Algoritmy a programování

Organizace předmětu

Úvod do Pythonu: proměnné, cykly, podmínky

Vojtěch Vonásek

Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

Vyučující

Vojtěch Vonásek

Přednášky, cvičení

Petr Štěpán

Vedoucí cvičení

Martin Řimnáč

Cvičení

Matej Novosad

Cvičení

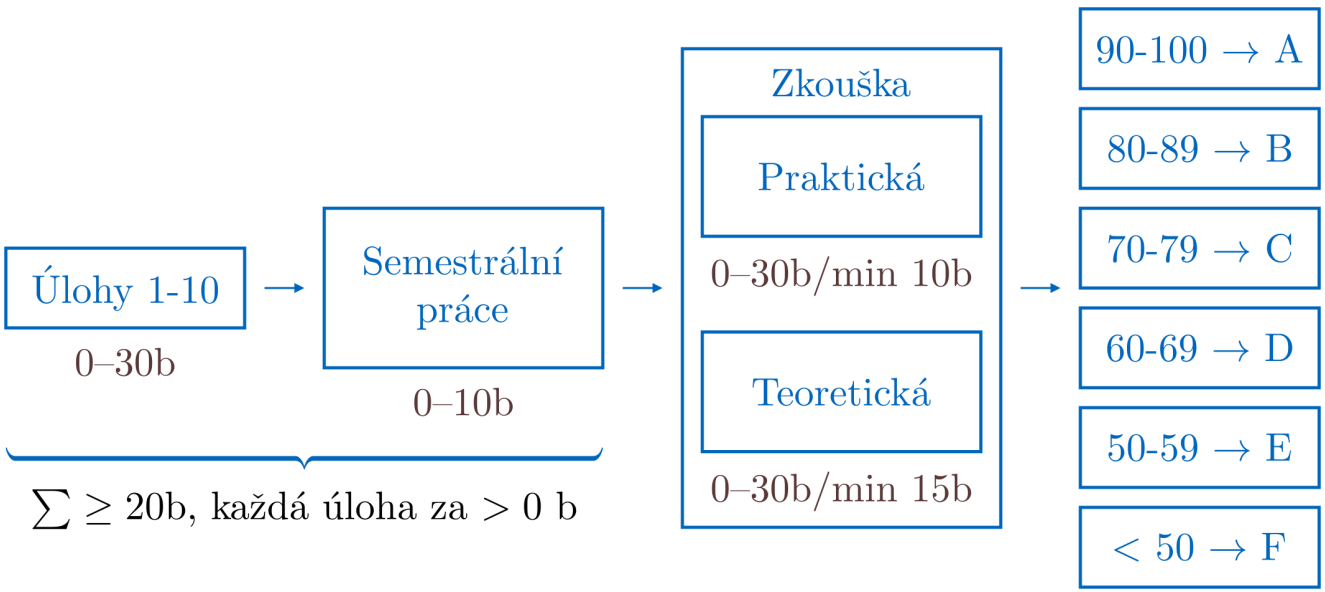
Michal Reiser

Cvičení

- Stránky předmětu: `cw.fel.cvut.cz/b231/courses/b3b33alp/`
- Komunikace s vyučujícími: email

Organizace předmětu

- Detaily jsou na webu předmětu
- Přednášky a cvičení jsou nepovinné
- Povinné odevzdání domácích úloh a semestrální práce
- Pokud splníte minimální počet bodů → zápočet → lze jít na zkoušku
- Zkouška: praktická a teoretická část, každá část musí být splněna na minimální počet bodů



Naplň ALP

- Základní kurz programování v Pythonu ($\sim$ první třetina semestru)
- Základní algoritmy, techniky a datové struktury

Náplň ALP

- Základní kurz programování v Pythonu ($\sim$ první třetina semestru)
- Základní algoritmy, techniky a datové struktury

Proč Python?

Výhody

- Skriptovací, dynamicky typovaný jazyk
- Není třeba (většinou) se starat o paměť
- Uživatelsky přívětivý $\rightarrow$ vhodný pro začátečníky
- Obrovská uživatelská základna
  - velké množství (open-source) knihoven
  - mnoho dokumentace a tutoriálů
- Hojně používaný v praxi (skriptování, web aplikace, zpracování dat, prototypování, AI

Nevýhody

- Rychlý vývoj $\rightarrow$ některé formy zápisu nefungují na nových verzích (a naopak)
- Pomalý (pokud nejsou výpočty přes kompilované knihovny)
- Umožňuje zápisy, jejichž pochopení vyžaduje detailní znalosti Pythonu (nevhodné pro začátečníky)

Proč Python?

- Python je velmi populární jazyk
- IEEE 2024 Spectrum: 1. místo (2024)
- Tiobe index (`https://www.tiobe.com/tiobe-index/`): 1. místo (2024)
- PYPL index (`https://pypl.github.io/PYPL.html`): 1. místo (2024)

Worldwide, Sept 2024 :

Rank	Change	Language	Share	1-year trend
1		Python	29.66 %	+1.6 %
2		Java	15.64 %	-0.2 %
3		JavaScript	8.3 %	-1.0 %
4		C#	6.64 %	-0.1 %
5		C/C++	6.46 %	-0.2 %
6	↑	R	4.66 %	+0.2 %
7	↓	PHP	4.35 %	-0.5 %
8		TypeScript	2.96 %	-0.0 %
9		Swift	2.69 %	+0.0 %
10	↑	Rust	2.65 %	+0.6 %
11	↓	Objective-C	2.45 %	+0.2 %
12		Go	2.08 %	+0.2 %

Proč Python?

