

Legal Tech: I

Jan Černý, Tomáš Knap

crimczech

Charles University

Version 1.0, 2026

Anonymní feedback



Naskenujte telefonem nebo klikněte: dvě minuty, bez jména

- **Proč programovat:** rutinu svěříme skriptu, právní posouzení zůstává na nás
- **Počítač** vykonává jednoduché instrukce: rychle, neúnavně a naprosto doslovně
- **Součásti:** procesor (úředník), paměť (stůl), úložiště (spisovna), vstup a výstup (podatelna)
- **Všechno jsou čísla:** text, obraz, zvuk i instrukce; význam jim dává až kontext
- **Programovací jazyk:** zápis, který čtou lidé a který jde strojově přeložit na čísla
- **Kompilace** přeloží celý program předem, **interpretace** ho provádí příkaz po příkazu
- **Python** je čitelný, interpretovaný a dynamicky typovaný; proto v něm pracujeme

Syntaxe Pythonu

Proměnné, datové typy, operace a podmínky

Základy syntaxe Pythonu

- Program je textový soubor, třeba `program.py`; spustíme ho trojúhelníčkem ► **Run Python File** vpravo nahoře ve VS Codium
- Interaktivní režim: `python` (na Macu `python3`) nebo `uv run python` otevře řádek se `>>>`, kam píšeme příkazy a hned vidíme výsledek
- Na slajdech: řádky se `>>>` píšeme my, řádky pod nimi jsou odpověď Pythonu

```
>>> 2 + 3
5
>>> print("Dobrý den")
Dobrý den
```

- Odsazení (4 mezery) říká, které řádky patří k sobě; velká a malá písmena se rozlišují: `print` funguje, `Print` ne

Interaktivní režim ve VSCode

1. Otevřete složku s projektem: **File** → **Open Folder...**
2. Otevřete terminál: **Terminal** → **New Terminal**; objeví se ve spodní části okna
3. Napište `python` (na Macu `python3`) nebo `uv run python` a potvrďte Enterem; objeví se `>>>` a Python čeká na příkazy

```
$ python
Python 3.14.7 (main, ...) on linux
>>> 2 + 3
5
>>> exit()
```

- Na Macu a většině Linuxů se příkaz jmenuje `python3`; samotné `python` tam nemusí existovat
- Ukončení: `exit()`. Šipka nahoru vrátí předchozí příkaz
- Nic se neukládá: je to hřiště na zkoušení, hotový program patří do souboru

```
# Komentář začíná mřížkou. Python ho přeskočí, je pro lidi.  
vek = 17 # může být i za příkazem  
  
# Hodí se i na dočasné vypnutí řádku:  
# print("tohle se nevypíše")
```

- Komentář je důvodová zpráva k programu: vysvětluje **proč**, ne co (to říká kód sám)
- Píšeme je pro kolegy a pro sebe za půl roku
- Přebytek škodí: `vek = 17 # uloží 17 do vek` neříká nic

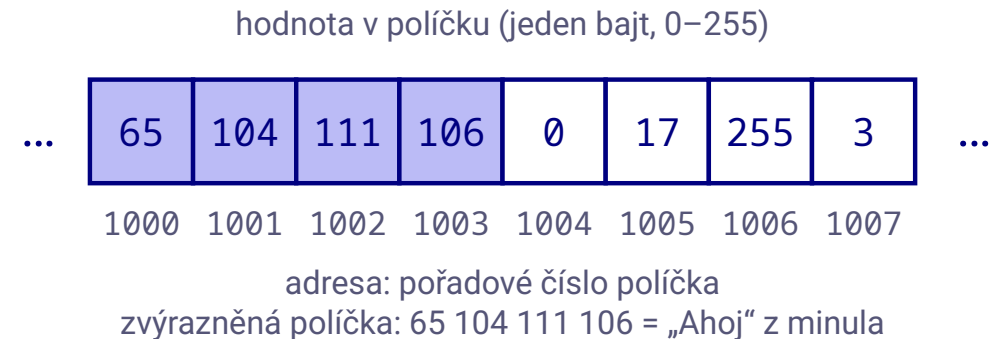
print(): výpis na obrazovku

```
>>> print("Dobrý den")           # text v uvozovkách
Dobrý den
>>> print(2026)                  # číslo bez uvozovek
2026
>>> print("Lhůta", 15, "dnů")    # více argumentů oddělených čárkou
Lhůta 15 dnů
>>> print("Úrok", 8.5, "%")
Úrok 8.5 %
```

