Ponořme se do Pythonu 3: http://diveintopython3.py.cz/index.html

1. ÚVOD

1.1 ÚVOD DO PROGRAMOVÁNÍ

Počítačové programování spočívá v umění donutit počítač, aby dělal to, co chcete aby dělal.

Počítačový program je sada instrukcí, která počítači říká, jak provést určitý úkol.

Jediný jazyk, kterému počítač rozumí, se nazývá **binární**. První programátoři museli opravdu binární kód zapisovat ručně. Tato činnost je známa jako programování ve **strojovém kódu** a je velmi obtížná. Dalším krokem bylo vytvoření **překladače**, který nahrazoval zápis binárních instrukcí zápisem využívajícím anglické ekvivalenty. Takže místo toho, aby si programátoři museli pamatovat, že kód `001273 05 04` znamená sečti `5 a 4` (anglicky `add 5 to 4`), mohli napsat `ADD 5 4`. Pro tento jazyk se používalo označení **jazyk symbolických instrukcí**.

I takový přístup se ukázal jako velmi primitivní. Pořád jste museli počítači říkat co má dělat na technické úrovni (co má provádět jeho procesor), jako například přesuň bajty z této oblasti paměti do jiné, sečti tento bajt s tímto, a podobně. Bylo to stále velmi složité a dokonce i vyřešení malých úkolů dalo hodně práce. Aby se programování zjednodušilo, byly postupně vyvinuty počítačové jazyky vyšší úrovně. V zásadě lze říci, že programátor píše program ve **vyšším programovacím jazyce**, srozumitelném pro programátora, a ten je přeložen na soubor bajtů, kterým rozumí počítač. Z technického pohledu programátor píše **zdrojový kód** a překladač z něj generuje **strojový kód**. Vyšší programovací jazyky se dají rozdělit do dvou skupin:

kompilované jazyky (C, Pascal, vizual Basic):

1. napíšu (třeba v text. editoru) **zdrojový text**
2. překladače ho **přeloží do strojového kódu**, uloží do souboru **(.exe**)
3. přeložený soubor jde spustit na jakémkoli počítači

interpretované jazyky (Basic, Python, Java, SQL, TCL, C#):

1. napíšu (třeba v text. editoru) **zdrojový text**
2. soubor uložím (v souladu s konvencí s příponou podle jazyka .py)
3. poklikáním na tento **spustím interpret** (Python.exe), který soubor načte, analyzuje a **provede**

Což je podstatně pomalejší. Pro urychlení spuštění programů v interpretovaných jazycích se používá toto: při načtení programu do interpretu se vytvoří tzv. **bytecode**, který se dá uložit do samostatného souboru a při dalším spouštění se použije místo zdrojového kódu tento bytecode-ový souor. Například programy v Javě se šíří v bytecodu.

Před mnoha lety přišel Edsgar Dijkstra s konceptem nazývaným **strukturované programování**. Tento koncept říká, že struktura všech programů může být vyjádřena pomocí následujících čtyř prostředků (stavebních prvků):

- Posloupnost instrukcí,
- cykly,
- větvení,
- moduly.

Kromě těchto strukturálních prvků potřebujeme doplnit několik dalších rysů, bez kterých by program ztrácel smysl:

- Data
- operace (sčítání, odčítání, porovnávání, atd.),
- schopnost vstupu a výstupu údajů (například výstup výsledků na displej).

Jakmile pochopíte tyto koncepty a to, jak se v určitém programovacím jazyce vyjadřují, jste schopni v tomto jazyce programovat.

1.2 PYTHON

K výuce programování jsem zvolila **objektově orientovaný jazyk Python**. Je jednoduchý, vhodný i pro začátečníky.

Download: třeba na www.python.org/download/

instalace: obvyklá

Vývojové prostředí

Programování v jakémkoliv vyšším programovacím jazyce znamená psaní **textového dokumentu** s důsledně dodržovanými **syntaktickými pravidly** daného programovacího jazyka, kde každá mezera, čárka, středník, tabulátor atd. má přesně definovanou funkci a nelze s nimi zacházet libovolně. Proto je psaní programů v textovém editoru sice možné, ale vhodné pro zkušeného programátora, který jazyk dokonale zná.

Pro nejisté začátečníky je užitečné psát programy ve **vývojovém prostředí**, které pomáhá s dodržováním přísných syntaktických pravidel a má i další pomocné funkce jako třeba testování programu a hledání chyb.

K jazyku Python existuje několik vývojových prostředí, jedno z nich - **IDLE** je součástí instalace Pythonu, nebo třeba PyScripter, který budeme používat. Tento si musíte stáhnout a nainstalovat zvlášť.

Spuštění IDLE: *Start/ Programy/Python xx/ IDLE*

Spuštění PyScripter: *Start/ Progamy/PyScipter/PyScripter for Python xx*

!pro psaní zdrojových kódů – nepoužívat diakritiku (angl. klávesnice)!

2. JEDNODUCHÉ DATOVÉ TYPY

Každý vyšší programovací jazyk dokáže pracovat s různými přesně definovanými typy dat. V některých programovacích jazycích (Pascal) se musí dopředu definovat, které proměnné budou jakého datového typu (deklarace proměnných). To proto, aby pro danou proměnnou bylo v operační paměti vyhrazeno optimální místo. V Pythonu definujeme datový typ proměnné prvním přiřazením dat.