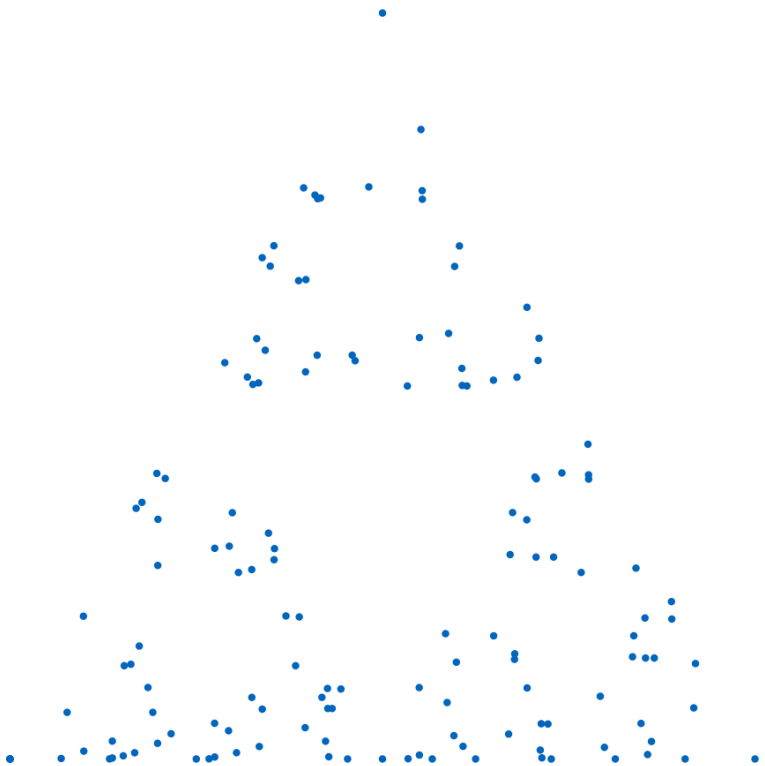
- Python je velmi populární jazyk
- IEEE 2024 Spectrum: 1. místo (2024)
- Tiobe index (`https://www.tiobe.com/tiobe-index/`): 1. místo (2024)
- PYPL index (`https://pypl.github.io/PYPL.html`): 1. místo (2024)

Sep 2024	Sep 2023	Change	Programming Language	Ratings	Change
1	1		Python	20.17%	+6.01%
2	3	↑	C++	10.75%	+0.09%
3	4	↑	Java	9.45%	-0.04%
4	2	↓	C	8.89%	-2.38%
5	5		C#	6.08%	-1.22%
6	6		JavaScript	3.92%	+0.62%
7	7		Visual Basic	2.70%	+0.48%
8	12	⇈	Go	2.35%	+1.16%
9	10	↑	SQL	1.94%	+0.50%
10	11	↑	Fortran	1.78%	+0.49%

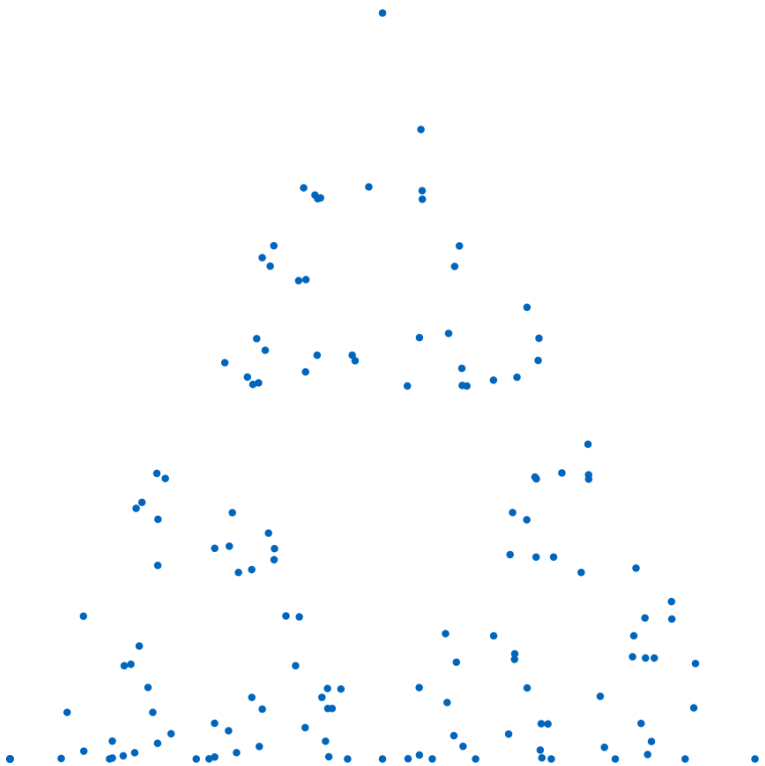
Proč Python?

