O mně

- Programování od 2012, Python od 2016, Natural Language Processing od 2018
- S Pythonem jsem pracoval jak na univerzitě (ČVUT, UPV) tak v průmyslu (Anywhere, Seznam, Amazon, PromethistAI, MSD IT)
- Lektorem jsem od roku 2021 ale rád uslyšim podněty ke zlepšení
- A budem si tykat, ano? A prosim, kdo můžete tak si zapněte kamery a nebojte se mě kdykoliv zastavit a zeptat.

Co už umíte?

- Podmínky
- Slovníky
- Množiny (seznam)
- For cyklus

Tak si to procvičíme:

In [ ]:
```
if podminka
elif podminka2
elif podminka3
else
```

In [3]:
```
slovnik = dict()
```

In [4]:
```
slovnik = {}
slovnik["klic"] = "hodnota"
```

In [5]:
```
slovnik
```

Out[5]:
```
{'klic': 'hodnota'}
```

In [6]:
```
slovnik["klic"]
```

Out[6]:
```
'hodnota'
```

In [ ]:
```
a = "a"
b = "b"
# ==
a, b = "a", "b"
```

In [7]:
```
a, b = list(), []
```

In [8]:
```
a
```

Out[8]:
```
[]
```

In [9]:
```
b
```

Out[9]:
```
[]
```

In [15]:
```
a, b = tuple(), ()
```

In [16]:
```
a
```

Out[16]:
```
()
```

In [12]:
```
type(b)
```

Out[12]:
```
tuple
```

In [13]:
```
a = set()
```

In [14]:
```
a
```

Out[14]:
```
set()
```

In [ ]:
```
slovnik, tuple, seznam = {}, (), []
```

In [20]:
```
for zastupce in [1,2,3,4,5]:
    print(zastupce)
    if zastupce == 5:
        break
else:
    print("kdy se stanu?")
```

```
1
2
3
4
5
```

In [21]:
```
for zastupce in [1,2,3,4,5]:
    print(zastupce)
    if zastupce == 5:
        break
print("kdy se stanu?")
```

```
1
2
3
4
5
kdy se stanu?
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [1]:
```
# 1. Spočítejte součet všech prvků v daném seznamu pomocí for cyklu
seznam = [1, 2, 3, 4, 5]

# 2. Napište kód, který zkontroluje, zda je jméno přítomno ve slovníku a
data = {"Jan": 30, "Marie": 25, "Petr": 35}

# 3. Máte dva seznamy. Najděte společné prvky mezi nimi a uložte je do nc
seznam1 = [1, 2, 3, 4, 5]
seznam2 = [3, 4, 5, 6, 7]

# 4. Napište program, který vezme seznam čísel a vytiskne pouze sudá čísl
seznam = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# 5. Máte seznam slov. Najděte a vypište nejdelší slovo v seznamu.
slova = ["jabko", "hory", "kočka", "programování", "knihovna"]
```

In [24]:
```
# 1. Spočítejte součet všech prvků v daném seznamu pomocí for cyklu
seznam = [1, 2, 3, 4, 5]

soucet = 0
for cislo in seznam:
    soucet += cislo
print(soucet)
```

```
15
```

In [34]:
```
# 2. Napište kód, který zkontroluje, zda je jméno přítomno ve slovníku a
jmeno = "Jan" # input("zadej jmeno")
for test in ["Jan", "Marie", "Petr", "Pavel"]:
    jmeno = test
    data = {"Jan": 30, "Marie": 25, "Petr": 35}
    if jmeno in data:
        print(data[jmeno])
```

```
30
25
35
```

In [35]:
```
# 3. Máte dva seznamy. Najděte společné prvky mezi nimi a uložte je do nc
seznam1 = [1, 2, 3, 4, 5]
seznam2 = [3, 4, 5, 6, 7]
seznam3 = []
for x in seznam1:
    if x in seznam2:
        seznam3.append(x)
```

In [36]:
```
seznam3
```

Out[36]: [3, 4, 5]

In [38]:
```
list(set([1, 2, 3, 4, 5]).intersection(set([3, 4, 5, 6, 7])))
```

Out[38]: [3, 4, 5]

In [40]:
```
# 4. Napište program, který vezme seznam čísel a vytiskne pouze sudá čísl
seznam = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
for cislo in seznam:
    if (cislo % 2) == 0:
        print(cislo)
```