Jednoduché datové typy, se kterými budeme v Pythonu pracovat:

- čísla celá, reálná (ve smyslu IEEE 754), komplexní
- řetězce
- boolean

2.1 ČÍSLA A OPERACE S ČÍSLY

Z pohledu Pythona rozdělujeme čísla do několika **datových** typů:

- **Celá čísla:** integer – nemají desetinnou část, rozsah od -MAXINT do +MAXINT (závisí na počtu bitů, které počítač používá na reprezentaci celého čísla, u 32 bitových je MAXINT něco kolem 2 mld.)
- **Reálná čísla:** float – v podstatě jsou to zlomky, oproti celým číslům mají větší rozsah, ale nejsou přesná: 3.5 je pro počítač i 3.49999999999999998

  ! desetinná čísla zapisujeme pomocí desetinné tečky, nikoliv pomocí desetinné čárky !
- **Komplexní čísla**

Aritmetické operace s čísly:

+	sčítání
-	odčítání
*	násobení
/	dělení (14/3 = 4.666666666666667)
//	celočíselné dělení (14/3 = 4)
**	umocňování (3**2 = 9)
%	zbytek po dělení (3%2 = 1)

Úkol: V intaktivním módu vyzkoušejte:
1) 12 + 4.3
*2) 123 * 4*
3) 9/2
4) 9.//2
5) 234%7

2.2 ŘETĚZCE A OPERACE S ŘETĚZCI

řetězec (string) - datový typ, který zahrnuje libovolnou posloupnost znaků. Znakem je i mezera.

Řetězec musí být zapsán v apostrofech, uvozovkách, nebo dlouhý řetězec v trojitých uvozovkách.

Př:

```
houby s voctem
"houby s voctem"
'rock'n'roll'
'houby s voctem'
rock'n'roll
"rock'n'roll"
```

```
6+3
"6+3"
'6+3'
"""Toto je dlouhý řetězec, obsahující
mnoho řádek textu"""
```

Operace s řetězci:

+	S1 + S2	slučování řetězců
*	S1 * 3	násobné opakování řetězce

Př:

```
'ale'+'ne'
"ale"*3
"ale"+"ne"*3
("ale"+"ne")*3
```

Python podporuje indexování řetězců. To znamená, že každý znak řetězce má svoje "pořadové číslo" počínaje číslem 0.

Př:

```
"alena"[0]              "a"
"alena"[1]...           "l"
```

Podporuje i indexování podřetězců:

"alena"[0:2] *vypíše první dva znaky*
"alena"[:3] *vypíše první 3 znaky (vynechání prvního indexu = 0)*
"alena"[2:] *vypíše od 3. znaku včetně do konce (vynecháme-li druhý index, dosadí konec řetězce)*

Úkol: zjistěte, jak se používají záporné indexy:

Řešení:

```
"lokomotiva"[-1]        poslední od konce
"lokomotiva"[-3]        třetí od konce
"lokomotiva"[-2:]       od -druhého do konce (poslední dva)
"lokomotiva"[:-2]       od 0-tého do -druhého (vše kromě posledních dvou)
```

Úkol: zjistěte co se stane, když použijete indexy sahající "mimo řetězec" (i u podřetězců).

Řešení:

```
"lokomotiva"[100]       chyba!
"lokomotiva"[0:100]     OK
"lokomotiva"[100:0]     OK – prázdný řetězec
```

2.3 DATOVÝ TYP BOOLEAN, LOGICKÉ OPERACE

Výrok: Jakýkoliv výraz, u nějž má smysl pravdivostní hodnota.

Př:

```
dřevěný stůl                    není výrok
dřevěný stůl je levný           je výrok
3 = 5                           je výrok
3 + 7 > 2                       je výrok
```

Datový typ boolean má pouze dvě hodnoty: **True** a **False**. Ty vyjadřují skutečnost, zda je něco pravdivé nebo nepravdivé. Datový typ boolean vzniká jako výsledek vyhodnocení pravdivostní hodnoty výroku.

Logické operace: (A a B jsou výroky)

and	A and B	Logický součin. True, když A i B jsou True. Jinak False.
or	A or B	Logický součet. True, když oba nebo jeden z A, B jsou True. Jinak False.
==	A == B	Rovnost. True, když A = B
!=	A != B	Nerovnost. True, když A různé od B
not()	not(A)	Negace. True, když A je false
in		„leží v“ (později – pro datový typ seznam)
not in		„neleží v“ (později – pro datový typ seznam)

Je tedy Python schopen vyhodnotit pravdivostní hodnotu výroku? Ano, pokud používáme datové typy a operace s nimi, kterým rozumí. A nejen to, Python přiřazuje pravdivostní hodnoty i číslům a řetězcům.

Úkol: Vyzkoušejte, jaké jsou pravdivostní hodnoty:

Řešení:

```
0                       False       Pouze 0
125                     True        Všechna čísla kromě 0
"lokomotiva"            True        Všechny neprázdné řetězce
""                      False       Pouze prázdný řetězec
```

Úkol: Určete pravdivostní hodnotu následujících výroků:
Výrok A: x < 5, Výrok B: y = 3, pro x = 0, y = 1

Řešení:

```
P(A)              True              P(A and B)              False
P(B)              False             P(A or B)               True
P(not(A))         False             P(A == B)               False
P(not(B))         True              P(A and not(B))         True
P(not(A and B))   True              P(not(A and B)or not(B))True
P("ano"=="ne")    False             P("ano"!="ne")          True
```

Výsledky ověřte v Pythonu s použitím proměnných A, B, x, y. Jakého datového typu jsou tyto proměnné?

Dú: Určete pravdivostní hodnotu výroků v předchozím cvičení pro x = 3, y = 3, a pro x = 7, y = 3. Výsledky ověřte v Pythonu s použitím proměnných A, B, x, y.

3. PROMĚNNÉ, VÝRAZY A PŘÍKAZY

3.1 PROMĚNNÉ

proměnná je jméno, které poukazuje na nějakou hodnotu. Proměnná získá svoji hodnotu **přiřazovacím příkazem**, kterým je **=**.