- `print()` je **funkce**: za jménem jsou závorky a v nich **argument**, tedy to, co má vypsát
- Argumentů může být víc; `print()` je vypíše za sebou a oddělí mezerou
- Text patří do uvozovek, čísla ne; každý `print()` začíná na novém řádku
- V souboru se nic nevypíše samo: co chceme vidět, musíme poslat do `print()`

Paměť: páska očíslovaných políček

- **Operační paměť (RAM):** dlouhá řada políček, v každém jedno číslo 0–255 (bajt)
- Každé políčko má **adresu**, tedy pořadové číslo
- Procesor čte i zapisuje na libovolnou adresu, v libovolném pořadí a stejně rychle
- Turingova páska z minula, jen konečná a očíslovaná
- Po vypnutí se vymaže; v políčkách jsou vždy jen čísla



Proměnná

Pojmenované místo v paměti. Má jméno a hodnotu, a tím pádem i typ.

```
vek = 17          # do proměnné vek ulož číslo 17
jmeno = "Anna"    # do proměnné jmeno ulož text
print(jmeno, "má", vek, "let")
```

- V zápisu `vek = 17` je `vek` **název proměnné** a `17` **literál**: hodnota zapsaná přímo v kódu
- `=` neznamená „rovná se“, ale **přiřad**: vpravo hodnota, vlevo jméno, kam ji uložit
- Jméno: malá písmena a podtržítka (`datum_narozeni`), bez diakritiky a mezer, nezačíná číslicí
- Volte výstižná jména: `vek` , ne `x` ; program se pak čte jako text

Práce s proměnnými

```
>>> pocet = 3          # zapíšu hodnotu 3 do proměnné pocet
>>> pocet
3
>>> pocet = 5          # přepíšu hodnotu v proměnné pocet na 5
>>> pocet
5
>>> pocet = pocet + 1
>>> pocet
6
>>> x = 5
>>> x = "pět"          # nová hodnota může mít i jiný typ (dynamické typování)
```

- Hodnotu lze kdykoli přepsat, i hodnotou jiného typu: typ patří hodnotě, ne proměnné
- Proměnná musí vzniknout dřív, než ji použijeme, jinak `NameError`

Stejná čísla, jiný význam

V paměti leží čtyři políčka: 65 104 111 106 . Co to je?

Čteno jako	Výsledek
čtyři znaky textu	„Ahoj“
jedno velké celé číslo	1 097 363 306
dvě menší celá čísla	16 744 a 28 522
desetinné číslo	14,53

- Samotná čísla význam nemají, musíme vědět, **jak je číst**. Jako ve spisu: „15“ může být věk, paragraf i délka trestu, podle kolonky
- Ta „kolonka“ se v programování jmenuje **datový typ**

Datový typ

Říká, co čísla v paměti znamenají a co s nimi jde dělat. Každá hodnota v Pythonu má svůj typ.

Typ	Co to je	Příklady
bool	pravda / nepravda	True , False
int	celé číslo	17 , -3 , 2026
float	desetinné číslo	3.14 , 0.5 , 2.0
str	text (řetězec znaků)	"Ahoj" , '§ 205' , "17"

- Python typ pozná sám ze zápisu; prozradí ho funkce `type()`, třeba `type(17)` vypíše `<class 'int'>`

bool: pravda, nebo nepravda

bool

Logická hodnota: odpověď na otázku typu ano/ne. Uplynula lhůta? Je smlouva podepsaná?

```
a = True  
b = False
```