```
2
4
6
8
10
```

In [41]:
```
# 5. Máte seznam slov. Najděte a vypište nejdelší slovo v seznamu.
slova = ["jablko", "hory", "kočka", "programování", "knihovna"]
nejdelsi_slovo = ""
for slovo in slova:
    if len(nejdelsi_slovo) < len(slovo):
        nejdelsi_slovo = slovo
print(nejdelsi_slovo)
```

```
programování
```

In [ ]:
```
# 5. Máte seznam slov. Najděte a vypište nejdelší slovo v seznamu.
slova = ["jablko", "hory", "kočka", "programování", "knihovna"]
nejdelsi_slovo = ""
for slovo in slova:
    if len(nejdelsi_slovo) < len(slovo):
        nejdelsi_slovo == slovo
print(nejdelsi_slovo)
```

In [ ]:

In [ ]:

In [25]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.
```

In [26]:
```
dir(print)
```

Out[26]:
```
['__call__',
 '__class__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__name__',
 '__ne__',
 '__new__',
 '__qualname__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__self__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__text_signature__']
```

In [27]:
```
a = "RetezeC"
```

In [28]:
```
a.lower()
```

Out[28]:
```
'retezec'
```

In [29]:
```
dir(a)
```

Out[29]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmod__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'capitalize',
 'casefold',
 'center',
 'count',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
 'format_map',
 'index',
 'isalnum',
 'isalpha',
 'isascii',
 'isdecimal',
 'isdigit',
 'isidentifier',
 'islower',
 'isnumeric',
 'isprintable',
 'isspace',
 'istitle',
 'isupper',
 'join',
 'ljust',
 'lower',
```

```
'lstrip',
'maketrans',
'partition',
'removeprefix',
'removesuffix',
'replace',
'rfind',
'rindex',
'rjust',
'rpartition',
'rsplit',
'rstrip',
'split',
'splitlines',
'startswith',
'strip',
'swapcase',
'title',
'translate',
'upper',
'zfill']
```

Prestavka do 19:00

In [ ]:

In [ ]:

In [ ]:

In [ ]:

- https://pythontutor.com/visualize.html#mode=edit
- https://projecteuler.net/archives
- https://www.kaggle.com/learn/intro-to-programming
- https://stackoverflow.com/
- https://chat.openai.com/
- https://krython.vnovak.cz/
- https://docs.python-guide.org/
- https://www.umimeinformatiku.cz/programovani-v-pythonu#ps595
- https://ucimeseit.cz/python-priklady/
- https://logickemysleni.cz/priklady-k-pythonu-zakladni-prikazy/
- https://kam.fit.cvut.cz/bi-pyt/tutorials/index.html

S tim co už ale umíte si můžem společně postavit malý jazykový model - co je ale ten jazykový model

In [42]:
```
# Things have fallen out, sir, so unluckily
# That we have had no time to move our daughter

last_word = "have"
print(last_word)

for _ in range(5):
    if last_word == "have":
        next_word = "fallen"
    elif last_word == "fallen":
        next_word = "out"
    else:
        break
    last_word = next_word
    print(next_word)
```

```
have
fallen
out
```

In [51]:
```
text = """Things have fallen out, sir, so unluckily
That we have had no time to move our daughter.
Look you, she lov'd her kinsman Tybalt dearly,
And so did I. Well, we were born to die.
'Tis very late; she'll not come down tonight.
I promise you, but for your company,
I would have been abed an hour ago.

These times of woe afford no tune to woo.
Madam, good night. Commend me to your daughter.

I will, and know her mind early tomorrow;
Tonight she's mew'd up to her heaviness.

Sir Paris, I will make a desperate tender
Of my child's love. I think she will be rul'd
In all respects by me; nay more, I doubt it not.
Wife, go you to her ere you go to bed,
Acquaint her here of my son Paris' love,
And bid her, mark you me, on Wednesday next,
But, soft, what day is this?

Monday, my lord.

Monday! Ha, ha! Well, Wednesday is too soon,
A Thursday let it be; a Thursday, tell her,
She shall be married to this noble earl.
Will you be ready? Do you like this haste?
We'll keep no great ado,—a friend or two,
For, hark you, Tybalt being slain so late,
It may be thought we held him carelessly,
```

```
Being our kinsman, if we revel much.
Therefore we'll have some half a dozen friends,
And there an end. But what say you to Thursday?

My lord, I would that Thursday were tomorrow.