Např:

```
m = "ahoj"
x = 3
r = 3.1415926
A = x < 5
```

!!! prvním přiřazením hodnoty do proměnné definujeme její datový typ. S proměnnou můžeme provádět pouze ty operace,které jsou přípustné pro daný datový typ **!!!**

Pro jména proměnných můžeme vybírat jakákoliv slova, která třeba připomínají její hodnoty, nebo účel, pro který jsme danou proměnnou vytvořili. Kromě tzv. **klíčových slov**:

and	**continue**	**else**	**for**	**import**	**not**	**raise**
assert	**def**	**except**	**from**	**in**	**or**	**return**
break	**del**	**exec**	**global**	**is**	**pass**	**try**
class	**elif**	**finally**	**if**	**lambda**	**print**	**while**

Klíčové slovo je takové slovo, které má pro interpret nějaký speciální význam. Najděte a označte si v tabulce ta, která už znáte.

*Př: Vraťme se k datovým typům proměnných. Jaké jsou datové typy proměnných v předchozím příkladu? Datový typ lze ověřit pomocí funkce **type (proměnná),** nebo se v PyScripteru přepnete na kartu **Variables**, kde je seznam všech použitých poměnných i s jejich datovými typy.*

Řešení:

```
>>> type(m)
<type 'str'>
>>> type(x)
<type 'int'>
>>> type(r)
<type 'float'>
>>> type(A)
<type 'bool'>
```

3.2 VÝRAZY

Výraz je kombinací hodnot (konstant), proměných a operátorů. V interaktivním modu je po zapsání výrazu tento ihned vyhodnocen a zobrazí se jeho hodnota. Ve skriptu je každý výraz příkazem, ale nedělá nic.

Př:

```
1+1          je výraz
3<5          je výraz
x            je výraz
"ahoj"       je výraz
type (x)     je výraz
```

3.3 PŘÍKAZY

Příkaz je instrukce, kterou Pythonův interpret provede.

Příkazy, které už umíme:

=	přiřazení hodnoty do proměnné	nemá viditelný výstup
jakýkoliv výraz	vyhodnocení výrazu	nemá viditelný výstup

*Úkol 1: Proveďte zobrazení **hodnot** proměnných z předchozích příkladů pomocí funkce **print(proměnná).***

```
Řešení:
>>> print(A)
True
>>> print(x)
3
>>> print(r)
3.1415926
>>> print(m)
ahoj
```

Nyní přichází okamžik pro napsání vašeho prvního (zatím velmi primitivního) skriptu.

Program (skript)= posloupnost příkazů

Je obvyklé, do skriptů vpisovat **komentáře**, které autorovi pomáhají v orientaci. Komentáře interpret ignoruje, musíme mu ale naznačit, že se jedná o komentář. **Komentáře začínají znakem #.**

Postup psaní skriptů v Pyscriptu:

Postupujeme, jakobychom vytvářeli např. textový dokument ve Wordu. V hlavní nabídce vybereme položku *File/ New Python module* (nebo výběr tlačítka pro **vytvoření nového skriptu** z panelu nástrojů). Otevře se nová záložka v okně PyScripteru, do níž můžeme začít **psát posloupnost příkazů** pro interpret Pythona. Po dopsání skript **uložíme** (*File/ Save as* z hlavní nabídky) pod nějakým jménem a **spustíme** pomocí *Run/ Run* z hlavní nabídky nebo pomocí tlačítka Run (**ctrl+F9**). Výsledek skriptu uvidíme v interaktivním okně PyScipteru.

Skript .py se dá spustit i bez vývojového prostředí. Dvojklikem na skript v průzkumníkovi skript spustíme v příkazové řádce. Ale pozor, okamžitě po ukončení skriptu se příkazová řádka vypne, takže ani nestihneme přečíst výsledek. Je proto nutné na konec skryptu přidat čekání na reakci uživatele, která skript ukončí.

Např.:

```
print()                 #(znamená tisk prázdného řádku)
c=input('stisknete "Enter" pro ukonceni programu')
```

Úkol 2: Napište skript, který vytiskne postupně číslice 1 až 5 do sloupce pod sebou.

Řešení:

```
# skript na tisk ciselne rady
print(1)
print(2)
print(3)
print(4)
print(5)
```

Př: Vytvořte skript, který vytiskne dny v týdnu.

4. PŘÍKAZY PRO ŘÍZENÍ TOKU PROGRAMU

4.1 CYKLUS S PEVNÝM POČTEM OPAKOVÁNÍ

Pro jednu proměnnou je možné použít několik přiřazovacích příkazů, například:

```
a = 7
a = 8
a = 9
a = "ne"
a = "ano"
```

Každý následující příkaz přemaže předchozí hodnotu proměnné. (Dokonce 4. příkaz změní i její datový typ.)

Vzpomeňte na skript tisku čísel 1 až 5. Mohl vypadat takto:

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

nebo

```
x=1
print(x)
x=2
print(x)
x=3
print(x)
x=4
print(x)
x=5
```

Co kdybychom potřebovali tisk třeba 1000 čísel? Tyto dva způsoby by byly moc pracné, potřebovali bychom něco jako:

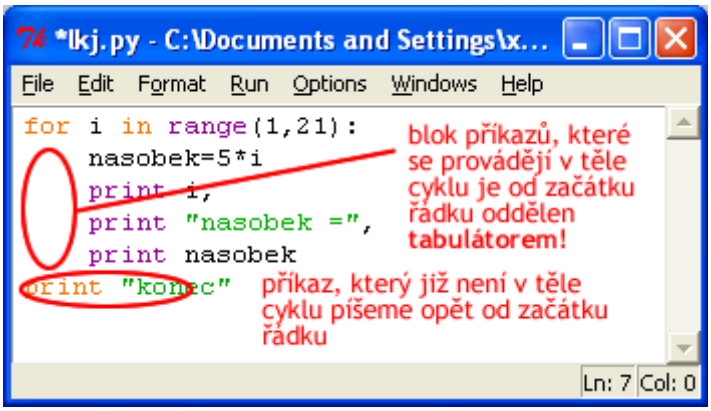
Pro x rovno postupně 1 až 5 tiskni x

Programovací jazyky pro tyto případy nabízejí **cyklus s pevným počtem opakování** neboli **for cyklus**:

```
for i in a:
    blok příkazů
```

i - řídící proměnná
a – řídící sekvence

Příkaz pro for cyklus zabírá minimálně dva řádky. Přičemž druhý (případně i další) řádek - příkaz v těle cyklu je **odsazený tabulátorem**. Příkazy, které již do těla cyklu nepatří jsou opět v textu na stejné úrovni odsazení jako slovo **for**. **Tabulátor tedy hraje významnou roli při strukturování skriptů!**