- Jen dvě hodnoty, `True` a `False`, s velkým písmenem na začátku
- Většinou nevzniká zápisem, ale **porovnáním**: podmínka splněna, nebo ne
- Základ rozhodování v programu, vrátíme se k němu u konstrukce `if`

bool: základní operace

```
>>> a = True
>>> b = False
>>> not a           # not hodnotu otočí: z True udělá False a naopak
False
>>> not b
True
>>> a == b          # == se ptá, zda jsou hodnoty stejné; odpověď je zase bool
False
>>> a == (not b)
True
```

- `not` čteme „není pravda, že“; `==` se píše **dvěma** rovnítky, jedno `=` je přiřazení

bool: and, or

a	b	a and b	a or b
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

```
>>> podepsano = True
>>> zaplaceno = False
>>> podepsano and zaplaceno
False
>>> podepsano or zaplaceno
True
```

- `and` je „a zároveň“: pravda, jen když platí **obojí**
- `or` je „nebo“: pravda, když platí **aspoň jedno**, tedy i obojí; nikdy neznamená „bud', anebo“

int: celá čísla

int

Celé číslo: kladné, záporné i nula, bez desetinné části. Název je zkratka slova *integer*.

```
a = 2  
b = -8  
c = 1_000_000_000
```

- Celá čísla bez omezení velikosti, na rozdíl od většiny jiných jazyků
- Podtržítko je jen pro čitelnost, Python ho ignoruje: `1_000_000`
- Hodí se na počty, roky, pořadí, částky v haléřích

float: desetinná čísla

float

Desetinné číslo uložené na 64 bitech, tedy s omezenou přesností. Název je zkratka z *floating point*.

```
a = 0.5  
b = -2.8  
c = 1.0    # takto můžete uložit celé číslo jako float
```

- Desetinná **tečka**, ne čárka: 3.14
- Uvnitř je uložené dvojkově, a proto není přesné: ukázka na dalším slajdu

float není přesný

```
>>> 0.1 + 0.2
0.30000000000000004
>>> 0.1 + 0.2 == 0.3
False
>>> 19.99 * 100
1998.9999999999998
>>> 1.1 * 3
3.3000000000000003
```

- Dvojkový zápis `0.1` je nekonečný, jako $1/3$ desítkově; uloží se nejbližší možná hodnota
- Chyba je až kolem 16. platné číslice, ale `==` ji odhalí: floaty neporovnáváme na přesnou rovnost
- Peníze proto počítáme v haléřích jako `int`; pro účetnictví existuje přesný typ `Decimal`

int a float: základní operace

Operátor	Co dělá	Příklad	Výsledek
+	sčítání	<code>7 + 2</code>	<code>9</code>
-	odčítání	<code>7 - 2</code>	<code>5</code>
*	násobení	<code>7 * 2</code>	<code>14</code>
/	dělení; výsledek je vždy <code>float</code>	<code>7 / 2</code>	<code>3.5</code>
//	celočíselné dělení: kolikrát se 2 vejde do 7	<code>7 // 2</code>	<code>3</code>
%	zbytek po dělení (modulo)	<code>7 % 2</code>	<code>1</code>
**	mocnina	<code>7 ** 2</code>	<code>49</code>
==	rovnost; výsledek je <code>bool</code>	<code>7 == 7.0</code>	<code>True</code>

- Dělit nulou nejde ani v Pythonu: `7 / 0` skončí chybou `ZeroDivisionError: division by zero`

int a float: typ výsledku

Operátor	int a int	int a float	float a float
+ - * ** // %	int	float	float
/	float	float	float
==	bool	bool	bool

```
>>> 7 * 2.0      # stačí jeden float a výsledek je float
14.0
>>> 6 / 2        # dělení dá float vždy, i když vyjde celé číslo
3.0
>>> 7 // 2.0     # celočíselné dělení s floatem: celé číslo, ale typu float
3.0
```

- Typ výsledku určuje operátor a typy hodnot, ne to, co vyjde; ověříte ho funkcí `type()`