Well, get you gone. A Thursday be it then.
Go you to Juliet ere you go to bed,
Prepare her, wife, against this wedding day.
Farewell, my lord.—Light to my chamber, ho!
Afore me, it is so very very late that we
May call it early by and by. Good night.

Wilt thou be gone? It is not yet near day.
It was the nightingale, and not the lark,
That pierc'd the fearful hollow of thine ear;
Nightly she sings on yond pomegranate tree.
Believe me, love, it was the nightingale.

It was the lark, the herald of the morn,
No nightingale. Look, love, what envious streaks
Do lace the severing clouds in yonder east.
Night's candles are burnt out, and jocund day
Stands tiptoe on the misty mountain tops.
I must be gone and live, or stay and die.

Yond light is not daylight, I know it, I.
It is some meteor that the sun exhales
To be to thee this night a torchbearer
And light thee on thy way to Mantua.
Therefore stay yet, thou need'st not to be gone.

Let me be ta'en, let me be put to death,
I am content, so thou wilt have it so.
I'll say yon grey is not the morning's eye,
'Tis but the pale reflex of Cynthia's brow.
Nor that is not the lark whose notes do beat
The vaulty heaven so high above our heads.
I have more care to stay than will to go.
Come, death, and welcome. Juliet wills it so.
How is't, my soul? Let's talk. It is not day.

It is, it is! Hie hence, be gone, away.
It is the lark that sings so out of tune,
Straining harsh discords and unpleasing sharps.
Some say the lark makes sweet division;
This doth not so, for she divideth us.
Some say the lark and loathed toad change eyes.
O, now I would they had chang'd voices too,
Since arm from arm that voice doth us affray,
Hunting thee hence with hunt's-up to the day.
O now be gone, more light and light it grows."""
```

a -> a=1, b=4, c=3, d=4, h=10, z h -> o=10 o -> j

In [ ]:
```
a = {"klic": {"klic2": 3}}
```

In [46]:
```
alphabet = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

In [49]:
```
language_model = {}
for pismenko1 in alphabet:
    language_model[pismenko1] = {}
    for pismenko2 in alphabet:
        language_model[pismenko1][pismenko2] = 0
```

In [55]:
```
for pismenko_aktualni in text:
    for pismenko_nasledujici in text[1:]:
        if pismenko_aktualni in alphabet and pismenko_nasledujici in alph
            language_model[pismenko_aktualni][pismenko_nasledujici] = lan
```

In [57]:
```
max_key = ""
max_value = 0
for key, value in language_model["a"].items():
    if max_value < value:
        max_key = key
language_model["a"]
```

Out[57]:
```
{'a': 27889,
 'b': 6012,
 'c': 4509,
 'd': 16533,
 'e': 45257,
 'f': 4676,
 'g': 9018,
 'h': 23213,
 'i': 20708,
 'j': 167,
 'k': 3674,
 'l': 18203,
 'm': 9352,
 'n': 22879,
 'o': 34068,
 'p': 2672,
 'q': 167,
 'r': 19539,
 's': 20708,
 't': 34235,
 'u': 11356,
 'v': 4676,
 'w': 7849,
 'x': 501,
 'y': 13193,
 'z': 167,
 'A': 1336,
 'B': 668,
 'C': 501,
 'D': 334,
 'E': 0,
 'F': 334,
 'G': 334,
 'H': 668,
 'I': 4008,
 'J': 334,
 'K': 0,
 'L': 835,
 'M': 1002,
 'N': 668,
 'O': 501,
 'P': 501,
 'Q': 0,
 'R': 0,
 'S': 1169,
 'T': 3006,
 'U': 0,
 'V': 0,
 'W': 1503,
 'X': 0,
 'Y': 167,
 'Z': 0}
```

In [62]:
```
pismenko = "Z"
print(pismenko, end="")
for _ in range(10):
    # nahodne vybrali pismeno z language_model ktery ma nejvetsi zastoupe
    max_key = ""
    max_value = 0
    for key, value in language_model[pismenko].items():
```

```
        if max_value < value:
            max_key = key
            max_value = value
    pismenko = max_key
    print(pismenko, end="")
```

```
ZZZZZZZZZZZ
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Programování je a postupného hledání řešení

Comprehensions

In [89]:
```
# nahrada za for smycku pri urcitych podminkach - vytvaret pouze jedno po
# kompaktni zapis
# obecne rychlejsi, prakticky jak kdy
# Comprehensions == vzdy vytvrareni nove struktury (napr. seznam)
```

In [67]:
```
l = []
for zastupce in range(10):
    l.append(zastupce * 2)
l
```

Out[67]:
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

In [ ]:
```
l = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