Jako řídící sekvenci lze použít:

- posloupnost čísel (používáme funkci range, např.: for i in range(1,21))
- posloupnost znaků v řetězc
- posloupnost prvků seznamu

Užitečné funkce a příkazy

funkce **range(z,k,kr)**
Výstupem je aritmetická posloupnost (datový typ seznam) od z po k-1 s krokem kr.

funkce **len(r)**
Výstupem je celé číslo udávající délku (počet znaků) v řetězci r

příkaz **break** ukončí for cyklus

Úkol 1: Napište skript, který tiskne posloupnost 20 čísel, které jsou násobky pěti. Uložte pod názvem ***nasobky.py****.*

Řešení: (tisk do sloupce)

```
for i in range(1,21):
      nasobek = 5*i
      print nasobek
```

tisk do řádku (přidáním čárky za poslední položku příkazu print)

```
for i in range(1,21):
      nasobek = 5*i
      print nasobek,
```

Úkol 2: Napište skript, který tiskne posloupnost jednotlivých znaků slova "automobil". Uložte pod názvem ***automobil.py****.*

Řešení:

```
r="automobil"
for i in r:
      print i
```

Úkol 3: Napište skript pod názvem ***soucet.py****, který bude mít následující výstup:*
rada cisel je: 1 2 3 4 5
jeji soucet je: 15

Řešení: soucet.py

```
# soucet nekolika prvnich cisel posloupnosti
souc=0
print ("rada cisel je:",)
for i in range(1,6):
    print (i,)
    souc=souc+i
print()
print ("jeji soucet je:",)
print (souc)
```

Úkol 4: Napište skript pod názvem ***retezec.py****, který bude mít následující výstup:*
slovo abeceda se sklada z techto pismen:
1 . znak je: a
2 . znak je: b
3 . znak je: e
4 . znak je: c
5 . znak je: e
6 . znak je: d
7 . znak je: a

Řešení: retezec.py

```
#vypis znaku retezce po jednom
ret="abeceda"
print
print ("slovo",ret,"se sklada z techto pismen:")
for i in range(0,len(ret)):
    print (i+1,".","znak je:",ret[i])
```

4.2 ZADÁVÁNÍ DAT KLÁVESNICÍ

Je načase, abychom začali s počítačem konverzovat. Sami cítíte, ze by bylo lepší, například v předchozím úkolu, abychom slovo, které má počítač po hláskách zobrazit na monitor, sami na jeho výzvu zadali. K tomu slouží:

Užitečné funkce:

input(”text, který se má zobrazit na monitor”)

int(r) - převádí řetězec nebo reálné číslo na přirozené číslo

Používají se takto:

```
vstup = input("zadejte něco:")
```

Do proměnné `vstup` se přiřadí **řetězec**, který zadáme z klávesnice po přečtení výzvy "zadejte něco:".

*Úkol 1: Předchozí skript retezec.py upravte tak, abyste slovo, jež má počítač vypsat po hláskách, zadávali z klávesnice. Uložte pod názvem **hlaskovani.py.***

řešení: hlaskovani.py

```
#vypis znaku retezce po jednom
ret=input("zadejte libovolne slovo:" )
print()
print ("retezec",ret,"se sklada z techto znaku:")
for i in range(0,len(ret)):
    print (i+1,".","znak je:",ret[i])
```

Pokud potřebujeme z klávesnice **načíst celé číslo**, musíme použít kombinaci funkcí **input()** a **int()**:

*Úkol 2: Napište skript, který ze zadaného čísla spočítá faktoriál. Uložte pod názvem **faktorial.py**.*

řešení: faktorial.py

```
a=input('zadej prirozene cislo:')
a=int(a)
for i in range ((a-1), 1,-1):
    a= a*i
print('faktorial= ',a)
```

4.3 VĚTVENÍ (PODMÍNĚNÝ PŘÍKAZ)

Abychom mohli vytvářet užitečné programy, musíme mít možnost naprogramovat rozhodování. Tím myslím třeba toto:

Jestli je zadané heslo správné, pak otevři aplikaci, jinak vytiskni zprávu: "Máte špatné heslo."

Další z řady příkazů – větvení vychází jak jinak než z angličtiny.

Syntaxe

```
if řídící podmínka:
     blok příkazů
else:
     blok příkazů
```

Nutno ještě uvést porovnávací operátory pro řídící podmínky. Jsou to:

<	menší než
>	větší než
==	rovno
<=	menší, rovno

<	menší než
>=	větší, rovno
!=	nerovno

Toto je nejobvyklejší větvení, má dvě větve. Jedna obsahuje blok příkazů pro případ, že je podmínka splněna, druhá větev obsahuje blok příkazů pro případ, že podmínka splněna není. Větví může být v podstatě libovolné množství, musíme ale upravit syntaxi:

Např: **Podmíněný příkaz s jednou větví**. Jestli je *x* rovno nule, tiskni "nula". (Jestli *x* není rovno nule, neděje se nic.)

```
if x=0"
      print "nula"
```

Např: **Podmíněný příkaz s více než dvěma větvemi**. Jestli je *a* rovno 1, počítej obsah čtverce. Jestli je *a* rovno 2, počítej obsah kruhu. Jestli je *a* rovno 3, počítej obsah trojúhelníku.

```
if a = 1:
      počítej obsah čtverce
elif a= 2:
      počítej obsah kruhu
elif a= 3:
      počítej obsah trojúhelníku
```

?? Co když *a* = 4 ??

