

PYTHON

Adam Klepáč

30. ledna 2023

Gymnázium Evolution Jižní Město

Programovací jazyky

Nejnižší forma komunikace

Strojový kód

Strojový kód je jazyk sestávající pouze ze **základních instrukcí** pro CPU.

Nejvyšší forma komunikace

Programovací jazyk

Programovací jazyk je jakýkoli jazyk, který lze **automaticky přeložit** do strojového kódu.

Nač programovací jazyky?

- Strojový kód je člověku nečitelný.

Nač programovací jazyky?

- Strojový kód je člověku nečitelný.
- Programovací jazyky se čím dál více přibližují lidské řeči.

Nač programovací jazyky?

- Strojový kód je člověku nečitelný.
- Programovací jazyky se čím dál více přibližují lidské řeči.
- Programovací jazyky jsou rozšiřitelné – umožňují přidání nových konceptů (proměnných, podmínek, ...)

Nač programovací jazyky?

- Strojový kód je člověku nečitelný.
- Programovací jazyky se čím dál více přibližují lidské řeči.
- Programovací jazyky jsou rozšiřitelné – umožňují přidání nových konceptů (proměnných, podmínek, ...)
- V programovacích jazycích lze některé běžné paměťové operace CPU automatizovat (rekurze, garbage collector, ...)

Jaká je cena?

- Překlad prog. jazyků je automatický – vzniká spousta přebytečného strojového kódu.

Jaká je cena?

- Překlad prog. jazyků je automatický – vzniká spousta přebytečného strojového kódu.
- Přebytečné instrukce zpomalují běh programu.

Jaká je cena?

- Překlad prog. jazyků je automatický – vzniká spousta přebytečného strojového kódu.
- Přebytečné instrukce zpomalují běh programu.
- Velká práce s údržbou – každá nová funkce programovací jazyka vyžaduje mnoho testování správnosti překladu do stroj. kódu

Jaká je cena?

- Překlad prog. jazyků je automatický – vzniká spousta přebytečného strojového kódu.
- Přebytečné instrukce zpomalují běh programu.
- Velká práce s údržbou – každá nová funkce programovací jazyka vyžaduje mnoho testování správnosti překladu do stroj. kódu
- V různých jazycích jsou stejné funkce psané jinak.

Typy programovacích jazyků

(1) strojový kód,

Typy programovacích jazyků

- (1) strojový kód,
- (2) assembly (jazyky symbolických adres):
 - symbolické reprezentace CPU instrukcí
 - zkratky pro běžné operace
 - žádná automatizace

Typy programovacích jazyků

- (1) strojový kód,
- (2) assembly (jazyky symbolických adres):
 - symbolické reprezentace CPU instrukcí
 - zkratky pro běžné operace
 - žádná automatizace
- (3) high-level (vysokoúrovňové) programovací jazyky:
 - pokročilé řídicí sekvence – proměnné, podmínky, cykly, ...
 - automatická správa běhu – procedury, funkce
 - částečně automatická správa paměti – pole, třídy, ...

Kam patří Python?

- Python je high-level programovací jazyk.

Kam patří Python?

- Python je high-level programovací jazyk.
- Python \rightarrow C \rightarrow (Assembly \rightarrow) stroj. kód

Kam patří Python?

- Python je high-level programovací jazyk.
- Python \rightarrow C \rightarrow (Assembly \rightarrow) stroj. kód
- Python je **interpretovaný** (vs. kompilovaný) programovací jazyk – to znamená, že počítač překládá Python za běhu programu.

Kam patří Python?

- Python je high-level programovací jazyk.
- Python \rightarrow C \rightarrow (Assembly \rightarrow) stroj. kód
- Python je **interpretovaný** (vs. kompilovaný) programovací jazyk – to znamená, že počítač překládá Python za běhu programu.
- Python má automatickou správu paměti a dokonce vás ani nenutí typovat.

I. Programování v Pythonu

Datové typy

Co to je?

Datový typ

Datový typ je doslova typ (forma, podoba, ...) informace uložené v paměti počítače.

Co to je?

Datový typ

Datový typ je doslova typ (forma, podoba, ...) informace uložené v paměti počítače.

- Narozdíl od pseudokódu, v programovacích jazycích musíte kromě názvu proměnné uvádět i její typ.

Co to je?

Datový typ

Datový typ je doslova typ (forma, podoba, ...) informace uložené v paměti počítače.

- Narozdíl od pseudokódu, v programovacích jazycích musíte kromě názvu proměnné uvádět i její typ.
- Základní typy v Pythonu jsou `int`, `float`, `str`, `set`, `list`, `tuple`, `dict`

Měnné vs. neměnné

- Python rozlišuje mezi **měnnými** (mutable) a **neměnnými** (immutable) datovými typy.

Měnné vs. neměnné

- Python rozlišuje mezi **měnnými** (mutable) a **neměnnými** (immutable) datovými typy.
- Do struktury měnných typů (seznamy, slovníky, ...) můžete zasahovat během programu, ale do struktury neměnných (čísla, slova, ...) nikoliv.