In [68]:
```
l = [zastupce * 2 for zastupce in range(10)]
```

```
l
```

Out[68]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

In [70]:
```
list_1 = []
list_2 = []
for zastupce in range(10):
    list_1.append(zastupce * 2)
    list_2.append(zastupce * 3)
```

In [71]:
```
list_1
```

Out[71]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

In [72]:
```
list_2
```

Out[72]: [0, 3, 6, 9, 12, 15, 18, 21, 24, 27]

In [75]:
```
[zastupce * 2 for zastupce in range(10)]
```

Out[75]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

In [76]:
```
[zastupce * 3 for zastupce in range(10)]
```

Out[76]: [0, 3, 6, 9, 12, 15, 18, 21, 24, 27]

In [78]:
```
l = []
for zastupce in range(10):
    if zastupce % 2 == 0:
        l.append(zastupce * 2)
l
```

Out[78]: [0, 4, 8, 12, 16]

In [79]:
```
[zastupce * 2 for zastupce in range(10) if zastupce % 2 == 0]
```

Out[79]: [0, 4, 8, 12, 16]

In [80]:
```
list_1 = []
list_2 = []
for zastupce in range(10):
    if zastupce % 2 == 0:
        list_1.append(zastupce * 2)
    else:
        list_2.append(zastupce * 3)
```

In [81]:
```
[zastupce * 2 for zastupce in range(10) if zastupce % 2 == 0]
```

Out[81]: [0, 4, 8, 12, 16]

In [82]:
```
[zastupce * 3 for zastupce in range(10) if zastupce % 2 == 1]
```

Out[82]: [3, 9, 15, 21, 27]

In [83]:
```
list_1 = []
for zastupce in range(10):
    if zastupce % 2 == 0:
```

```
        list_1.append(zastupce * 2)
    else:
        list_1.append(zastupce * 3)
list_1
```

Out[83]: [0, 3, 4, 9, 8, 15, 12, 21, 16, 27]

In [84]:
```
[zastupce * 2 if zastupce % 2 == 0 else zastupce * 3 for zastupce in rang
```

Out[84]: [0, 3, 4, 9, 8, 15, 12, 21, 16, 27]

In [88]:
```
kamarad = False
"ahoj" if kamarad else "dobry den"
```

Out[88]: 'dobry den'

In [90]:
```
a = [zastupce * 2 for zastupce in range(10)]
```

In [91]:
```
a
```

Out[91]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

In [ ]:
```
a.remove(10)
a.remove(12)
a.remove(14)
a.remove(16)
a.remove(18)
a # puvodni list
```

In [ ]:
```
a = [zastupce * 2 for zastupce in range(10)]
```

In [93]:
```
[zastupce for zastupce in a if zastupce < 10] # novy list
```

Out[93]: [0, 2, 4, 6, 8]

In [94]:
```
a
```

Out[94]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

In [ ]:
```
a = [zastupce for zastupce in a if zastupce < 10] # novy list
```

In [ ]:
```
# list
# slovnik
# mnozinu
```

Pauza do 20:20

In [96]:
```
type({})
```

Out[96]: dict

In [99]:
```
d = {}
for klic in range(10):
    hodnota = klic * 2
```

```
    d[klic] = hodnota
d
```

Out[99]: {0: 0, 1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}

In [97]:
```
{klic: klic * 2 for klic in range(10)}
```

Out[97]: {0: 0, 1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}

In [ ]:

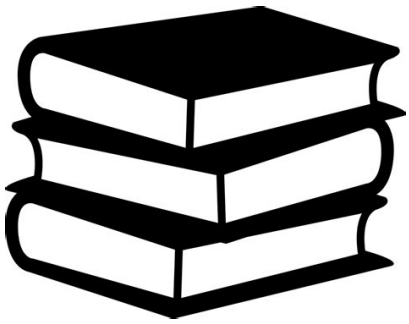
In [104…
```
d = set()
for zastupce in range(10):
    d.add(zastupce)
d
```

Out[104… {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

In [103…
```
{zastupce for zastupce in range(10)}
```

Out[103… {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

In [ ]:



Iterační protokol, podruhé

In [ ]:
```
for prvek in pole:
    # na kazdem prvku neco delej

    # ukonceni - dojdeme do konce pole a nebo break
```

In [ ]:
```
while plati_podminka:
    # delej neco
```

Pro periodické opakování ohlášení existují tzv. *iterační protokoly* (příp. označovány jako *smyčky*, *cykly*, *loopy*).