Užitečné funkce:

funkce: **int(c)**
c je libovolné číslo, funkce vrací celou část z c. Funkce se používá také vzápětí po funkci input(c), chceme-li vynutit převod vkládaného řetězce na celé číslo.

funkce: **abs(z)**
z je celé číslo, funkce vrací jeho absolutní hodnotu.

Úkol 1: Upravte skript na výpočet faktoriálu tak, aby v případě, že místo přirozeného čísla zadáte číslo záporné, nebo desetinné, výstupem bylo hlášení "zadané číslo nebylo přirozené".

řešení: faktorial.py

```
a=input('zadej prirozene cislo:')
a=int(a)                                  #převod řetězce a na celé číslo
if a==abs(a):
    for i in range((a-1),1,-1):
        a= a*i
    print( 'faktorial= ',a)
else:
    print('bohuzel, zadane cislo nebylo prirozene')
```

*Úkol 2: Napište skript, na jehož vstupu bude libovolné číslo, výstupem bude informace, zda číslo je záporné, kladné, nebo rovno nule. Uložte pod názvem **kladne_zaporne.py**.*

řešení: kladne_zaporne.py

```
cislo=input("zadejte cislo:")
cislo=float(cislo)                        #převod řetězce a na reálné číslo
if cislo>0:
    print("zadane cislo je kladne")
elif cislo==0:
    print("zadane cislo je nula")
else:
    print("zadane cislo je zaporne")
```

*Úkol 3: Napište skript, na jehož vstupu bude libovolné číslo, výstupem bude informace, zda zadané číslo je sudé, nebo liché (případně špatně zadané). Uložte pod názvem **sude_liche.py**.*

řešení: sude_liche.py

```
cislo=input("zadejte cele cislo:")
cislo=int(cislo)
if cislo==abs(cislo):
    if cislo%2==0:
        print("cislo je sude")
    else:
        print("cislo je liche")
else:
    print("bohuzel nezadali jste prirozene cislo")
```

*Domácí úkol: Napište skript, který pro zadané přirozené číslo vypíše jeho dělitele, nebo oznámí, že se jedná o prvočíslo. Ošetřete chybu vstupu - uživatel nezadá přirozené číslo. Uložte pod názvem **prvocisla.py**.*

Domácí úkol: Vytvořte skript, který po zadání přirozeného čísla vypíše prvočísla menší než zadané číslo.
*Nápověda: při nalezení prvního dělitele přerušte for cyklus příkazem break. Uložte pod názvem **tab_prvocisel.py**.*

4.4 PODMÍNĚNÝ CYKLUS

neboli **while cyklus**. Hodí se v situacích, kdy chceme opakovat nějaké příkazy, ale na rozdíl od **for cyklu** nevíme kolikrát. Víme jen, že se mají opakovat, pokud je splněna nějaká podmínka, či naopak, dokud není splněna nějaká podmínka.

Např: Chcete jít ven, ale rodiče doma trvají na tom, že se musíte učit slovíčka tak dlouho, dokud se je nenaučíte. Kdybychom chtěli tuto situaci popsat v programovacím jazyce, vypadalo by to takto:

```
pokud neumíš slovíčka:
      uč se
jdi ven
```

syntakticky přesněji v Pythonu:

```
while neumíš slovíčka:
      uč se
jdi ven
```

Syntaxe

```
while řídící podmínka:
          blok příkazů
```

Přesný popis toku programu:

1. Interpret vyhodnotí řídící podmínku (jestli je **true** nebo **false**).
2. Jestli je **false**, **ukončí** while cyklus a pokračuje dalším příkazem.
3. Jestli je **true**, provede všechny příkazy v bloku příkazů a **vrací** se k bodu 1.

*Úkol 1: vytvořte skript, který žádá zadání hesla tak dlouho, dokud ho nezadáte správně. Uložte pod názvem **heslo.py***

řešení: heslo.py

```
password = "zatim_nic"
while password != "unicorn":
    password = input("Password:")
print("vitejte!")
```

*Úkol 2: Vytvořte skript, který vypíše Fibonacciho posloupnost do zadané horní meze. Uložte pod názvem **Fibonacci.py***

Řešení (první varianta s vícenásobným přířazením): Fibonacci.py

```
h=int(input("zadejte horni mez: "))
print()
a,b=0,1
while b<h:
```

```
        print b,
        a,b=b,b+a
```

nebo Fibonacci2.py

```
h=int(input("zadejte horni mez: "))
print()
a=0
b=1
while b<h:
    print(b)
    c=b
    b=b+a
    a=c
```

Užitečné příkazy

Pass: používá se na místě, kde by program ze syntaxe očekával blok příkazů k provedení, ale programátor žádnou akci provést nechce. Třeba v některé z větví **if** a **elif**.

Vícenásobné přiřazení: a, b = b, b+a , výrazy na pravé straně přiřazení jsou vyhodnoceny ještě před přiřazením, takže při výměně hodnot mezi proměnnými není třeba použít třetí proměnné.

Ošetření chyby vstupu

Jistě se vám několikrát stalo, že jste ve skriptu požadovali například vstup celého čísla, ale uživatel zadá číslo reálné, nebo řetězec znaků, který čísla neobsahuje. Běh programu skončí chybovým hlášením: **ValueError: invalid literal for int() with base 10: 'asd'**

Tuto situaci můžeme ošetřit pomocí vyjímky:

```
while True:
    try:
        x=int(input('zadejte cele cislo: '))
        break
    except ValueError:
        print('to nebylo cele cislo!')
print("uz je to dobre")
```

pro ošetření vstupu **přirozeného čísla** můžeme modifikovat:

```
while True:
    try:
        x=int(input('zadejte platbu v celych Kc: '))
        if x<1:
            pass
        else: break
    except ValueError:
        print('zadej spravne cislo!')
```

Vraťte se k řešení předchozích úkolů a upravte své skripty tak, aby jste měli ošetřené chyby vstupu z klávesnice.