Datové typy

Číselné typy

Celá čísla

Datový typ `int`

Zkratkou `int` (z angl. `integer`) Python označuje typ celých čísel, tj. čísel bez desetinné části.

Celá čísla

Python umí následující operace na celých číslech.

- součet (+);
- rozdíl (-);
- součin (*);

Celá čísla

Python umí následující operace na celých číslech.

- součet (+);
- rozdíl (-);
- součin (*);
- celočíselný podíl (//), např. `11 // 3 == 3`;

Celá čísla

Python umí následující operace na celých číslech.

- součet (+);
- rozdíl (-);
- součin (*);
- celočíselný podíl (//), např. `11 // 3 == 3`;
- zbytek po dělení (%), např. `11 % 3 == 2`;

Celá čísla

Python umí následující operace na celých číslech.

- součet (+);
- rozdíl (-);
- součin (*);
- celočíselný podíl (//), např. `11 // 3 == 3`;
- zbytek po dělení (%), např. `11 % 3 == 2`;
- mocninu (**), např. `4 ** 3 == 64`.

Desetinná čísla

Datový typ float

Zkratka `float` (z angl. `floating point`) označuje v Pythonu typ desetinných čísel.

Desetinná čísla

Datový typ float

Zkratka `float` (z angl. `floating point`) označuje v Pythonu typ desetinných čísel.

Poznámka. Celá čísla jsou samozřejmě zároveň desetinná. Aby je Python v tomto případě rozlišil, píše 2.0 pro „desetinné číslo“ dva a 2 pro „celé číslo“ dva.

Desetinná čísla

Python umí následující operace na desetinných číslech.

- součet (+);
- rozdíl (-);
- součin (*);
- podíl (/);
- mocninu (**).

celá ↔ desetinná

- Slova `int` a `float` jsou zároveň názvy funkcí/procedur v Pythonu pro převod mezi číselnými typy.

celá \leftrightarrow desetinná

- Slova `int` a `float` jsou zároveň názvy funkcí/procedur v Pythonu pro převod mezi číselnými typy.
- `int(x: float)` vrátí tzv. „celou část“ z `x`; např. `int(3.9) == 3`.

celá \leftrightarrow desetinná

- Slova `int` a `float` jsou zároveň názvy funkcí/procedur v Pythonu pro převod mezi číselnými typy.
- `int(x: float)` vrátí tzv. „celou část“ z `x`; např. `int(3.9) == 3`.
- `float(x: int)` převede celé číslo `x` na desetinné prostě tak, že k němu přidá „.0“. Takže třeba `float(3) == 3.0`.

Datové typy

Řetězce

Řetězce (stringy)

Datový typ `str`

Zkratkou `str` (z angl. `string`) Python označuje typ „řetězců znaků“, tj. posloupností v zásadě libovolných symbolů.

Řetězce (stringy)

Datový typ `str`

Zkratkou `str` (z angl. `string`) Python označuje typ „řetězců znaků“, tj. posloupností v zásadě libovolných symbolů.

- Stringy se píší do uvozovek, buď jednoduchých (') nebo dvojitých ("). Na výběru nezáleží, ale string musí začínat končit stejnou uvozovkou.

Řetězce (stringy)

Datový typ `str`

Zkratkou `str` (z angl. `string`) Python označuje typ „řetězců znaků“, tj. posloupností v zásadě libovolných symbolů.

- Stringy se píší do uvozovek, buď jednoduchých (') nebo dvojitých ("). Na výběru nezáleží, ale string musí začínat končit stejnou uvozovkou.
- Python používá pro kódování textu UTF-8 (**U**nicode **T**ransformation **F**ormat – 8-bit). Tedy umí rozpoznat každý znak v tomto kódování.

Řetězce (stringy)

Python umí následující operace na řetězcích.

- součet/spojení (+ nebo mezera)
 - např. `"auto" + "bus" == "autobus"`
 - např. `"mrt" "vola" == "mrtvola"`

Řetězce (stringy)

Python umí následující operace na řetězcích.

- součet/spojení (+ nebo mezera)
 - např. `"auto" + "bus" == "autobus"`
 - např. `"mrt" "vola" == "mrtvola"`
- součin/opakování (*): např. `"hehe" * 3 == "hehehehehehe"`

Řetězce (stringy)

Python umí následující operace na řetězcích.

- součet/spojení (+ nebo mezera)
 - např. `"auto" + "bus" == "autobus"`
 - např. `"mrt" "vola" == "mrtvola"`
- součin/opakování (*): např. `"hehe" * 3 == "hehehehehehe"`
- výběr prvku (`str[pořadí prvku]`): např. `"python"[2] == "t"`.
Pozor! Python čísluje od 0.

stringy ↔ čísla

- Zkratka `str` je zároveň procedura na převod dané proměnné na string. V případě čísel máme
 - `str(x: int)` převede celé číslo na string. Třeba `str(3) == "3"`.
 - `str(x: float)` převede desetinné číslo na string. Např. `str(3.14159) == "3.14159"`.