- Python je velmi populární jazyk
- IEEE 2024 Spectrum: 1. místo (2024)
- Tiobe index (`https://www.tiobe.com/tiobe-index/`): 1. místo (2024)
- PYPL index (`https://pypl.github.io/PYPL.html`): 1. místo (2024)

```
import random as R
p = [[0,0],[2,0],[1,1]]
last = [0,0]
for i in range(10000):
    rp = p[R.randint(0,2)]
    x = (last[0]+rp[0])/2
    y = (last[1]+rp[1])/2
    last = [x,y]
    print(x,y)
```



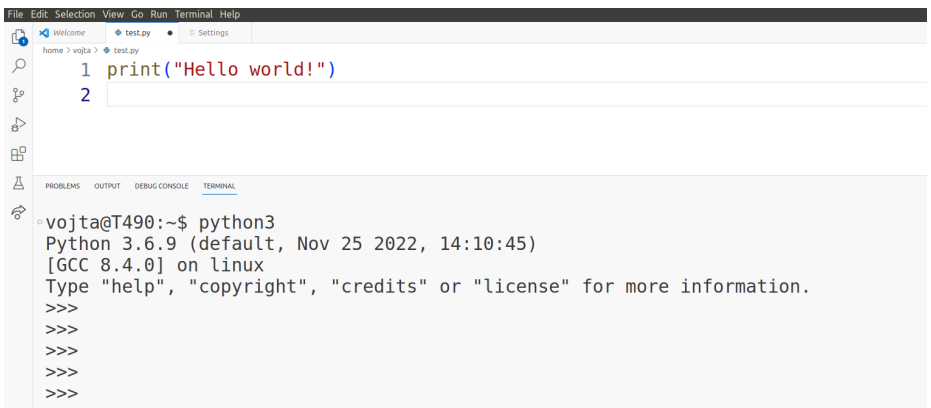
```
import random as R
p = [[0,0],[2,0],[1,1]]
last = [0,0]
for i in range(10000):
    rp = p[R.randint(0,2)]
    x = (last[0]+rp[0])/2
    y = (last[1]+rp[1])/2
    last = [x,y]
    print(x,y)
```



Interaktivní režim (interpret)

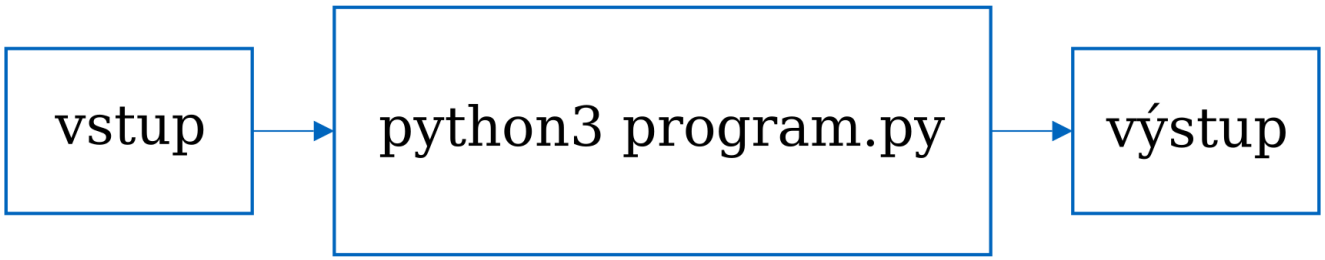
- Příkazy jsou přímo zadávány (z klávesnice) do interpreteru, a ihned vyhodnoceny
- Testování kratkých programů
- REPL — Read-Evaluate-Print Loop
- Spuštění:
  - Linux/Win: > `python3`
  - GUI (VSCODE apod.): Sekce TERMINAL, příkaz `python3`
- Ukončení CTRL+D nebo příkaz `quit()`

```
vojta@T490:~$ python3
Python 3.6.9 (default, Nov 25 2022, 14:10:45)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "Hello World!"
>>> a
'Hello World!'
>>> print(a)
Hello World!
>>>
```



Spuštění programu ze souboru

- Programy jsou typicky uloženy v txt souborech (přípona .py)
- Spuštění:
  - Linux/Win: `> python3 program_v_pythonu.py`
  - VSCODE: CTRL+F5



Vstup

- **Standardní vstup:** (typicky) klávesnice: funkce `input()`
- Data ze souboru

Výstup

- **Standardní výstup:** tisk na obrazovku, funkce `print()`
- Uložení dat do souboru

Datové typy

Skalární datové typy — obsahují jeden prvek

- `int` — celá čísla `5`
- `float` — reálná čísla `5.0`
- `bool` — hodnoty `True` nebo `False`
- `NoneType` — hodnota `None`

Složené datové typy — mají vnitřní strukturu, k vnitřním prvků lze individuálně přistupovat

- `str` — řetězce `"ahoj"`
- `list` — pole `[1,2,3]`
- `tuple` — n-tice `(1,2,3)`
- a další + uživatelem definované objekty

Funkce type() — určí datový typ

- `type(5)` → `<class 'int'>`
- `type(5.0)` → `<class 'float'>`
- `type(False)` → `<class 'bool'>`

Proměnné

- Proměnná označuje místo v paměti, kde jsou uložena data
- Proměnná má jméno, hodnotu a datový typ

```
1 a = 5
```

- ‘=’ přiřadí hodnotu 5 do proměnné ‘a’
- pokud proměnná (nalevo od =) neexistuje, je vytvořena

Jaké jsou hodnoty proměnných po skončení programu?

```
1 a = 5
2 b = a
3 a = 3
```

Jména proměnných

- Musí začínat písmenem nebo '_' (podtržítko)
- Může obsahovat písmena, číslice nebo '_'
- Jména proměnných jsou case-sensitive
- Nesmí být stejné jako tzv. klíčová slova:

```
1 False None True and as assert async await break class
    continue def del elif else except finally for from
    global if import in is lambda nonlocal not or pass
    raise return try while with yield
```