*Úkol 3: Napište skript, který nabízí výpočet obsahu trojúhelníku, čtverce, kruhu, nebo ukončení programu. Uložte pod názvem **menu.py**.*

řešení: menu.py

```
# zakladni model pro dialog s uzivatelem pomoci menu
#vetveni if, elif, else


volba='nic'
while volba!='4':
    print("""
    Vyberte si z nasledujiciho seznamu:
    1) Trojuhelnik
    2) Ctverec
    3) Kruh
    4) Konec programu
    """)

    volba=input('Vase volba? (1, 2, 3, 4)')

    if volba=='1':#trojuhelnik
        v=float(input('zadejte vysku trojuhelniku:'))
        vc=float(input('zadejte zakladnu trojuhelniku:'))
        print( 'plocha trojuhelniku je:', 0.5*v*vc)

    elif volba=='2': #ctverec
        a=float(input('zadejte stranu ctverce:'))
        print( 'plocha ctverce je:', a*a)

    elif volba=='3':#kruh
        r=float(input('zadejte polomer kruhu:'))
        print( 'plocha kruhu je:', 3.14159*r*r)

    elif volba=='4': #konec programu
        pass

    else:
        print( 'neplatna volba')
```

4.5 VLASTNÍ FUNKCE

Programátor by se při psaní kódu měl držet zásady, že blok příkazů vztahující se k jedné činnosti by neměl přesahovat rozměr jedné obrazovky. V rozsáhlých programech se lehce ztratí přehled a nevyzná se v nich ani samotný programátor. Ke zkrácení a zpřehlednění programu slouží **funkce** a **moduly** (v další kapitole). Python obsahuje mnoho předdefinovaných (built-in) funkcí, s některými jsme se už setkali - int, abs, range... Programátor si může definovat i vlastní funkce. Na začátku programu si vytvoří jakousi hlavička programu, ve které nachystá (**definuje**) funkce a moduly, jež budeme v těle programu používat (**volat**).

Syntaxe

```
def jmeno_ funkce (n1, n2, ...nn):
    blok příkazů
```

kde n1 až nn jsou parametry funkce

v těle programu pak voláme tuto funkci jednoduše zápisem jejího jména a zadáním konkrétních parametrů.

*Úkol 1: Upravte svůj skript menu.py tak, že pro tisk nabídky a výpočty obsahů geom. objektů (spolu s tiskem výsledků) definujete vlastní funkce. Uložte jako **menu_fce.py**.*

Řešení (menu_fce.py):

```
# ####################hlavička - definice funkci#######################
def tisk_menu():
    print( """
    Vyberte si z nasledujiciho seznamu:
    1) Trojuhelnik
    2) Ctverec
    3) Kruh
    4) Konec programu
    """)
def troj(v,n):
    print( 'plocha trojuhelniku je:', 0.5*v*n)
def ctverec(a):
    print( 'plocha ctverce je:', a*a)

def kruh(r):
    print( 'obsah kruhu je: ',3.1415926*r*r)


# ######################telo programu#####################
volba='nic'
while volba<>'4':
    tisk_menu()
    volba=input('Vase volba? (1, 2, 3, 4)')

    if volba=='1':#trojuhelnik
        v=float(input('zadejte vysku trojuhelniku:'))
        vc=input('zadejte zakladnu trojuhelniku:')
        troj(v,vc)

    elif volba=='2': #ctverec
        a=float(input('zadejte stranu ctverce:'))
        ctverec(a)

    elif volba=='3':#kruh
        r=float(input('zadejte polomer kruhu:'))
        kruh(r)

    elif volba=='4': #konec programu
        pass

    else: # neplatna volba
        print( 'neplatna volba')
```

Chceme-li, aby naše vlastní funkce nejen něco prováděla, ale také „vracela“ hodnoty, je třeba v definici funkce použít příkazu **return**. Například definujeme funkci, která má vpočítat obsah kruhu:

```
def obkruh(r):
      return 3,1415926 * r * r
```

Funkce vrací hodnotu zapsanou za příkazem return. Tuto hodnotu pak můžeme v těle skriptu přiřadit do proměnné:

```
s = obkruh(6)
```

*Domácí úkol: Definujte vlastní funkci, jejíž návratovou hodnotou bude faktoriál zadaného čísla. Funkci použijte ve skriptu s názvem **fcefaktorial.py**.*

4.6 MODULY

Skript může být kvůli přehlednosti rozdělen do několika částí, které se uloží jako samostatné skripty. Těmto částem se říká **moduly.**

Zabudované moduly

Python má jako příslušenství velké množství už **zabudovaných modulů (time, math, sys, pickle...**), které obsahují mnoho užitečných funkcí. Pokud chceme využít těchto funkcí, musíme příslušný modul **importovat** do našeho skriptu pomocí příkazu:

```
import jméno modulu
```

potom, chci-li pužít konkrétní funkci z importovaného modulu jednoduše napíšu:

```
jméno modulu.jméno funkce
```

Například:

```
import  math
s=math.pi
```

v proměnné *s* nyní mám hodnotu konstanty pí.

Vlastní moduly

Moduly si můžeme vytvářet i sami. Pokud hlavní skript i moduly uložíme do stejného adresáře, můžeme bez komplikací importovat.