Přiřazení s operací: +=, -= a další

Zkrácený zápis	Znamená totéž co	Z <code>x = 7</code> udělá
<code>x += 2</code>	<code>x = x + 2</code>	9
<code>x -= 2</code>	<code>x = x - 2</code>	5
<code>x *= 2</code>	<code>x = x * 2</code>	14
<code>x /= 2</code>	<code>x = x / 2</code>	3.5
<code>x %= 2</code>	<code>x = x % 2</code>	1

- Stejně fungují i `//=` a `**=`; u textu `+=` připojuje: `jmeno += " Nováková"`
- Typické použití: počítadlo (`pocet += 1`) a průběžný součet (`celkem += castka`)
- Proměnná už musí existovat: `pocet += 1` bez `pocet = 0` skončí chybou `NameError`

Vestavěné funkce pro čísla

Funkce	Co dělá	Příklad	Výsledek
<code>abs()</code>	absolutní hodnota	<code>abs(-8)</code>	8
<code>pow()</code>	mocnina, totéž co <code>**</code>	<code>pow(2, 10)</code>	1024
<code>min()</code>	nejmenší z hodnot	<code>min(3, 7, 1)</code>	1
<code>max()</code>	největší z hodnot	<code>max(3, 7, 1)</code>	7

- Jsou k dispozici vždy a výsledek **vracejí**: `x = abs(-8)` uloží do `x` číslo 8

str

Text: řada znaků za sebou, třeba písmen, číslic a mezer. Název je zkratka slova *string*.

```
a = "Ahoj!"  
b = "42"  
c = "Legal tech"
```

- Text je v uvozovkách, dvojitéch nebo jednoduchých; uvnitř může být ten druhý druh
- Samostatný typ pro jeden znak neexistuje: písmeno je prostě text délky 1
- Co je v uvozovkách, je text, i "17": s číslem 17 nemá nic společného

```
>>> "Legal" + " " + "tech"      # + texty spojí za sebe
'Legal tech'
>>> "ha" * 3                     # * text zopakuje
'hahaha'
>>> "tech" in "Legal tech"      # in se ptá, zda je první text uvnitř druhého
True
>>> "Anna" == "anna"            # == porovnává přesně, včetně velikosti písmen
False
```

- Výsledkem `+` a `*` je nový text, výsledkem `in` a `==` je `bool`
- Stejný operátor dělá u textu něco jiného než u čísel: `2 + 3` je `5`, ale `"2" + "3"` je `'23'`

Neplatné operace: TypeError

```
>>> "17" - 1
TypeError: unsupported operand type(s) for -: 'str' and 'int'
>>> "17" + 1
TypeError: can only concatenate str (not "int") to str
>>> "Ahoj" / 2
TypeError: unsupported operand type(s) for /: 'str' and 'int'
>>> "3" * "4"
TypeError: can't multiply sequence by non-int of type 'str'
```

- Python nehádá, co jsme mysleli: `"17"` je text, ne číslo, a odečítat od textu neumí
- `TypeError` čteme jako „tato operace pro tyto typy neexistuje“; program se na tom řádku zastaví
- Zrádnější je, co projde, ale dělá něco jiného: `"17" * 2` není `34`, ale `'1717'`

str: funkce len(), ord() a chr()

```
>>> len("Legal tech")      # počet znaků, mezera se počítá
10
>>> len("")                # prázdný text
0
>>> ord("A")               # číslo znaku v tabulce Unicode
65
>>> ord("ř")               # 
345
>>> chr(80)                # a zpět: z čísla znak
'p'
```

- Funkce voláme jménem a text jí dáme do závorek: `len(text)`; výsledek je číslo
- `ord()` a `chr()` jsou okénko do tabulky z minulé přednášky: každý znak je uvnitř jen číslo
- Zkuste si: `chr(80) + chr(114) + chr(225) + chr(118) + chr(111)`

str: metody .lower(), .upper(), .strip()