Pomocí smyčky *for* umíš zapsat takovou *iteraci*, kdy postupně projdeš **všechny hodnoty**.

Co když budeš potřebovat *iterovat* bez zadané hodnoty, ale za jistých podmínek?

Až bude mít `list` 3 hodnoty, dokud uživatel zadává vstupy, atd.

Potom bude potřeba, povědět si ještě o druhém typu smyček:

1. smyčka `for`,
2. smyčka `while`.

While smyčka

In [ ]:
```
for zastupna_promenna in iterovatelny_list:
    # blok prace
```

In [ ]:
```
while PODMINKA:
    # blok prace
    # ...
    # posledni radek -> vyhodnoceni PODMINKA
# blok po while
```

In [ ]:
```
while dokud_zasobnik_je_prazdny:
    vezmi_log
    zkontroluj
    pripadne prijde do zasobniku
```

Někdy ale není nutné *iterovat* přes **celý objekt**, jak tomu bylo u smyčky `for`.

Naopak, budeš potřebovat provádět proces *iterování* tak dlouho, *dokud* to bude nutné.

Za takovým účelem můžeš využít druhý typ smyček, `while`.

Obecně while loop

In [105…
```
for index in range(1, 6):
    print("Ještě nemáš 6, ale ", index, ", pokračuji..", sep="")
print("Hotovo, máš 6!")
```

```
Ještě nemáš 6, ale 1, pokračuji..
Ještě nemáš 6, ale 2, pokračuji..
Ještě nemáš 6, ale 3, pokračuji..
Ještě nemáš 6, ale 4, pokračuji..
Ještě nemáš 6, ale 5, pokračuji..
Hotovo, máš 6!
```

In [110…
```
index = 1

while index < 6:
    print("Ještě nemáš 6, ale ", index, ", pokračuji..", sep="")
    index += 1 # index = index + 1

print("Hotovo, máš 6!")
```

```
Ještě nemáš 6, ale 1, pokračuji..
Ještě nemáš 6, ale 2, pokračuji..
Ještě nemáš 6, ale 3, pokračuji..
Ještě nemáš 6, ale 4, pokračuji..
Ještě nemáš 6, ale 5, pokračuji..
Hotovo, máš 6!
```

In [107…
```
index
```

Out[107… 6

In [108…
```
index < 6
```

Out[108… False

1. `while` je **klíčkové slovo** v záhlaví,
2. `index < 6` je **podmínka**. Pokud je vyhodnocená jako `True`, proveď *odsazené ohlášení*,
3. `index < 6` ... `False`, ukonči smyčku a pokračuj **s neodsazeným zápisem** pod smyčkou,
4. `:` řádek s předpisem musí být **zakončený dvojtečkou**,
5. `print("Ještě nemáš...")`, následují *odsazené ohlášení*, které se budou opakovat v každém kroku,
6. `print("Hotovo, " ... )`, pokračuje *neodsazený zápis*, pod smyčkou.

In [ ]:
```
index = 1

while index < 6:
    print("Ještě nemáš 6, ale ", index, ", pokračuji..", sep="")
    index += 1 # index = index + 1

print("Hotovo, máš 6!")
```

While s doplňující podmínkou

Cyklus `while` samotný podmínku obsahuje. Určitě je ale možnost, tento podmínkový strom ještě rozšířit:

In [111…
```
True and True
```

Out[111… True

In [112…
```
True and False
```

Out[112… False

In [113…
```
True or True
```

Out[113… True

In [114…
```
True or False
```

Out[114… True

In [115…
```
# https://cs.wikipedia.org/wiki/Tabulka_pravdivostn%C3%ADch_hodnot_z%C3%A
```

In [ ]:
```
index = 0

while index <= 20:
    if len(str(index)) != 2:
        index = index + 1
    else:
        print(index)
        index = index + 1
```

Takové rozšíření může být obzvlášť přínosné, pokud podmínku v předpise nelze jednodušše rozšířit:

In [ ]:
```
index = 0

while index < 20 and len(str(index)) == 2:
    print(index)
    index = index + 1
```

