí apod.;
  - v případě komplikovanějšího kódu dokumentace (komentáře)
  - (ideálně ovšem psát samodokumentující se kód).
- Neopakujte se, nepište „copy&paste kód“.
- Větší kód členěte do rozumných menších celků:
  - pomocné funkce, (objekty, moduly, …).

Konkrétní

- Různé jazyky mají různou představu o „slušném kódu“.
- Různé skupiny (různá pracovní prostředí) mají různé zvyky:
  - coding styles, coding guides.
- Pro Python existuje soubor oficiálních doporučení – PEP8: https://www.python.org/dev/peps/pep-0008/.

Nástroje pro odhalování chyb (nejen) stylu

- Některá IDE umí chyby detekovat sama.
- Jiné nástroje: `pycodestyle`, `flake8`, `pylint` (příkazový řádek).
- Automatické testy domácích úkolů používají `edulint`, je vhodné se s tímto nástrojem seznámit co nejdříve:
  - viz návody v učebních materiálech.
- Později v semestru začneme používat nástroj `mypy` pro statickou analýzu typových anotací.

Automatické formátování

- Nástroje, které udělají část práce za vás.
- Někdy součást IDE.
- `autopep8`, `black`, `yapf`, ...

Používejte, co uznáte za vhodné, ale za výsledek odpovídáte Vy.

Kde hledat pomoc?

Primární zdroj – učební text:

- popis (omezeného) jazyka na začátku každé kapitoly,
- úvodní text, demonstrační příklady.

Nápověda přímo v Pythonu – příkaz `help(jméno_funkce)`.

Dokumentace na webu – https://docs.python.org/3/

Příklady k procvičení

- https://leetcode.com/
- https://www.hackerrank.com/
- https://www.codewars.com/
- https://adventofcode.com/
- ...