Doporučení

- Jméno proměnné vystihuje její význam

  `numberOfWords`, `sumOfProducts`, `actualPrice`, `positionX`
- Jednopísmenné proměnné `i`,`j`,`k`,... používáme pro krátké cykly
- Nepoužíváme `tooLongNamesOfVariablesThatAreDifficultToType`

Operace pro float a int

- Sčítání, odečítání, násobení: `a+b` `a-b` `a*b`
  - pokud jsou oba operandy int, výsledek je int
  - pokud je alespoň jeden operand float, výsledek je float

    `1+1` $\rightarrow$ `1` `1+1.0` $\rightarrow$ `1.0`
- Dělení: `a/b`
  - výsledek je vždy float (v Python3)

    `1/2` $\rightarrow$ `0.5` `1/1` $\rightarrow$ `1.0` `4./2` $\rightarrow$ `2.0`
- Celočíselné dělení: `a//b`
  - pokud jsou oba operandy int, výsledkem je int
  - pokud jeden operand float, výsledek je float
  - je to výsledek dělení, na které se aplikuje funkce floor: `a//b` = $\lfloor \frac{a}{b} \rfloor$

    `1//3` $\rightarrow$ `0` `1//1` $\rightarrow$ `1` `7//2` $\rightarrow$ `3`

Pořadí vyhodnocování operátorů

Co je výsledek? `3**2**3//1000`

- `(3**2)**(3//1000)` → 1
- `3**(2**(3//1000))` → 3
- `(3**(2**3))//1000` → 6
- `((3**2)**3)//1000` → 0
- `3**2**3//1000` → ?

PEMDAS

- Python používá systém PEMDAS (Parenthesis, Exponentiation, Multiplication, Division, Addition, Substraction)
- Vyhodnocování operátorů dle jejich priority
- Priority operátorů mohou být v různých jazycích jiné
- Řešením je nařídit prioritu použitím závorek
- **Vždy používejte závorky pro určení priority operací**

Priority operátorů

`(expressions...)` `[expressions...]` `{key: value...}`, `{expressions...}`	Binding or parenthesized expression list display, dictionary display, set display
`x[index]`, `x[index:index]` `x(arguments...)`, `x.attribute`	Subscription, slicing, call, attribute reference
`await x`	Await expression
`**`	Exponentiation
`+x`, `-x`, `~x`	Positive, negative, bitwise NOT
`*`, `@`, `/`, `//`, `%`	Multiplication, matrix multiplication, division, floor division, remainder
`+`, `-`	Addition and subtraction
`<<`, `>>`	Shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `!=`, `==`	Comparisons, including membership tests and identity tests
`not x`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR
`if-else`	Conditional expression
`lambda`	Lambda expression
`:=`	Assignment expression

Přetypování

- `int(a)`: výsledek je datový typ integer
- `float(a)`: výsledek je float
- `str(a)`: výsledek je string (řetězec)
- `float()` a `int()` mohou selhat, pokud vstup nejde převést na float/int

```
1 a = 5  # a je integer
2 print(a, type(a))
3
4 a = str(a)  # a je retezec
5 print(a, type(a))
6
7 a = float(a)  # a je float
8 print(a, type(a))
```

```
5 <class 'int'>
5 <class 'str'>
5.0 <class 'float'>
```

Kdy lze přetypovat

- `int(a)`
  - pokud je a float, lze vždy
  - pokud je a string, musí obsahovat **pouze celé číslo**
- `float(a)`
  - pokud je a int, lze vždy
  - pokud je a string, musí obsahovat **pouze desetinné číslo**

```
1 int(5.0)
```

```
5
```

```
1 float(5)
```

```
5.0
```

Kdy lze přetypovat

- `int(a)`
  - pokud je a float, lze vždy
  - pokud je a string, musí obsahovat **pouze celé číslo**
- `float(a)`
  - pokud je a int, lze vždy
  - pokud je a string, musí obsahovat **pouze desetinné číslo**

```
1 int("5.0")
```

```
ValueError: invalid literal for int() with base 10: '
    5.0'
```

```
1 float("a=5")
```

```
ValueError: could not convert string to float: 'a=5'
```

Vědecký zápis čísel

- Obecně: `MeN`, kde M je float, N je int

$$MeN = M \cdot 10^N$$

- `2e3` $\rightarrow$ 2000.0, `1.1e-2` $\rightarrow$ 0.011
- Vhodné zejména pro zápis velkých nebo malých čísel

```
1 float("15e-5")
```

```
0.00015
```

```
1 int("15e-5")
```

```
ValueError: invalid literal for int() with base 10: '
    14e-5'
```

Zápis desetinných čísel

Jaká hodnota bude uložena v proměnné `b`?

```
1 a = 3,14159
2 b = a/2
3 print(b)
```

Zápis desetinných čísel

Jaká hodnota bude uložena v proměnné `b`?

```
1 a = 3,14159
2 b = a/2
3 print(b)
```

```
    b = a/2
TypeError: unsupported operand type(s) for /: 'tuple'
   and 'int'
```

- Reálná čísla (float) zapisujeme se znakem '.' (tečka), nikoliv ',' (čárka)
- Znak ',' je určen pro datový typ `tuple`