*Úkol 1: upravte svůj skript pro výpočet obsahů geom. útvarů tak, že ho rozdělíte na hlavní skript a modul obsahující všechny vámi definované funkce. Hlavní skript uložte pod názvem **menu_modul.py**, modul s vlastními funkcemi uložte pod názvem **modul_obsah.py**.*

Řešení: modul_obsah.py

```
"""modul  obsahujici funkce pro vypocet obsahu geom. utvaru"""
def  tisk_menu():
        print(  """
Vyberte  si z nasledujiciho seznamu:
        1)  Trojuhelnik
        2)  Ctverec
        3)  Kruh
        4)  Konec programu
""")

def  troj(v,n):
        print(  'plocha trojuhelniku je:', 0.5*v*n)

def  ctverec(a):
        print(  'plocha ctverce je:', a*a)

def  kruh(r):
        print(  'obsah kruhu je: ',3.1415926*r*r)
```

Řešení: použití v hlavním skriptu menu_obsah.py:

```
######################import  modulu#############
import  modul_obsah
######################telo  programu############
volba='nic'
while  volba<>'4':
    modul_obsah.tisk_menu()
    volba=input('Vase  volba? (1, 2, 3, 4)')

    if  volba=='1':#trojuhelnik
         v=float(input('zadejte  vysku trojuhelniku:'))
         vc=input('zadejte  zakladnu trojuhelniku:')
         modul_obsah.troj(v,vc)

    elif  volba=='2': #ctverec
         a=float(input('zadejte  stranu ctverce:'))
         modul_obsah.ctverec(a)

    elif  volba=='3':#kruh
         r=float(input('zadejte  polomer kruhu:'))
         modul_obsah.kruh(r)

    elif  volba=='4': #konec  programu
         pass

    else:  #  neplatna volba
         print(  'neplatna volba')
```

5. SLOŽENÉ DATOVÉ TYPY

Dosud známe jednoduché datové typy: celé číslo (integer), reálné číslo (float), řetězec (string) a boolean. Python podporuje i složené datové typy, což jsou seznam (list), sekvence (tuple), slovník (dictionary).

5.1 SEZNAM (LIST)

Seznam je výčet libovolných hodnot (kterým říkáme **prvky**) oddělených čárkou mezi hranatými závorkami. Jednotlivé hodnoty nemusí být stejného datového typu.

Např:

```
s=['a','lavice',12,'S.O.S',]
l=[9,12,103]
```

pak proměnné s a l jsou typu seznam, zkráceně - jsou seznamy.

Pro seznamy jsou definovány **operace** podobné jeko pro řetězce:

+	s + l	spojování seznamů
*	s * 3	násobné opakování seznamu

Stejně tak jako u řetězců je seznam indexovaný, podporuje i operace s podřetězci.

Vyzkoušejte a zapište výsledek každé operace (s a l jsou seznamy: s= ['a', 'lavice', 12, 'S.O.S'], l= [9, 12, 103]):

operace	výsledek operace	slovní popis
`s+l`		
`s*3`		
`s[1]`		
`s[-1]`		
`s[2]="meloun"`		
`s[1:3]`		
`s[:3]`		
`s[3:]`		
`len(s)`		
`s[len(s):]=["alfa", "beta","gama",36]`		
`del s[3]`		
`"beta" in s`		
`"omega" in s`		
`36 not in s`		
`26 not in s`		

Podstatný rozdíl mezi datovým typem řetězec a seznam je ten, že **řetězec je neměnný** (nemůžeme měnit jednotlivé znaky), kdežto **seznam je proměnný** (můžeme jednotlivé prvky měnit, mazat).

Domácí úkol: Vytvořte skript, pomocí něhož budete hrát s počítačem „Kámen, nůžky, papír“. Bude se vám hodit funkce ***choice(seznam)*** *z modulu* ***random****, která vrací náhodně hodnotu ze seznamu. Uložte jako* ***kamen_nuzky.py****.*

5.2 METODY NAD SEZNAMY

Metoda je **funkce**, která se **váže na určitý objekt**. Je-li tím objektem seznam, můžeme použít následující metody:

```
s a l jsou seznamy
s=['a','lavice',12,'S.O.S',]
l=[9,12,103]
```
s.append("Marek") `provede pridani prvku na konec seznamu`
s.extend(l) `provede přidani seznamu l na konec s`
s.count(12) `vrací počet výskytů prvku 12 v seznamu`
s.remove(103) `provede odstranění prvku 103`
s.pop(2) `provede odstranění prvku s indexem 2 a vraci hodnotu odstraněného prvku`
s.index("lavice") `vrací index prvku lavice`
s.sort() `přeuspořádá seznam alfanumericky`
s.reverse() `zrevertuje seznam`

Úkol: Vytvořte skript ***seznam_jmen.py****, který má následující nabídku činností:*
1. *přidat jméno do seznamu*
2. *vypsat seznam*
3. *vymazat jméno ze seznamu*
4. *vypsat počet jmen v seznamu*
5. *vypsat seznam jmen seřazený podle abecedy*

Nápověda: k vytištění seřazeného seznamu můžeme použít **funkci sorted()**, pak seznam zůstane uložen neseřazený, seřadí se pouze k tisku.

```
print (sorted(mujseznam))
```

nebo použijeme **metodu .sort**, ta seznam seřadí a uloží. Seřazený seznam vytiskneme:

```
mujseznam.sort
print (mujseznam)
```

5.3 TUPLE

Uspořádaná n-tice, jejíž prvky nelze měnit.

Syntaxe:

```
t = 'cau', 123, 564
```

může být i v kulatých závotkách.

```
t = ('cau', 123, 564)
```

Použití: vzhledem k tomu, že jde o neměnný datový typ, práce s ním je rychlejší než se seznamy a používá se jako klíč ke slovníkům.

Operace: podobné jako u řetězců.

5.4 SLOVNÍKY (DICTIONARIES)

Je to datový typ nazývaný též **asociativní pole**. Jde o **neuspořádanou** množinu **dvojic *klíč:hodnota*.** Klíčem musí být neměnný datový typ, integer, string nebo tuple.