Pozor! Python neřeší, jestli je ve stringu číslo. Takže třeba `"1" + "1" == "11"`, ale `1 + 1 == 2`.

stringy ↔ čísla

Procedury `int` a `float` taky převádějí stringy na číslo, pokud to lze. Např.

- `int("69") == 69`,

stringy ↔ čísla

Procedury `int` a `float` taky převádějí stringy na číslo, pokud to lze. Např.

- `int("69") == 69`,
- `float("3.14159") == 3.14159`, ale

stringy ↔ čísla

Procedury `int` a `float` taky převádějí stringy na číslo, pokud to lze. Např.

- `int("69") == 69`,
- `float("3.14159") == 3.14159`, ale
- `float("hehe")` i `int("9.11")` hodí chybu.

Datové typy

Seznamy

Seznamy

Datový tip list

Slovem `list` označuje Python seznam; vlastně množinu, kde každý prvek má jednoznačné pořadí. **Prvky v seznamu mohou nahrazovat, přidávat a odebírat.**

Seznamy

Datový tip list

Slovem `list` označuje Python seznam; vlastně množinu, kde každý prvek má jednoznačné pořadí. **Prvky v seznamu mohou nahrazovat, přidávat a odebírat.**

- Seznamy se píší do **hranatých závorek** `[]` a prvky oddělují čárkami. Třeba `[2, "hora", 4, 7]` je seznam se čtyřmi prvky.

Seznamy

Datový tip list

Slovem `list` označuje Python seznam; vlastně množinu, kde každý prvek má jednoznačné pořadí. **Prvky v seznamu mohou nahrazovat, přidávat a odebírat.**

- Seznamy se píší do **hranatých závorek** `[]` a prvky oddělují čárkami. Třeba `[2, "hora", 4, 7]` je seznam se čtyřmi prvky.
- Prvkem seznamu může být další seznam. Třeba `[1, [2, "tři"], 4]` je seznam, jehož druhým prvkem je seznam `[2, "tři"]`.

Seznamy

Python umí následující operace na seznamech.

- součet/spojení (+)
 - např. `[69, 420] + [911, 1337] == [69, 420, 911, 1337]`.

Seznamy

Python umí následující operace na seznamech.

- součet/spojení (+)
 - např. `[69, 420] + [911, 1337] == [69, 420, 911, 1337]`.
- součin/opakování (*)
 - např. `[1, 2] * 4 == [1, 2, 1, 2, 1, 2, 1, 2]`.

Seznamy

Python umí následující operace na seznamech.

- součet/spojení (+)
 - např. `[69, 420] + [911, 1337] == [69, 420, 911, 1337]`.
- součin/opakování (*)
 - např. `[1, 2] * 4 == [1, 2, 1, 2, 1, 2, 1, 2]`.
- výběr prvku (`list[pořadí prvku]`)
 - např. `[1, 4, 7, "hroch"][2] == 7`.
 - např. `[1, 4, 7, "hroch"][-1] == "hroch"`.

Pozor! Python čísluje buď od 0 nahoru (začátek → konec) nebo od -1 dolu (konec → začátek)

Datové typy

N-tice

N-tice

Datový typ tuple

Slovem `tuple` Python označuje n-tici, neboli posloupnost n prvků.
Prvky n-tice nemohu nahrazovat, přidávat ani odebírat.

N-tice

Datový typ tuple

Slovem **tuple** Python označuje n-tici, neboli posloupnost n prvků.
Prvky n-tice nemohu nahrazovat, přidávat ani odebírat.

- N-tice se píší buď kulatou závorkou s prvky oddělenými čárkou, třeba (1, 2), nebo v mnoha případech i bez závorky, třeba 1, 2.

N-tice

Datový typ tuple

Slovem `tuple` Python označuje n-tici, neboli posloupnost n prvků.
Prvky n-tice nemohu nahrazovat, přidávat ani odebírat.

- N-tice se píší buď kulatou závorkou s prvky oddělenými čárkou, třeba (1, 2), nebo v mnoha případech i bez závorky, třeba 1, 2.
- Python umí stejné operace na n-ticích jako na seznamech.

Datové typy

Slovníky

Slovníky

Datový typ dict

Zkratkou `dict` (z angl. `dictionary`) označuje Python typ slovníku, tj. množiny hodnot, které jsou zařazeny pod klíči. **Slovník umožňuje nahrazování, přidávání i odebírání klíčů i hodnot.**

Slovníky

Datový typ dict

Zkratkou `dict` (z angl. `dictionary`) označuje Python typ slovníku, tj. množiny hodnot, které jsou zařazeny pod klíči. **Slovník umožňuje nahrazování, přidávání i odebírání klíčů i hodnot.**

- Slovník se píše do složených závorek `{}` a prvky jsou v podobě klíč: hodnota odděleny čárkami. Např. `{(1, 2): "kočka", 3: [4, 5], "pes": 6}`.