Standardní vstup

- Načtení vstupu z klávesnice: funkce `input()`
- Tato funkce vrací řetězec

```
1 a = input()  #a je string
2 print("Uzivatel zadal", a)
```

- Pokud je třeba s výsledkem pracovat jako s číslem, je třeba přetypovat
- `float()` nebo `int()`

```
1 a = int( input() ) #vstup musi byt cele cislo
2 a += 1
3 print(a)
```

Standardní vstup

- Pro testování se hodí mít vstup připraven v souboru, např. `test.txt`
- Tento soubor lze přesměrovat na tzv. standarní vstup na příkazové řádce
- Každé volání `input()` načte jednu řádku ze souboru

`test.txt`

```
10
Python is my friend
20.3
```

`program.py`

```
1 a = input()
2 b = input()
3 c = input()
4 print("a:",a)
5 print("b:",b)
6 print("c:",c)
```

```
1 python3 program.py < test.txt
```

```
a: 10
b: Python is my friend
c: 20.3
```

Podmíněné příkazy

- Pokud chceme vykonat část programu jen při splnění určitých podmínek

```
1 if condition:
2     code1
3 code2
```

- Příkazy `code1` se vykonají jen pokud je podmínka splněna
  - Příkazy `code1` jsou odsazené o alespoň jednu mezeru oproti úrovni, kde je `if`
  - Konec bloku `code1` je rozpoznán podle změny odsazení
- Příkazy `code2` se vykonají bez ohledu na podmínku

Podmíněné příkazy: if-then


```
1 a = 4
2 if a > 0:
3     print("positive")
4 print("after if")
```

```
positive
after if
```

```
1 a = 10
2 if a**2 < 100:
3     print("true")
4 print("after if")
```

```
after if
```

Podmíněné příkazy

- Část programu je vykonaná při splnění podmínky (`if`)
- Druhá část při nesplnění této podmínky (`else`)

```
1 a = 11
2 if a % 2 == 0:
3     print("even")
4 else:
5     print("odd")
```

```
odd
```

Podmíněné příkazy: if-elif

- Vícenásobné větvení: if-elif

```
1 a = (-1)**3
2 if a > 0:
3     print("positive")
4 elif a < 0:
5     print("negative")
6 else:
7     print("zero")
```

```
negative
```

Podmíněné příkazy: if-elif

- Pozor na správné pořadí podmínek

```
1  n = 5
2  print("Rad",n,"jsou")
3  if n < 10:
4      print("jednotky")
5  elif n < 100:
6      print("desitky")
7  elif n < 1000:
8      print("stovky")
9  else:
10     print("tisice+")
```

```
Rad 5 jsou
jednotky
```

```
1  n = 5
2  print("Rad",n,"jsou")
3  if n < 1000:
4      print("stovky")
5  elif n < 100:
6      print("desitky")
7  elif n < 10:
8      print("jednotky")
9  else:
10     print("tisice+")
```

```
Rad 5 jsou
stovky
```

Podmíněné příkazy

- Podmínky lze spojovat logickými operátory `and`, `or`, `not`

```
1 a = int(input())
2 b = int(input())
3 if b != 0 and (a % b) == 0:
4     print("a␣is␣dividable␣by␣b")
```

Porovnání hodnot

- `a != b` a není rovno b
- `a < b` a je menší než b
- `a > b` a je větší než b
- `a <= b` a je menší nebo rovno b
- `a >= b` a je větší nebo rovno b

Podmíněné příkazy

- Podmínka složená z více termů (spojených `and`, `or` a `not`) se vyhodnocuje podle priority operátorů
- Vyhodnocení se ukončuje jakmile je výsledek podmínky jednoznačně znám
  - to znamená, že ne všechny termy jsou vyhodnoceny
  - může vést na chyby, které se objeví “jen někdy”

```
1 a = 10
2 b = "10"
3 if a == 0 and (b % 2) == 0:
4     print("true")
5 else:
6     print("false")
```

```
false
```

Podmíněné příkazy

- Program obsahuje chybu, která se projeví jen pokud $a = 0$

```
1 a = 0
2 b = "10"
3 if a == 0 and (b % 2) == 0:
4     print("true")
5 else:
6     print("false")
```

```
    if a == 0 and (b % 2) == 0:
TypeError: not all arguments converted during string
    formatting
```

- Pořadí termů v podmínkách je třeba promyslet

Cykly

- Cykly umožňují opakované vykonání části programu
  - Předepsaný počet opakování (for cyklus)
  - Počet opakování je ukončen podmínkou (while cyklus)

For cyklus

- Předepsaný (přesný) počet opakování
- Odsazené příkazy `code` jsou vykonány opakovaně

```
1 for variable in iterator/iterable_object:
2     code
```

Iterátor

- Poskytuje sekvenci hodnot, např. $0, 1, 2, \ldots$
- Funkce `range()`

Iterable object

- Je proměnná, která umožňuje iterovat (procházet) přes její vnitřní hodnoty
- string, pole, n-tice, dictionary

For cyklus: funkce range

`range()`

- Funkce, která vrací celá čísla z definované posloupnosti
- Posloupnost lze definovat třemi způsoby:
  - `range(n)` — vrací čísla 0,1, ... n
  - `range(a,b)` — vrací čísla $a, a+1, a+2, \ldots, b-1$
  - `range(a,b,s)` — vrací čísla $a, a+s, a+2s, \ldots$, maximálně do $b-1$
- Ve všech případech se jedná o zhora otevřený internal, tj. horní mez b není výstupem
- Důvodem je indexování stringů a polí