Metoda	Co udělá	Z " Anna " vznikne
.lower()	všechna písmena malá	' anna '
.upper()	všechna písmena velká	' ANNA '
.strip()	odstraní mezery na obou koncích	'Anna'
.lstrip()	odstraní mezery jen zleva	'Anna '
.rstrip()	odstraní mezery jen zprava	' Anna '

- **Metoda** patří k typu a volá se tečkou za hodnotou: `jmeno.strip()`, ne `strip(jmeno)`
- Původní text se nemění, metoda vrací nový; výsledek si uložíme: `jmeno = jmeno.strip()`
- Typické použití: úklid vstupu `input("Jméno: ").strip()` a porovnávání přes `.lower()`

str: přístup ke znakům, slovo[i]

Znak	P	r	á	v	o
Index	0	1	2	3	4
Od konce	-5	-4	-3	-2	-1

```
>>> slovo = "Právo"
>>> slovo[0]           # znak na pozici 0, tedy první
'p'
>>> slovo[-1]          # záporný index počítá od konce
'o'
>>> slovo[5]           # za koncem textu nic není
IndexError: string index out of range
```

- Čísluje se od **nuly**, jako políčka v paměti; poslední znak má index `len(slovo) - 1`
- Výsledek je zase text délky 1; výřez `slovo[0:3]` vrátí `'Prá'`, koncový index se už nepočítá

str: metody na kontrolu obsahu

Metoda	True, když text obsahuje jen...	True pro	False pro
<code>.isalpha()</code>	písmena	"Nováková"	"Anna2", "A B"
<code>.isdigit()</code>	čísllice	"2026"	"20.5", "-5"
<code>.isalnum()</code>	písmena nebo čísllice	"ABC123"	"ABC 123"
<code>.isspace()</code>	mezery a jiné bílé znaky	" "	"a ", ""

- Výsledek je `bool`; hodí se na kontrolu vstupu od uživatele, než ho převedeme na číslo
- `.isdigit()` nepozná záporné ani desetinné číslo; prázdný text dá u všech metod `False`
- Písmena s diakritikou jsou písmena: `"Nováková".isalpha()` je `True`

f-string: text s dosazenými hodnotami

```
>>> jmeno = "Anna"
>>> vek = 17
>>> f"{jmeno} má {vek} let"           # f před uvozovkami, {} dosadí hodnotu
'Anna má 17 let'
>>> f"Za rok bude mít {vek + 1} let"  # ve {} může být i výpočet
'Za rok bude mít 18 let'
>>> f"Cena: {19.99 * 3:.2f} Kč"       # :.2f zaokrouhlí na dvě desetinná místa
'Cena: 59.97 Kč'
>>> print(f"{jmeno.upper()} má {len(jmeno)} písmena")
ANNA má 4 písmena
```

- **f** před uvozovkami zapne dosazování: obsah složených závorek se vyhodnotí a vloží do textu
- Čísla se na text převedou samy, nic nemusíme spojovat přes **+**
- Za dvojtečkou může být formát: **:.2f** znamená dvě desetinná místa

Převodní funkce: str(), int(), float(), bool()

Funkce	Co udělá	Příklad	Výsledek
str()	převeďte na text	str(17)	'17'
int()	převeďte na celé číslo; desetiny ořízne	int("17")	17
float()	převeďte na desetinné číslo	float("3.5")	3.5
bool()	převeďte na pravdu, nebo nepravdu	bool(0)	False

- Obvyklý vzor pro vstup: `vek = int(input("Věk: "))`; teprve pak jde s věkem počítat
- `bool()` dá `False` pro nulu a prázdný text, jinak `True`; pozor, i `bool("ne")` je `True`
- Co převést nejde, skončí chybou `ValueError`: třeba když uživatel napíše `sedmnáct`

Převody mezi datovými typy

```
>>> int("17") + 1          # text na celé číslo v desítkové soustavě
18
>>> float("3.5")           # text na desetinné číslo v desítkové soustavě
3.5
>>> str(2026) + ". rok"    # číslo na text
'2026. rok'
>>> int(3.99)              # ořízne desetinnou část
3
>>> int("sedmnáct")        # co převést nejde, skončí chybou
ValueError: invalid literal for int() with base 10: 'sedmnáct'
```