In [123…
```
index = 0
stack = []
logs = list(range(10, 1000))

while input("continue?:") == "y" and index < len(logs):
    while len(stack) < 64 and index < len(logs):
        if logs[index] % 3 == 0: # podminka pridani
            stack.append(logs[index])
        index += 1
    stack = []
    print(index)
```

```
192
384
576
768
960
990
```

In [119…
```
len(stack)
```

Out[119… 64

In [130…
```
zadane_heslo = input("zadej heslo")
while zadane_heslo != "admin":
    zadane_heslo = input("zadej heslo")
print("Jsi admin")
```

```
Jsi admin
```

While/else

Cyklus `while` lze rozšířit o podmínkovou větev `else` (podobně jako *for loop*).

K ní se *interpret* dostane, pokud je podmínka v předpisu vyhodnocená jako `False`.

Současně nesmí narazit na ohlášení `break`:

In [131…
```
index = 0

while index < 20:
    if len(str(index)) != 2:
        index = index + 1

    else:
        print(index)
        index = index + 1

else:
    print("-" * 23, "Podmínka -> False".center(23), "-" * 23, sep="\n")

print(">Pokračuji pod smyčkou<")
```

```
10
11
12
13
14
15
16
17
18
19
-----------------------
   Podmínka -> False
-----------------------
>Pokračuji pod smyčkou<
```

Pokud doplníš ohlášení `break`, *interpret* přeskočí nejenom zbytek smyčky *while* ale také větev `else`:

In [137…
```
index = 0

while index < 20:
    if len(str(index)) != 2:  # pokud není číselná hodnota ze dvou znaků
        index = index + 1

    else:
        print(index)
        index += 1
        break

else:
    print("-" * 23, "Podminka -> False".center(23), "-" * 23, sep="\n")

print(">Pokračuji pod smyčkou<")
```

```
10
>Pokračuji pod smyčkou<
```

Nekonečný while loop

Jednou z aplikací smyčky `while` je zápis tzv. *nekonečného cyklu*.

Obecně řečeno, že v případě **nekonečných smyček** můžeš potkat dva typy:

1. **řízené** nekonečné smyčky,
2. **neřízené** nekonečné smyčky.

Neřízené nekonečné smyčky

Ty mohou nastat v důsledku **špatného zápisu** `while` cyklu:

In [ ]:
```
index = 1

while index < 20:
    print(index)
    # neinkrementuji hodnotu v proměnné 'index'
    # .. hodnota je v každém kroce 1 a smyčka nekončí.
    # .. Ctrl + C ->
```

In [133…
```
for x in range(10):

```

```
  Cell In[133], line 2

    ^
SyntaxError: incomplete input
```

In [134…
```
for x in range(10):
    pass
```

In [135…
```
pass
```

In [136…
```
pass
```

In [ ]:
```
continue/break/pass
```

In [132…
```
index = 1

while True:
    print(index)
    if index == 20:
        break
    index += 1
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

poznámka. výše ukázaná varianta představuje tzv. *nežádoucí nekonečnou smyčku*, kde vznikla chyba **v odsazené části zápisu**.

Chyba ovšem může nastat i při **špatném ohlášení** v zadání smyčky *while*:

In [ ]:
```
index = 1

while index > 0:  # vyhodnocené ohlášení má stále hodnotu `True`
    print(index)
    index = index + 1
```

poznámka. výše ukázaná varianta představuje tzv. *nežádoucí nekonečnou smyčku*, kde vznikla chyba **ve špatně zapsané podmínce**.

In [ ]:
```
index = 1

while index > 0:  # vyhodnocené ohlášení má stále hodnotu `True`
    print(index)
index = index + 1
```

Řízené nekonečné smyčky

Nekonečný cyklus s `while` je možné formulovat jako *řádnou/žádoucí nekonečnou smyčku*:

In [ ]:
```
while True:
    # blok kodu
    if timeout:
        break
```

In [ ]:
```
while True:
    # blok kodu
    if timeout:
        break
else:
    print("Nikdy nenastanu")
```

In [ ]:

In [ ]:

In [ ]:
```
while True:
    uziv_vstup = input("Zapiš libovolný text [nebo 'q' pro ukončení]: ")

    if uziv_vstup == "q":
        break
    print(uziv_vstup.upper())