For cyklus: range(n)

`range(n)`

- Vrací n celých čísel $0, 1, \ldots, n-1$
- Hodnota n není součástí sekvence!
- Podmínka pro vykonání: $n > 0$
- n musí být integer (jinak nastane chyba běhu)

```
1 for i in range(5):
2     print("i",i)
3 print("end")
```

```
i 0
i 1
i 2
i 3
i 4
end
```

```
1 for XYZ in range(-5):
2     print(XYZ)
3 print("end")
```

```
end
```

- Pokud $n \leq 0$, cyklus se neprovede (program běží dále) — nejedná se o chybu běhu

For cyklus: range(n)

`range(n)`

- *n* musí být integer

```
1 a = int(input())  # assume that user types 10
2 n = a/2
3 for i in range(n): #we want to iterate 0,1,...a/2
4     print("i",i)
5 print("end")
```

```
    for i in range(n): #we want to iterate 0,1,...a/2
TypeError: 'float' object cannot be interpreted as an
    integer
```

- Pokud *n* není integer, vznikne chyba běhu (program se ukončí)
- Tato chyba se nedá (obecně) detekovat před spuštěním programu
- Proto je při volání `range(n)` mít jistotu, že *n* je integer!

For cyklus: range(a,b)

`range(a,b)`

- Vrací celá čísla $a, a+1, a+2, \ldots, b-1$
- Hodnota b není součástí sekvence!
- Musí platit, že $a < b$
- a, b musí být typu integer

```
1 for i in range(1,5):
2     print(i)
```

```
1
2
3
4
```

```
1 for i in range(4,5):
2     print(i)
```

```
4
```

For cyklus: range(a,b)

`range(a,b)`

- Vrací celá čísla $a, a+1, a+2, \ldots, b-1$
- Hodnota b není součástí sekvence!
- Musí platit, že $a < b$

```
1 for i in range(5,5):
2     print(i)
3 print("end")
```

```
end
```

```
1 for i in range(-4,0):
2     print(i)
```

```
-4
-3
-2
-1
```

For cyklus: range(a,b,s)

`range(a,b,s)`

- s je krok (step) sekvence , $s \neq 0$
- Vrací celá čísla $a, a+s, a+2s, a+3s \ldots$, maximálně do čísla $b-1$
- Hodnota b není součástí sekvence!
- Pokud $s > 0$, musí $a < b$
- Pokud $s < 0$, musí, $a > b$
- a, b, s musí být typu integer

```
1 for i in range(1,6,2):
2     print(i)
```

```
1
3
5
```

```
1 for i in range(0,-10,-3):
2     print(i)
```

```
0
-3
-6
-9
```

For cyklus: příklady

Co bude výstupem následujících programů?

```
1 for i in range(1,6,2):
2     a = 2**i
3     print(a)
```

```
1 for i in range(10):
2     print(i)
3     i = 1
```

```
1 a = 0
2 for b in range(5):
3     print(a,b)
4     a = b
```

For cyklus: příklady

Co se stane při spuštění následujících programů?

```
1 for i in range(1,6,-2):
2     print(i)
```

```
1 for i in range(-10):
2     print(a)
```

```
1 a = 0
2 for a in range(5):
3     print(2*b)
4     b = 2
```

```
1 for q in range(-3,2,-1):
2      print(q)
```

While cyklus

- Opakované vykonání příkazů dokud je splněna podmínka
- Podmínka/y se zapisují stejně jako u konstrukce `if`
- Počet opakování není dopředu znám (záleží na podmínce)
- Při nevhodně formulované podmínce může vzniknout nekonečný cyklus!

```
while condition:
    code
```

```
i = 0
while i < 5:
    print(i)
    i += 1
print("after while")
```

```
0
1
2
3
4
after while
```

While cyklus: nekonečný cyklus


```
1 i = 0
2 while i < 5:
3     print(i)
4 #     i += 1
```

```
1 a = 1
2 b = 0
3 while b < 10:
4     print(a,b)
5     a += 1
```

```
1 while True:
2     print("true")
```

Příkazy break a continue

- Slouží k řízení cyklu (for i while)
- `break` — ukončení cyklu
- `continue` — další iterace cyklu
- Jsou typicky použity v podmínce
- U vnořených cyklů se `break` a `continue` týká pouze nejbližšího nadřazeného cyklu

```
1 i = 0
2 while i < 5:
3     print(i)
4     i += 1
5 print("end")
```

```
0
1
2
3
4
end
```

```
1 i = 0
2 while i < 5:
3     print(i)
4     i += 1
5     if i == 3:
6         break
7 print("end")
```

```
0
1
2
end
```

break: příklady


```
1 for i in range
      (3,10,2):
2     if i > 5:
3         break
4     print(i)
```

```
3
5
```

```
1 i = 0
2 while True:
3     print(i)
4     if i > 5:
5         break
6     i += 1
```

```
0
1
2
3
4
5
6
```

continue: příklady

- Klíčové slovo `continue` zajistí další iteraci cyklu

Provést opakování pro všechny hodnoty $0, 1 \ldots, 9$ kromě 5

```
1 for i in range(10):
2     if i == 5
3         continue
4     print(i)
```

Jiný způsob:

```
1 for i in range(10):
2     if i != 5:
3         print(i)
```

Cyklus: příklady

Co se stane při spuštění následujících programů?

```
1 print("start")
2 for i in range(-1,2,1):
3     break
4     print(i)
5 print("end")
```

```
1 for i in range(10):
2     print(a)
3     continue
```