Slovníky

Datový typ dict

Zkratkou `dict` (z angl. `dictionary`) označuje Python typ slovníku, tj. množiny hodnot, které jsou zařazeny pod klíči. **Slovník umožňuje nahrazování, přidávání i odebírání klíčů i hodnot.**

- Slovník se píše do složených závorek `{}` a prvky jsou v podobě klíč: hodnota odděleny čárkami. Např. `{(1, 2): "kočka", 3: [4, 5], "pes": 6}`.
- Hodnotou může být cokoli, ale klíč musí být **neměnný datový typ** (číslo, slovo, n-tice apod.).

Slovníky

Slovníky nelze sčítat/spojovat ani násobit/opakovat. Jedinou základní operací je výběr prvku příkazem `dict[klíč]`. Pár příkladů:

Slovníky

Slovníky nelze sčítat/spojovat ani násobit/opakovat. Jedinou základní operací je výběr prvku příkazem `dict[klíč]`. Pár příkladů:

- `{"kočka": 2, "pes": 3}["pes"] == 3.`

Slovníky

Slovníky nelze sčítat/spojovat ani násobit/opakovat. Jedinou základní operací je výběr prvku příkazem `dict[klíč]`. Pár příkladů:

- `{"kočka": 2, "pes": 3}["pes"] == 3`.
- `{(1, 2, 3): "ted'", 4: 5}[(1, 2, 3)] == "ted'".`

Slovníky

Slovníky nelze sčítat/spojovat ani násobit/opakovat. Jedinou základní operací je výběr prvku příkazem `dict[klíč]`. Pár příkladů:

- `{"kočka": 2, "pes": 3}["pes"] == 3.`
- `{(1, 2, 3): "ted'", 4: 5}[(1, 2, 3)] == "ted'".`
- `{0: "nula", 1: "jedna", 2: "dva"}[1] == "jedna".`

Datové typy

list, tuple **a** dict **jako** procedury

list jako procedura

Procedure/funkce `list` umožňuje převod jiného datového typu na seznam, pokud to (podle Pythonu) dává smysl. Obecné pravidlo je, že Python umí převést na seznam jen ty datové typy, **které jsou číslované**.

- `list(x: int|float)` hodí chybu,

list jako procedura

Procedure/funkce `list` umožňuje převod jiného datového typu na seznam, pokud to (podle Pythonu) dává smysl. Obecné pravidlo je, že Python umí převést na seznam jen ty datové typy, **které jsou číslované**.

- `list(x: int|float)` hodí chybu,
- `list(x: str)` převede řetězec na seznam jeho znaků,
 - např. `list("kočka") == ["k", "o", "č", "k", "a"]`.

list jako procedura

Procedure/funkce `list` umožňuje převod jiného datového typu na seznam, pokud to (podle Pythonu) dává smysl. Obecné pravidlo je, že Python umí převést na seznam jen ty datové typy, **které jsou číslované**.

- `list(x: int|float)` hodí chybu,
- `list(x: str)` převede řetězec na seznam jeho znaků,
 - např. `list("kočka") == ["k", "o", "č", "k", "a"]`.
- `list(x: tuple)` převede n-tici na seznam se stejnými prvky,
 - např. `list((1, 2, 3)) == [1, 2, 3]`.

list jako procedura

Procedure/funkce `list` umožňuje převod jiného datového typu na seznam, pokud to (podle Pythonu) dává smysl. Obecné pravidlo je, že Python umí převést na seznam jen ty datové typy, **které jsou číslované**.

- `list(x: int|float)` hodí chybu,
- `list(x: str)` převede řetězec na seznam jeho znaků,
 - např. `list("kočka") == ["k", "o", "č", "k", "a"]`.
- `list(x: tuple)` převede n-tici na seznam se stejnými prvky,
 - např. `list((1, 2, 3)) == [1, 2, 3]`.
- `list(x: dict)` převede slovník na seznam klíčů.
 - např. `list({"pes": "haf", "kočka": "mňau"}) == ["pes", "kočka"]`.

tuple jako procedura

Procedura/funkce **tuple** funguje v zásadě stejně jako list. Tzn.

tuple jako procedura

Procedura/funkce `tuple` funguje v zásadě stejně jako `list`. Tzn.

- `tuple(x: int|float)` hodí chybu,
- `tuple(x: str)` udělá z řetězce n-tici jeho symbolů,
- `tuple(x: list)` převede seznam na n-tici se stejnými prvky.
- `tuple(x: dict)` převede slovník na n-tici jeho klíčů.

dict jako procedura

Procedura/funkce `dict` lze použít pouze na převod seznamu nebo n-tice, jejichž **každý prvek má délku 2** (tj. dvojice nebo seznam o dvou prvcích). Příklady:

- `dict([("pes", 2), ("kočka", 3)]) == {"pes": 2, "kočka": 3}`.
- `dict(("pes", 2), ("kočka", 3)) == {"pes": 2, "kočka": 3}`.