- Tak se opraví `"17" + 1` ze slajdu o `TypeError`: buď `int("17") + 1`, nebo `"17" + str(1)`
- Pozor na desetinnou čárku: `float("3,5")` skončí chybou, Python čeká tečku

```
vek: int = 17  
jmeno: str = "Anna"  
cena: float = 199.90  
zletily: bool = False
```

- Za jméno proměnné můžeme napsat dvojtečku a typ: **anotace**
- Python ji **nekontroluje**: `vek: int = "sedmnáct"` projde. Je to poznámka pro lidi a pro nástroje
- Editor podle ní napovídá a kontrolní nástroje (např. `mypy`) hlásí nesrovnalosti
- V kurzu je používáme, protože zpřehledňují kód; u funkcí se k nim vrátíme

Logické operace: porovnání

```
>>> vek = 17
>>> vek < 18
True
>>> vek == 17          # rovnost se píše dvěma rovnítky
True
>>> vek != 17          # nerovnost
False
```

- Porovnání `<`, `<=`, `>`, `>=`, `==`, `!=`; výsledek je vždy `bool`
- Nejčastější začátečnická chyba: `=` (přiřazení) místo `==` (porovnání)

Logické operace: and, or, not

```
>>> vek = 16
>>> vek >= 15 and vek < 18      # obě podmínky zároveň
True
>>> vek < 15 or vek >= 18      # aspoň jedna z podmínek
False
>>> not vek >= 18               # opak
True
>>> 15 <= vek < 18              # Python umí i zápis jako v matematice
True
```

- Stejné `and`, `or` a `not` jako u `bool`, jen s podmínkami z porovnání; jako v právním textu, ale bez výkladových sporů

input(): vstup od uživatele

```
jmeno = input("Jak se jmenujete? ") # zobrazí otázku a počká na odpověď  
print("Dobrý den,", jmeno)
```

- `input()` vypíše otázku, počká na odpověď ukončenou Enterem a napsaný text vrátí
- Odpověď si uložíme do proměnné, jinak se ztratí
- `input()` **vrátí vždy string**, i když uživatel napíše číslo:

```
>>> vek = input("Věk: ")  
Věk: 17  
>>> vek + 1  
TypeError: can only concatenate str (not "int") to str
```

- Proto vstup převádíme převodními funkcemi

Řízení běhu

Podmínky a cykly: jak program rozhoduje a opakuje

```
vek = 15

if vek < 18:
    print("Nezletilý")
    print("Smlouvu podepisuje zákonný zástupce")

print("Děkujeme")      # není odsazeno, provede se vždy
```

- `if podmínka:` a pod ním **odsazený blok**; blok se provede, jen když podmínka platí
- Co je odsazené, patří k `if`; první neodsazený řádek už je zase pro všechny
- Dvojtečka a odsazení jsou povinné, bez nich `SyntaxError`

if ... else

```
if vek < 18:  
    print("Nezletilý")  
else:  
    print("Zletilý")
```

```
odpoved = input("Souhlasíte? (ano/ne) ")  
if odpoved == "ano":  
    print("Pokračujeme")  
else:  
    print("Končíme")
```

- `else` = „jinak“: provede se právě jedna z větví; `else` nemá vlastní podmínku

if ... elif ... else

```
vek = int(input("Věk pachatele: "))

if vek < 15:
    print("Trestně neodpovědný")
elif vek < 18:
    print("Mladistvý: soudnictví ve věcech mládeže")
else:
    print("Dospělý: trestní zákoník")
```

- Zkouší se shora, provede se **první splněná** větev, ostatní se přeskočí
- Proto stačí `vek < 18`: kdo se dostal až sem, má aspoň 15
- `elif` může být kolikrát chceme, `else` chytí vše ostatní



```
match odpoved:  
    case "ano":  
        print("Pokračujeme")  
    case "ne":  
        print("Končíme")  
    case _:  
        print("Nerozumím")
```