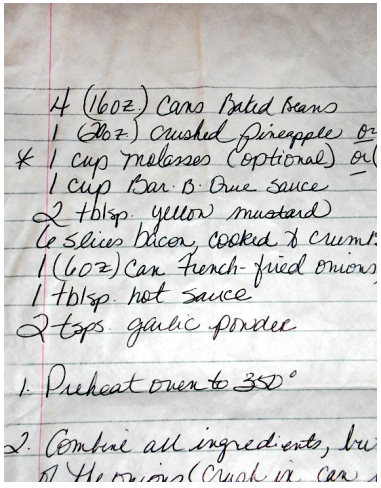
```
1 for i in range(10):
2     continue
3     print(a)
```

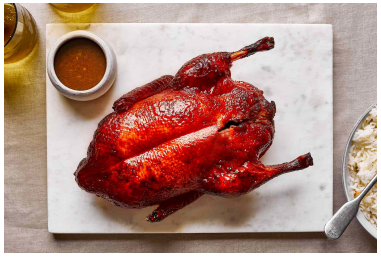
```
1 i = 0
2 while i < 10:
3     break
4     print(i)
5     i += 1
```

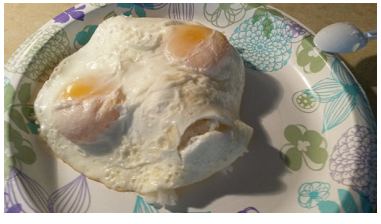
```
1 i = 0
2 while i < 10:
3     print(i)
4     continue
5     i += 1
```

Algoritmus

- Postup vyřešení nějaké úlohy
- Slovní popis, rovnice, vývojový diagram, pseudokód
- “Algoritmus” se objevuje i v jiných odvětvích
  - kuchařský recept, chemická reakce, výrobní postup







Implementace

- Implementace je zápis algoritmu ve vybraném programovacím jazyce
- + ladění, testování, dokumentace

Programování

- Obecný pojem zahrnující tvorbu algoritmů (algoritmizace), implementaci, testování, ladění, tvorba dokumentace . . .

Příklady algoritmů

Algorithm 1: A*

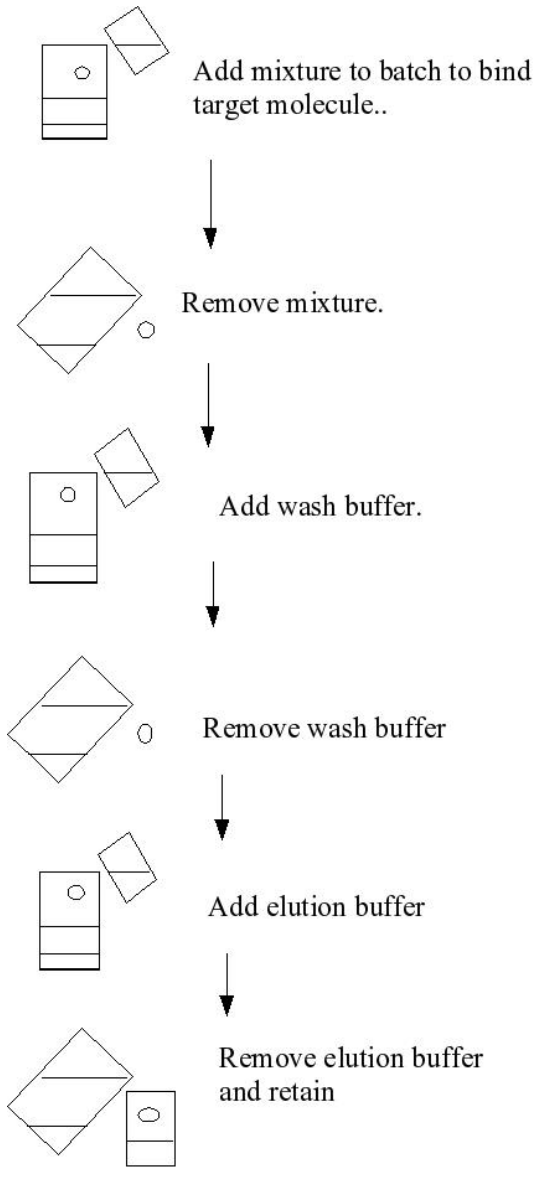
Input: start, $goal(n)$, $h(n)$, $expand(n)$
Output: path

```
1  if goal(start) = true then return makePath(start)
2
3  open ← start
4  closed ← ∅
5  while open ≠ ∅ do
6      sort(open)
7      n ← open.pop()
8      kids ← expand(n)
9      forall the kid ∈ kids do
10         kid.f ← (n.g + 1) + h(kid)
11         if goal(kid) = true then return makePath(kid)
12         if kid ∩ closed = ∅ then open ← kid
13     closed ← n
14 return ∅
```

PPPD-JBH Cold Tone Paper Developer

SOLUTION A		
Water at 125° F	**750 ml**	**1,500 ml**
Phenidone *_NOTE_	**0.3 g**	**0.6 g**
Sodium sulfite	**43 g**	**86 g**
Pyrocatechol	**10 g**	**20 g**
Citric Acid	**5 g**	**10 g**
Potassium bromide	**3.5 g**	**7 g**
Water to make	**1 liter**	**2 liter**

***NOTE:** Dissolve in 5-8ml of Isopropyl Alcohol, then add to water.*



Co je program?