Proměnné a řídicí sekvence

Co tím myslím?

Co myslím řídicí sekvencí.

- podmínky (`if` → `elif` → `else`),

Co tím myslím?

Co myslím řídící sekvencí.

- podmínky (`if` → `elif` → `else`),
- cykly (`for` nebo `while`),

Co tím myslím?

Co myslím řídicí sekvencí.

- podmínky (`if` → `elif` → `else`),
- cykly (`for` nebo `while`),
- procedury/funkce (`def`).

Proměnné a řídicí sekvence

Proměnné

Proměnné v Pythonu

- Proměnné v Pythonu se dají pojmenovat v podstatě jakoukoli posloupností znaků (až na výjimky).

Proměnné v Pythonu

- Proměnné v Pythonu se dají pojmenovat v podstatě jakoukoli posloupností znaků (až na výjimky).
- Nemusíte Pythonu říkat, jaký má proměnná datový typ; on si to určí sám.

Proměnné v Pythonu

- Proměnné v Pythonu se dají pojmenovat v podstatě jakoukoli posloupností znaků (až na výjimky).
- Nemusíte Pythonu říkat, jaký má proměnná datový typ; on si to určí sám.
- Táž proměnná může být v průběhu programu různých typů.

Proměnné v Pythonu

- Proměnné v Pythonu se dají pojmenovat v podstatě jakoukoli posloupností znaků (až na výjimky).
- Nemusíte Pythonu říkat, jaký má proměnná datový typ; on si to určí sám.
- Táž proměnná může být v průběhu programu různých typů.
- Hodnota se do proměnné ukládá jednoduchým `=`.

Pozor! Tohle `=` nemá nic společného se stejným symbolem v matematice.

Čte se zprava doleva.

- např. `number = 3` znamená „do `number` dosad' 3“ a
- `number = number + 2` znamená „do `number` dosad' `number + 2`“.

Proměnné – příklady

Příklad s čísly

Program

```
first_number = 4
second_number = 5
print(first_number * second_number)
```

vytiskne 20.

Proměnné – příklady

Příklad se stringy

Program

```
first_word = "kocour"  
second_word = "kočka"  
print(first_word[3] + second_word[-2])
```

vytiskne "ok".

Proměnné – příklady

Příklad se seznamy

Program

```
inner_list = [4, "blb"]  
outer_list = ["ano", inner_list, 5, 6]  
print(outer_list)
```

vytiskne ["ano", [4, "blb"], 5, 6].

Proměnné a řídicí sekvence

Podmínky

Podmínky v Pythonu

- Podmínky se píší ve tvaru

`if` nějaká podmínka:

Pro další možnosti píšete `elif` (zkráceno z `else if`) a nakonec `else`.

Podmínky v Pythonu

- Podmínky se píší ve tvaru

`if` nějaká podmínka:

Pro další možnosti pište `elif` (zkráceno z `else if`) a nakonec `else`.

- Kód, který se má za dané podmínky vykonat, **musí být odsazen!** Ideálně odsazujte klávesou Tab.

Podmínky v Pythonu

- Podmínky se píší ve tvaru

`if` nějaká podmínka:

Pro další možnosti pište `elif` (zkráceno z `else if`) a nakonec `else`.

- Kód, který se má za dané podmínky vykonat, **musí být odsazen!** Ideálně odsazujte klávesou Tab.
- Každá (správně napsaná) podmínka je v Pythonu vyhodnocena buď jako pravda (`True`), nebo lež (`False`).

Tvoření podmínky – vnitřek

Uvnitř podmínky budeme nejčastěji používat operátory

- `in` (doslova „v“ – testuje, jestli to nalevo je uvnitř toho napravo)
 - Např. `("s" in "synek") == True`, ale
 - `(3 in [1, 2, 4, 5]) == False`.

Tvoření podmínky – vnitřek

Uvnitř podmínky budeme nejčastěji používat operátory

- `in` (doslova „v“ – testuje, jestli to nalevo je uvnitř toho napravo)
 - Např. `("s" in "synek") == True`, ale
 - `(3 in [1, 2, 4, 5]) == False`.
- `==` (testuje, jestli je nalevo to samé, co napravo). **Tohle je ten ekvivalent jednoduchého = v matici.** V Pythonu jednoduché = dosazuje do proměnných!
 - Např. `("sova"[2] == "v") == True`.

Tvoření podmínky – vnitřek

Uvnitř podmínky budeme nejčastěji používat operátory

- `in` (doslova „v“ – testuje, jestli to nalevo je uvnitř toho napravo)
 - Např. `("s" in "synek") == True`, ale
 - `(3 in [1, 2, 4, 5]) == False`.
- `==` (testuje, jestli je nalevo to samé, co napravo). **Tohle je ten ekvivalent jednoduchého = v matice.** V Pythonu jednoduché = dosazuje do proměnných!
 - Např. `("sova"[2] == "v") == True`.
- `<, >, <=, >=` (porovnání toho, co je nalevo, s tím, co je napravo). Symboly `<=` a `>=` značí „menší nebo rovno“ a „větší nebo rovno“, resp.
 - Např. `(5 > 3) == True` a
 - `("c" <= "f") == True`.