- Od Pythonu 3.10; vybírá z mnoha možností podle hodnoty, `_` znamená „cokoli jiného“
- Umí mnohem víc (porovnávání vzorů), to teď nepotřebujeme; `if` a `elif` nám bohatě stačí

Cyklus while: opakuj, dokud platí podmínka

```
pocet = 0
while pocet < 3:           # dokud podmínka platí, opakuj odsazený blok
    print("Opakování", pocet)
    pocet += 1             # bez změny by podmínka platila navždy
print("Hotovo")           # sem se dostaneme, až podmínka přestane platit
```

```
Opakování 0
Opakování 1
Opakování 2
Hotovo
```

- Jako `if`, jen se podmínka po každém průchodu blokem vyhodnotí **znovu**
- Něco v bloku musí podmínku ovlivnit, jinak vznikne **nekonečný cyklus** (přeruší ho Ctrl+C)
- Hodí se, když předem nevíme, kolikrát budeme opakovat

while v praxi: ptej se, dokud není odpověď platná

```
odpoved = input("Souhlasíte? (ano/ne) ")
while odpoved != "ano" and odpoved != "ne":      # dokud odpověď není platná
    print("Nerozumím, napište ano, nebo ne.")
    odpoved = input("Souhlasíte? (ano/ne) ")
print("Vaše odpověď:", odpoved)
```

```
Souhlasíte? (ano/ne) možná
Nerozumím, napište ano, nebo ne.
Souhlasíte? (ano/ne) ano
Vaše odpověď: ano
```

- Kolikrát se uživatel splete, nevíme; `while` se ptá tak dlouho, dokud nedostane platnou odpověď
- Vzor: zeptat se, a dokud odpověď nevyhovuje, upozornit a zeptat se znovu
- Odolnější varianta: `input(...).strip().lower()` přijme i „Ano“

Když program spadne: chybová hláška

```
Traceback (most recent call last):  
  File "program.py", line 3, in <module>  
    print("Věk:", vek + 1)  
          ~~~~^~~  
TypeError: can only concatenate str (not "int") to str
```

- Čteme **odspodu**: poslední řádek říká druh chyby a co se stalo
- Nad ním je soubor, číslo řádku a řádek samotný: tam se dívat
- Hláška není nadávka, je to návod; poslední řádek klidně vložte do vyhledávače
- Python zastaví na **první** chybě: opravit, spustit znovu, opakovat

Tři nejčastější chyby

Chyba	Co znamená	Typická příčina
SyntaxError	zápis nejde přepočítat, program se nespustí	chybí dvojtečka či uvozovka, špatné odsazení
NameError	jméno, které neexistuje	překlep ve jménu, nebo proměnná ještě nevznikla
TypeError	operace pro tento typ nedává smysl	"17" + 1 : chybí převod po input()

```
if vek < 18
    print("Nezletilý")
```

```
SyntaxError: expected ':'
```

Konvence: jak psát čitelný kód

Konvence	Ano	Raději ne
výstižná jména, klidně dlouhá	<code>vek_klienta</code>	<code>v</code> , <code>x2</code>
slova ve jméně odděluje podtržítko	<code>datum_narozeni</code>	<code>datumNarozeni</code>
konstanty velkými písmeny	<code>SAZBA_DPH</code>	<code>sazba</code>
text primárně ve dvojitých uvozovkách	<code>"Anna"</code>	<code>'Anna'</code>
typové anotace všude, kde to dává smysl	<code>vstup: int</code>	
komentář tam, kde dává smysl: říká proč	<code># věk v den činu</code>	<code># uloží 17 do vek</code>

- Program běží i bez nich: konvence jsou pro lidi, kteří ho budou číst
- Další (odsazení čtyřmi mezerami, mezery kolem operátorů...) shrnuje průvodce [PEP 8](#)

Děkujeme za pozornost