- Sekvence instrukcí
  - výpočetní operace
  - práce s pamětí
  - větvení, cykly ...

```
a = 2**(1/3)+4
a[i],b[j] = b[j], a[i]
for i in range(10)
```

- Programovací jazyk je způsob zápisu algoritmu pro realizaci na HW
- High-level vs. low-level jazyky

```
def generate_points(self):
    p_min, p_max = (
        np.array([self.x_range[0], self.y_range[0]]),
        np.array([self.x_range[1], self.y_range[1]]),
    )
    curves = plot_isoline(
        fn=lambda u: self.function(u[0], u[1]),
        pmin=p_min,
        pmax=p_max,
        min_depth=self.min_depth,
        max_quads=self.max_quads,
    )  # returns a list of lists of 2D points
    curves = [
        np.pad(curve, [(0, 0), (0, 1)]) for curve in curves if curve != []
    ]  # add z coord as 0
    for curve in curves:
        self.start_new_path(curve[0])
        self.add_points_as_corners(curve[1:])
    if self.use_smoothing:
        self.make_smooth()
    return self
```

Low-level programovací jazyk

- Program je sestaven z instrukcí pro konkrétní procesor
- Vyžaduje znalost HW architektury (registry, adresování, časování, . . .)
- Složitý na naučení, ladění, opravy
- Strojový kód
  - Binární sekvence, každý byte (skupina bytů) obsahuje zakódovanou instrukci a její argumenty
  - Programuje se přímo v binárním kódu
- Assembler
  - Krátké příkazy pro jednotlivé operace procesoru
  - např. LDA (Load to Accumulator), MUL (násobení)
  - Program je txt soubor, který je přeložen (assemblerem) do strojového kódu

```
00000398 CD 90 09 B2   25 6D 0C 43
000003C0 F3 D7 1E 55   8B 17 EA FE
000003E8 64 FC BD BE   B3 6C 63 F6
00000410 36 36 6F 88   FC 6A 80 23
00000438 98 78 45 E1   9F 1C 70 B8
```

Opcode		Addressing mode	Assembler format	Length in bytes	Number of cycles
Dec	Hex				
169	A9	Immediate	LDA *#nn*	2	2
173	AD	Absolute	LDA *nnnn*	3	4
189	BD	Absolute,X	LDA *nnnn*,X	3	4*
185	B9	Absolute,Y	LDA *nnnn*,Y	3	4*
165	A5	Zeropage	LDA *nn*	2	3
181	B5	Zeropage,X	LDA *nn*,X	2	4
161	A1	Indexed-indirect	LDA (*nn*,X)	2	6
177	B1	Indirect-indexed	LDA (*nn*),Y	2	5*

Machine code bytes	Assembly language statements
	`foo:`
`B8 22 11 00 FF`	`movl $0xFF001122, %eax`
`01 CA`	`addl %ecx, %edx`
`31 F6`	`xorl %esi, %esi`
`53`	`pushl %ebx`
`8B 5C 24 04`	`movl 4(%esp), %ebx`
`8D 34 48`	`leal (%eax,%ecx,2), %esi`
`39 C3`	`cmpl %eax, %ebx`
`72 EB`	`jnae foo`
`C3`	`retl`

Instruction stream

```
B8 22 11 00 FF 01 CA 31 F6 53 8B 5C 24
04 8D 34 48 39 C3 72 EB C3
```

High-level programovací jazyk

- Program jsou příkazy podobné (ideálně) lidskému jazyku
- Je nezávislý na HW a operačním systému
- Strojový kód (“binárka”, “exe”) vzniká překladačem
- Vývoj, modifikace, ladění, optimalizace je jednodušší než u low-level jazyků
- C/C++, C#, Java, Python, Rust . . .

```
const vector<int> &neighbors(edges[actual]);
for(int i=0;i<(int)neighbors.size();i++) {
    const int node = neighbors[i];
    if (isKnown[node] ==  0 ) {
        isKnown[ node ] = 1;
        const double dd = nodeDistance(query, nodes[ node ]);
        if (dd < upperBound) {
            openList.push_back( std::pair<int, double>( node, dd ) );
        }
        if (dd < bestSolutionDist) {
            bestSolutionDist = dd;
            bestSolution = node;
            upperBound = bestSolutionDist;
        }
    }
}
```

```
1 import random as R
2 p = [[0,0],[2,0],[1,1]]
3 last = [0,0]
4 for i in range(10000):
5     rp = p[R.randint(0,2)]
6     x = (last[0]+rp[0])/2
7     y = (last[1]+rp[1])/2
8     last = [x,y]
9     print(x,y)
```

Proč Python?

Výhody

- Skriptovací, dynamicky typovaný jazyk
- Není třeba (většinou) se starat o paměť
- Uživatelsky přívětivý → vhodný pro začátečníky
- Obrovská uživatelská základna
  - velké množství (open-source) knihoven
  - mnoho dokumentace a tutoriálů
- Hojně používaný v praxi (skriptování, web aplikace, zpracování dat, prototypování, AI

Nevýhody

- Rychlý vývoj → některé formy zápisu nefungují na nových verzích
- Pomalý (pokud nejsou výpočty přes kompilované knihovny)
- Umožňuje zápisy, jejichž pochopení vyžaduje detailní znalosti Pythonu (nevhodné pro začátečníky)