print("Ukončuji ukázku!")
```

In [ ]:
```
switch = True

while switch:
    uziv_vstup = input("Zapiš libovolný text [nebo 'q' pro ukončení]: ")

    if uziv_vstup == "q":
        switch = False
    print(uziv_vstup.capitalize())

print("Ukončuji ukázku!")
```

In [ ]:
```
switch = True

while switch:
    uziv_vstup = input("Zapiš libovolný text [nebo 'q' pro ukončení]: ")

    if uziv_vstup == "q":
        switch = False
    print(uziv_vstup.capitalize())
else:
    print("Konec while")

print("Ukončuji ukázku!")
```

In [ ]:
```
switch = True

while switch:
    uziv_vstup = input("Zapiš libovolný text [nebo 'q' pro ukončení]: ")

    if uziv_vstup == "q":
        switch = False
    else:
```

```
        print(uziv_vstup.capitalize())
else:
    print("Konec while")

print("Ukončuji ukázku!")
```

In [ ]:
```
for dummy_var in range(??)
    uziv_vstup = input("Zapiš libovolný text [nebo 'q' pro ukončení]: ")
```

In [ ]:
```
given_words = []
switch = True

while switch:
    uziv_vstup = input("Zapiš libovolný text [nebo 'q' pro ukončení]: ")

    if uziv_vstup == "q":
        switch = False
    else:
        given_words.append(uziv_vstup.capitalize())

print(given_words)
print("Ukončuji ukázku!")
```

No description has been provided for this image

Walrus operátor

Přiřazovací operátor nebo jinak *walrus operátor* je formulace, která je v Pythonu poměrně nová (3.8+).

Jde o zápis, který ti umožní **dva procesy**, při použití **jednoho operátoru**:

1. nejprve **hodnotu přiřadí** proměnné,
2. přímo ji použije.

Vytvoření hodnoty a uložení

In [124…
```
jmeno = "Matous"
```

In [125…
```
print(jmeno)
```

```
Matous
```

In [127…
```
print(jmeno = "Matous")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[127], line 1
----> 1 print(jmeno = "Matous")

TypeError: 'jmeno' is an invalid keyword argument for print()
```

In [126…
```
print(jmeno := "Matous")
```

```
Matous
```

V předchozí ukázce jde čistě **o vysvětlivku**.

Proměnné jinak nadále a přehledně zapis po jedné a pod sebe. :)

Praktické ukázky skutečného využití najdeš níže.

In [138…
```
vek = 15
if vek > 7:
    print("Pujdu do skoly")
```

```
Pujdu do skoly
```

In [139…
```
if (vek := 15) > 7:
    print("Pujdu do skoly")
```

```
Pujdu do skoly
```

In [140…
```
if vek = 15:
    print("Pujdu do skoly")
```

```
  Cell In[140], line 1
    if vek = 15:
       ^
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

Kombinace s podmínkou

In [141…
```
jmeno = input("Zapiš jméno: ".upper())

if jmeno == "Matouš":
    print("Toto je ", jmeno, sep="")
else:
    print("Tak ", jmeno, ", toho neznám.", sep="")
```

```
Tak petr, toho neznám.
```

Obzvlášť v kombinaci se **zabudovanými funkce** a *uživatelskými funkcemi* je nápomocný.

Zásadní je **doplnění kulatých závorek**, kterými *interpretu* zdůrazníš pořadí:

In [142…
```
TEXT = "Zapiš jméno: ".upper()

if (jmeno := input(TEXT)) == "Matouš":
    print("Toto je ", jmeno, sep="")
else:
    print("Tak ", jmeno, ", toho neznám.", sep="")
```

```
Tak petr, toho neznám.
```

Pokud zapomeneš kulaté závorky, ohlášení **nemusí logicky pracovat**:

In [143…
```
TEXT = "Zapiš jméno: ".upper()

if jmeno := input(TEXT) == "Matouš":
    print("Toto je ", jmeno, sep="")
else:
    print("Tak ", jmeno, ", toho neznám.", sep="")
```

```
Toto je True
```

Copak se stane:

1. *Interpret* nejprve uloží vstup do funkce `input()`,
2. následně vloženou hodnotu porovná,
3. výsledek (`True` / `False`) uloží do proměnné `jmeno`.