Tvoření podmínky – vnějšek

Vně podmínek budeme používat (logické) operátory

- `not` (doslova „ne“ – logický opak podmínky)
 - Např. `not (5 > 3) == False` a
 - `not ("x" in "kocour") == True`.
Místo `not ("x" in "kocour")` lze psát (přirozenějc) `"x" not in "kocour"`.

Tvoření podmínky – vnějšek

Vně podmínek budeme používat (logické) operátory

- **not** (doslova „ne“ – logický opak podmínky)
 - Např. `not (5 > 3) == False` a
 - `not ("x" in "kocour") == True`.
Místo `not ("x" in "kocour")` lze psát (přirozenějc) `"x" not in "kocour"`.
- **and** (doslova „a“ – musí platit obě podmínky)
 - Např. `(5 > 3 and "s" in "synek") == True`,
 - `(3 <= 4 and "x" in "kocour") == False` a
 - `(1 in [2, 3] and "x" in "kocour") == False`.

Tvoření podmínky – vnějšek

Vně podmínek budeme používat (logické) operátory

- **or** (doslova „nebo“ – musí platit **alespoň** jedna z podmínek)
 - Např. `(5 > 3 or "s" in "synek") == True`,
 - `(3 <= 4 or "x" in "kocour") == True` a
 - `(1 in [2, 3] or "x" in "kocour") == False`.

Příklad – liché číslo

Program, který určuje, jestli je číslo liché, může vypadat třeba takto.

```
number = 5
if number % 2 == 1:
    print(str(number) + " je liché.")
else:
    print(str(number) + " je sudé.")
```

Proměnné a řídící sekvence

Cykly

for cyklus v Pythonu

- for cyklus se v Pythonu píše

```
for prvek in seznam/n-tice/slovník:
```

a kód uvnitř cyklu se odsazuje.

Pozor! Python prochází seznam a n-tici po prvcích, ale **slovník po klíčích**.

for cyklus v Pythonu

- for cyklus se v Pythonu píše

```
for prvek in seznam/n-tice/slovník:
```

a kód uvnitř cyklu se odsazuje.

Pozor! Python prochází seznam a n-tici po prvcích, ale **slovník po klíčích**.

- Proměnná pro cyklus se může jmenovat jakkoliv. Python do ní během cyklu postupně dosazuje všechny prvky seznamu/n-tice, klíče slovníku a po jeho konci ji zapomene.

for cyklus v Pythonu

- for cyklus se v Pythonu píše

```
for prvek in seznam/n-tice/slovník:
```

a kód uvnitř cyklu se odsazuje.

Pozor! Python prochází seznam a n-tici po prvcích, ale **slovník po klíčích**.

- Proměnná pro cyklus se může jmenovat jakkoliv. Python do ní během cyklu postupně dosazuje všechny prvky seznamu/n-tice, klíče slovníku a po jeho konci ji zapomene.
- Důležitá je funkce `range(n: int)`, která vrací seznam přirozených čísel menších než `n`. Např. `range(5) == [0, 1, 2, 3, 4]`.

Příklad – průchod seznamem

Program, který vytiskne každý prvek seznamu krát dva lze napsat jako

```
random_stuff = [1, "hračka", [2, 3], (4, 5)]  
for wtv in random_stuff:  
    print(wtv * 2)
```

Příklad – průchod seznamem

Program, který vytiskne každý prvek seznamu krát dva lze napsat jako

```
random_stuff = [1, "hračka", [2, 3], (4, 5)]  
for wtv in random_stuff:  
    print(wtv * 2)
```

nebo použitím `range` jako

```
random_stuff = [1, "hračka", [2, 3], (4, 5)]  
for index in range(4):  
    print(random_stuff[index] * 2)
```

while cyklus v Pythonu

- while cyklus se v Pythonu píše

```
while podmínka:
```

a obsah cyklu je odsazený.

while cyklus v Pythonu

- while cyklus se v Pythonu píše

```
while podmínka:
```

a obsah cyklu je odsazený.

- Stavění podmínek ve while cyklu je stejné jako v if.

Příklad – mocniny dvojky

Program, který vypíše všechny mocniny dvojky menší než dané číslo `limit`, může vypadat třeba takhle.

```
limit = 69 ** 69
power_of_two = 2
while power_of_two < limit:
    print(power_of_two)
    power_of_two = power_of_two * 2
```

Proměnné a řídicí sekvence

Funkce

Funkce/procedury v Pythonu

- V Pythonu se funkce píše

```
def jméno funkce(jména parametrů):
```

a obsah funkce je odsazený.