Syntaxe:

```
slovnik1 = {'jedna':'one', 'dva':'two', 'tri':'three'}
```

nebo prázdný slovník:

```
slovnik2 = {}
```

nebo praktický slovník:

```
slovník3 = {'jmeno':'', 'prijmeni':'','mobil':'', 'emil':''}
```

K hodnotám slovníku přistupujeme podobně jako u seznamu, s tím rozdílem, že místo indexu používáme klíč.

```
>>> slovnik1['dva']
'two'
```

můžeme přidávat do slovníku:

```
>>> slovnik1['ctyry']='four'
>>> slovnik1
{'tri': 'three', 'ctyry': 'four', 'dva': 'two', 'jedna': 'one'}
```

můžeme mazat ze slovníku:

```
>>> del slovnik1['jedna']
>>> slovnik1
{'tri': 'three', 'dva': 'two'}
```

Úkol 1: Napište skript, který vytvoří adresář kontaktů. Kontakt bude typu dictionary s klíči: Jméno, příjmení, číslo mobilního telefonu. Adresář bude typu seznam. V menu bude nabídka dvou činností: Přidání nového kontaktu a vytisknutí celého adresáře. Uložte jako ***adresar1.py****.*

5.5 PRÁCE SE SOUBORY

Pokud program běží, data jsou uložena v paměti. Až program skončí data z paměti zmizí. Chtělo by to uložit data do souboru na disk. Python se soubory pracuje jako s knihou. Na začátku se musí otevřít, abychom do ní mohly psát nebo z ní číst, na konci ji zavřeme.

Otevření souboru:

```
f=open('soubor.txt','w')
```

Otevřením souboru vznikne v paměti nový souborový objekt datového typu stream – proud dat , proměnná `f` na něj odkazuje. Funkce `open` má několik parametrů, vysvětlím na příkladu:

```
f = open('skripty/soubor.txt','r', encoding='utf-8')
```

- jméno souboru (může být i s relativní adresou)
  např: `'skripty/soubor.txt'`
- kódování (pokud nezadáte, Python si zjistí výchozí kódování a to použije)
  např: `encoding='utf-8'`
- mód (pokud nezadáte, výchozí je `'r'`)

Módy:

`w`	znamená otevření souboru pro zápis **!!**pokud v souboru už nějaká data jsou, budou bez varování přepsána**!!**
`r`	znamená otevření souboru pro čtení.
a	znamená otevření souboru pro přidávání, což je stejné jako pro zápis, ale nová data se přidají na konec.
b	otevření souboru v binárním módu
t	otevření souboru v textovém módu
wb	znamená otevření souboru pro zápis v binárním kódu
...	

V případě, že otevíráme soubor pro psaní nebo přidávání, který ještě neexistuje, funkce *open* ho vytvoří v aktuálním adresáři (chceme-li ukládat jinam, musíme uvést cestu).

Tak tedy máme soubor `soubor.txt` otevřený pro zápis. K souborovému objektu `f` typu stream se váže několik metod.

Zápis a čtení ze souboru:

f.read()
metoda vrací celý obsah textového souboru jako řetězec . Při opakovaném použití vrací prázdný řetězec, chceme-li znovu načítat obsah souboru, musíme soubor „přetočit " na začátek pomocí metody seek:

f.seek(0)
metoda seek() zajistí přesun v textovém souboru na určenou bajtovou pozici – 0 znamená na začátek.

f.write('co chci zapsat')
metoda zapíše do textového souboru f řetězec 'co chci zapsat'

f.readline()
metoda načte jeden řádek z textového souboru (bez zadání parametrů čte včetně znaků konce řádku, což je obvykle '\n')

f.readlines()
metoda načte všechny řádky textového souboru a vrací je jako seznam řádků

Zavření souboru:

f.close()
metoda zavře otevřený soubor, na což bychom neměli zapomínat. Pro zapomětlivé je pohodlnější používat automatické uzavírání souborů pomocí konstrukce **with**

Otevření souboru s automatickým zavíráním:

```
with open('skripty/soubor.txt', 'r') as f:
    prvni_radek = f.readlines[0]
    print(prvni_radek)
```

vyzkoušejte:

```
>>> f=open('pokus.txt','w')
>>> f
<open file 'pokus.txt', mode 'w' at 0x012467E0>
```

podívejte se do aktuálního adresáře, jestli se tam objevil soubor pokus.txt

```
>>> f.write('vo co de? \n')
>>> f.close()
```

podívejte se do aktuálního adresáře, co je zapsáno v souboru pokus.txt

```
>>> f=file('pokus.txt','r')
>>> x=f.read()
>>> x
>>>'vo co de? \n'
>>> x=f.read()
>>> x
>>> ''
```

Znak pro konec řádku: '\n'

```
>>> f=file('pokus.txt','a')
>>> f.write('jo \n')
>>> f=file('pokus.txt','r')
>>> x=f.readline()
>>> x
>>> 'vo co de? \n'
>>> f.close()
```

Tímto způsobem lze pracovat s textovými soubory. Zapisujeme i čteme řetězce. Někdy ale potřebujeme uložit složitější datovou strukturu, třeba jako v úkolu1 seznam tvořený slovníky. Pro tyto případy slouží modul `Picle`.

Data ukládáme do souborů s příponou .pickle

Do souboru f chci uložit objekt x:

```
import pickle
with open('nazev souboru na disku', wb) as f
        pickle.dump(x,f)
```

Pro rekonstrukci objektu ze souboru f stačí použít:

```
with open('nazev souboru na disku', rb) as f
z=pickle.load(f)
```

Vyzkoušejte:

```
>>> f=open('pokus.pickle','wb')
>>> x=['jedna','dva','tri']
>>> import pickle
>>> pickle.dump(x,f)
>>> f.close()
>>> f=open('pokus.pickle','rb')
>>> z=pickle.load(f)
>>> z
>>> f.close()
```

*Úkol: Vytvořte skript s názvem **textovy-adresar.py** , který načítá jména a telefonní čísla osob. Zvolte vhodnou datovou strukturu. Po načtení se vytvořený seznam uloží do textového souboru **textovy-adresar.txt**, každá osoba na jeden řádek.*