Kombinace s cyklem while

In [144…
```
given_words = []
switch = True

while switch:
    uziv_vstup = input("Zapiš libovolný text [nebo 'q' pro ukončení]: ")

    if uziv_vstup == "q":
        switch = False
    else:
        given_words.append(uziv_vstup.capitalize())

print(given_words)
print("Ukončuji ukázku!")
```

```
['A', 'B']
Ukončuji ukázku!
```

In [145…
```
TEXT = "Zapiš libovolný text [nebo 'q' pro ukončení]: "

vstup = input(TEXT)
while vstup != "q":
    print(vstup)
    vstup = input(TEXT)
```

```
else:
    print(vstup, "Konec smyčky!", sep="\n")
```

```
a
b
c
d
q
Konec smyčky!
```

In [146…

```
TEXT = "Zapiš libovolný text [nebo 'q' pro ukončení]: "

while (vstup := input(TEXT)) != "q":
    print(vstup)
else:
    print(vstup, "Konec smyčky!", sep="\n")
```

```
a
b
c
d
q
Konec smyčky!
```

Analogicky můžeš opsat celý postup také **bez přiřazovacího operátoru**:

Pokud ti tedy dovede operátor `:=` šikovně pomoct, **určitě jej využij**.

Není nutné jej **zneužívat** v situacích, kdy je zápis málo čitelný, nebo špatně pochopitelný.

Opatrně na **verze**.

Pokud vyvíjíš na jiném prostředí, než produkčním, můžeš zjistit, že **má starší verzi Pythonu** a *walrus* nemusí podporovat.

Zkrácené přiřazování

Jde o **kratší způsob** pro úpravu hodnoty v existující proměnné.

Efektivnější není jen způsob zápisu, ale také způsob zpracování.

Doposud znáš tento zápis:

1. **Vytvoříš hodnotu** v proměnné `x`,
2. **upravíš hodnoty** v proměnné `x`,
3. **použiješ** novou hodnotu `x`.

In [ ]:
```
x = 2
```

In [ ]:
```
x = x + 3
```

In [ ]:
```
print(x)
```

Augmented assignment

Zkrácená varianta vypadá tak, že původní proměnnou `x` nepoužiješ a aritmetický operátor přesuneš přes rovnítko:

In [ ]:
```
x = 2
```

In [ ]:
```
x += 2
```

In [ ]:
```
print(x)
```

In [ ]:
```
y = 2
x += y
-=
*=
/=
```

Rozdíl

Pro tebe, jako uživatele, je tento zápis pouze o něco kratší.

Vypadá odlišně, jinak je stejně zapsaný pomocí 3 řádků.

V čem je tedy lepší?

Lepší je z hlediska využití paměti.

Klasický zápis rozdělený na jednotlivé kroky:

1. Vytvoříš **novou hodnotu** a **uložíš ji** do proměnné,
2. **načteš** původní hodnotu,
3. **zvětšíš ji** o hodnotu 4,
4. **uložíš** novou hodnotu,
5. **vypíšeš ji**.

In [ ]:
```
x = 2
```

In [ ]:
```
x = x + 4
```

In [ ]:
```
print(x)
```

Ve zkráceném zápise:

1. Vytvoříš **novou hodnotu** a **uložím ji** do proměnné,
2. **zvětšíš existující hodnotu**,
3. **vypíšeš ji**.

In [ ]:
```
x = 2
```

In [ ]:
```
x += 4
```

In [ ]:
```
print(x)
```

Některé zkrácené operátory

Původní operátor	Zkrácená varianta
`+`	`+=`
`-`	`-=`
`*`	`*=`
`/`	`/=`
`**`	`**=`

Souhrn úvodu smyček

Nyní máš za sebou stručný úvod do problematiky **iterátorů**. Jaké pojmy jsou tedy zásadní.

Iterable

Anglické ozn., které představuje **takový objekt**, který umí vytvořit *iterátor* (pomůcka zab. funkce `iter()`).

Iterator

Anglické ozn., které představuje tzv. **iterátor**. Tedy objekt, který dovede podávat jednotlivé hodnoty (pomocí funkce `next()`).

Iteration

Anglické ozn., které představuje proces **iterace**. Což je proces, který postupně prochází hodnoty. Krok za krokem.

For vs. while smyčka

Kdy máš vybrat co, lze popsat jako:

1. Pokud potřebuješ *iterovat* (~procházet) hodnotu od začátku do konce (tedy přes všechny hodnoty), použij `for`,
2. pokud potřebuješ *iterovat*, dokud platí nějaké kritérium, použij `while`.