Funkce/procedury v Pythonu

- V Pythonu se funkce píše

```
def jméno funkce(jména parametrů):
```

a obsah funkce je odsazený.

- `def` je z angl. `define`.

Funkce/procedury v Pythonu

- V Pythonu se funkce píše

```
def jméno funkce(jména parametrů):
```

a obsah funkce je odsazený.

- `def` je z angl. `define`.
- Pro ukončení funkce a vrácení nějaké hodnoty slouží

```
return hodnota
```

Pozor! Funkce **nemusí vracet nic**. Jakmile provede svůj obsah, skončí sama, i když `return` nikam nenapišete.

Příklady funkcí

Funkce, co dostane jméno a příjmení a vrátí je spojená dohromady, se dá napsat třeba takhle.

```
def whole_name(name, surname):  
    return name + " " + surname
```

Příklady funkcí

Funkce, co dostane jméno a příjmení a vrátí je spojená dohromady, se dá napsat třeba takhle.

```
def whole_name(name, surname):  
    return name + " " + surname
```

Další funkce, co dostane věk a připojí za něj „let“, vypadá

```
def age_to_string(age):  
    return str(age) + " let"
```

Příklad – využití funkcí z před. slidu

Řekněme, že máme daný seznam `data` trojic (jméno, příjmení, věk), kde jméno a příjmení jsou stringy a věk je int. Pomocí funkcí z předchozího slidu ho pěkně vytiskneme.

```
for (name, surname, age) in data:
    whole_name = whole_name(name, surname)
    age = age_to_string(age)
    print(whole_name + ", " + age)
```

Built-in funkce

Co jsou built-in funkce?

- Funkce, které jsou součástí Pythonu a provádějí běžné nebo užitečné operace.

Co jsou built-in funkce?

- Funkce, které jsou součástí Pythonu a provádějí běžné nebo užitečné operace.
- Např. zjišťování délky stringu/seznamu, převádění mezi datovými typy, přidávání prvků do seznamu, řazení seznamu apod.

Co jsou built-in funkce?

- Funkce, které jsou součástí Pythonu a provádějí běžné nebo užitečné operace.
- Např. zjišťování délky stringu/seznamu, převádění mezi datovými typy, přidávání prvků do seznamu, řazení seznamu apod.
- Některé built-in funkce lze použít na různé datové typy a některé se vážou ke konkrétním typům.

Built-in funkce

Obecné

Funkce `abs`

Vrací absolutní hodnotu celého/desetinného čísla.

Funkce `abs`

Vrací absolutní hodnotu celého/desetinného čísla.

Například

- `abs(-5) == 5`,
- `abs(2.718) == 2.718`.

Pro žádný jiný běžný datový typ nefunguje.

Funkce `enumerate`

Vytvoří ze seznamu/n-tice seznam dvojic (pořadí, hodnota).

Funkce `enumerate`

Vytvoří ze seznamu/n-tice seznam dvojic (pořadí, hodnota).

Obvykle se používá s for cyklem. Např.

```
lst = ["pes", "kočka"]  
for i, element in enumerate(lst):  
    print(i, element)
```

Funkce `enumerate`

Vytvoří ze seznamu/n-tice seznam dvojic (pořadí, hodnota).

Obvykle se používá s `for` cyklem. Např.

```
lst = ["pes", "kočka"]  
for i, element in enumerate(lst):  
    print(i, element)
```

vytiskne

```
0 pes  
1 kočka
```

Funkce `filter`

Dostává dva parametry – **funkci** a **seznam/n-tici/slovník**. Vráť jako seznam prvky, pro které **funkce** vrací `True`.

Funkce `filter`

Dostává dva parametry – **funkci** a **seznam/n-tici/slovník**. Vráť jako seznam prvky, pro které **funkce** vrací `True`.

Například, pokud si definuju funkci

```
def divisible_by_five(cislo):  
    return (cislo % 5 == 0)
```

Funkce `filter`

Dostává dva parametry – **funkci** a **seznam/n-tici/slovník**. Vrábí jako seznam prvky, pro které **funkce** vrací `True`.

Například, pokud si definuju funkci

```
def divisible_by_five(cislo):  
    return (cislo % 5 == 0)
```

pak můžu ze seznamu `lst = [4, 5, 10, 13]` dostat seznam všech čísel dělitelných pěti jako

```
lst2 = filter(divisible_by_five, lst)
```

Funkce `input`

Vrátí vstup od uživatele jako typ `string`. Jako parametrem dostává string, který se uživateli zobrazí.

Funkce `input`

Vrátí vstup od uživatele jako typ `string`. Jako parametrem dostává `string`, který se uživateli zobrazí.

Například, napíšu-li

```
birth_year = input("Zadejte svůj rok narození: ")
```

Funkce `input`

Vrátí vstup od uživatele jako typ `string`. Jako parametrem dostává `string`, který se uživateli zobrazí.

Například, napíšu-li

```
birth_year = input("Zadejte svůj rok narození: ")
```

uživateli se zobrazí

Zadejte svůj rok narození:

a odpověď se uloží **jako `string`** do proměnné `birth_year`.

Funkce `isinstance`

Dostane dva parametry – **proměnnou** a **datový typ** – a odpoví, zda je proměnná onoho typu.

Funkce `isinstance`

Dostane dva parametry – **proměnnou** a **datový typ** – a odpoví, zda je proměnná onoho typu.

Například,

- `isinstance('kočka', str) == True`,
- `isinstance(3.14, float) == True`,
- `isinstance(4, list) == False` a
- `isinstance([1, 2], tuple) == False`.

Funkce `len`

Parametrem dostane cokoli, kde má „smysl“ počítat počet prvků. Ten vrátí jako celé číslo.

Funkce `len`

Parametrem dostane cokoli, kde má „smysl“ počítat počet prvků. Ten vrátí jako celé číslo.

Například

- `len("auto") == 4`,
- `len([4, 5, 6]) == 3` a
- `len({1: "jedna", 2: "dva"}) == 2`.

Funkce `map`

Dostane parametrem **funkci** a **seznam/n-tici/slovník** a vrátí seznam s **funkcí** aplikovanou na každý prvek.

Funkce `map`

Dostane parametrem **funkci** a **seznam/n-tici/slovník** a vrátí seznam s **funkcí** aplikovanou na každý prvek.

Například s funkcí

```
def last_letter(word):  
    return word[-1]
```

Funkce `map`

Dostane parametrem **funkci** a **seznam/n-tici/slovník** a vrátí seznam s **funkcí** aplikovanou na každý prvek.

Například s funkcí

```
def last_letter(word):  
    return word[-1]
```

můžu ze seznamu slov `words = ["pes", "kočka", "dikobraz"]` získat seznam jenom jejich posledních písmen jako

```
last_letters = map(last_letter, words)
```

Funkce `max/min`

Ze seznamu/n-tice/slovníku vrátí největší/nejmenší prvek.

Funkce `max/min`

Ze seznamu/n-tice/slovníku vrátí největší/nejmenší prvek.

Například,

- `max([4, 7, 2]) == 7,`
- `min([4, 7, 2]) == 2,`
- `max(["pes", "kočka", "dikobraz"]) == "pes",`
- `min(["pes", "kočka", "dikobraz"]) == "dikobraz".`

Funkce `print`

Vytiskne to, co dostane parametrem, do konzole.

Funkce `print`

Vytiskne to, co dostane parametrem, do konzole.

Například

- `print(3)` vytiskne číslo 3 a
- `print("kočka")` vytiskne kočka.

Funkce `range`

Dostane parametrem celé číslo a vrátí seznam celých čísel menších než toto číslo počínaje 0.

Funkce `range`

Dostane parametrem celé číslo a vrátí seznam celých čísel menších než toto číslo počínaje 0.

Nejčastěji se používá s for cyklem. Tedy, např.

```
for i in range(5):  
    print(i * 2)
```

vytiskne postupně čísla 0, 2, 4, 6, 8, protože `range(5) == [0, 1, 2, 3, 4]`.

Funkce `round`

Zaokrouhlí dané desetinné číslo – od .5 nahoru, jinak dolu.

Funkce `round`

Zaokrouhlí dané desetinné číslo – od .5 nahoru, jinak dolu.

Například,

- `round(3.14) == 3` a
- `round(2.718) == 3.`

Funkce `sum`

Vrátí součet všech prvků v daném seznamu/n-tici. Funguje pouze pro čísla.

Funkce `sum`

Vrátí součet všech prvků v daném seznamu/n-tici. Funguje pouze pro čísla.

Například,

- `sum([1, 2, 3, 4]) == 10,`
- `sum((69, 420, 1337)) == 1826.`

Funkce `zip`

Dostane parametrem libovolný počet seznamů/n-tic a vrátí seznam, kde každým prvkem je n-tice prvků na odpovídajících pozicích těchto seznamů/n-tic.

Funkce `zip`

Dostane parametrem libovolný počet seznamů/n-tic a vrátí seznam, kde každým prvkem je n-tice prvků na odpovídajících pozicích těchto seznamů/n-tic.

Například,

- `zip([1, 2, 3], [4, 5, 6]) == [(1, 4), (2, 5), (3, 6)]`
- `zip("kočka", "pes"), ("dělá", "dělá"), ("haf", "mňau")) == [("kočka", "dělá", "haf"), ("pes", "dělá", "mňau")].`

Funkce `int`, `float`, `str`, `list`, `tuple`, `dict`

Převádějí jiné datové typy na stejnojmenné.

Funkce `int`, `float`, `str`, `list`, `tuple`, `dict`

Převádějí jiné datové typy na stejnojmenné.

Například,

- `int("5") == 5`,
- `str(1.23) == "1.23"`,
- `tuple([4, 5, 6]) == (4, 5, 6)`.

Díky za pozornost.