

```
270     childpos = rightpos
271     # Move the smaller child up.
272     heap[pos] = heap[childpos]
273     pos = childpos
```

Algoritmy a programování

Organizace předmětu

Úvod do Pythonu: proměnné, cykly, podmínky

```
284     # newitem fits.
285     while pos > startpos:
286         parentpos = (pos - 1) // 2
287         parent = heap[parentpos]
288         if parent < newitem:
289             heap[pos] = parent
290             pos = parentpos
291             continue
292         break
293     heap[pos] = newitem
```

Vojtěch Vonásek

Department of Cybernetics

Faculty of Electrical Engineering

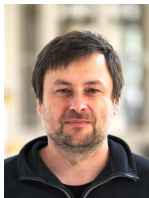
Czech Technical University in Prague

```
294
295     def _siftup_max(heap, pos):
296         'Maxheap variant of _siftup'
297         endpos = len(heap)
298         startpos = pos
299         newitem = heap[pos]
300         # Bubble up the larger child until hitting a leaf.
301         childpos = 2*pos + 1     # leftmost child position
302         while childpos < endpos:
```



Vojtěch Vonásek

Přednášky, cvičení



Petr Štěpán

Vedoucí cvičení



Martin Řimnáč

Cvičení



Pavel Petráček

Cvičení

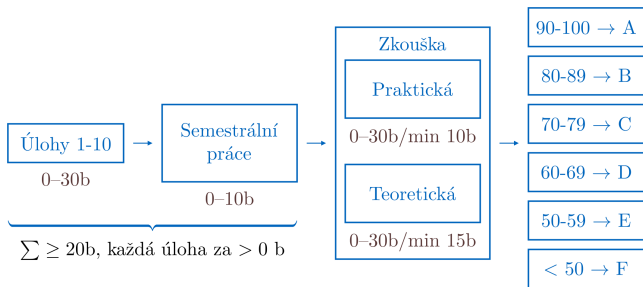


Jan Kybic

Garant

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

- Přednášky a cvičení jsou nepovinné
- Povinné odevzdání domácích úloh a semestrální práce
- Pokud splníte minimální počet bodů → zápočet
- Ke zkoušce pouze se zápočtem
- Zkouška: praktická a teoretická část, ~ 1.5 h/část, každá část musí být splněna na minimální počet bodů



Náplň ALP

- Základní kurz programování v Pythonu (~ první třetina semestru)
- Základní algoritmy

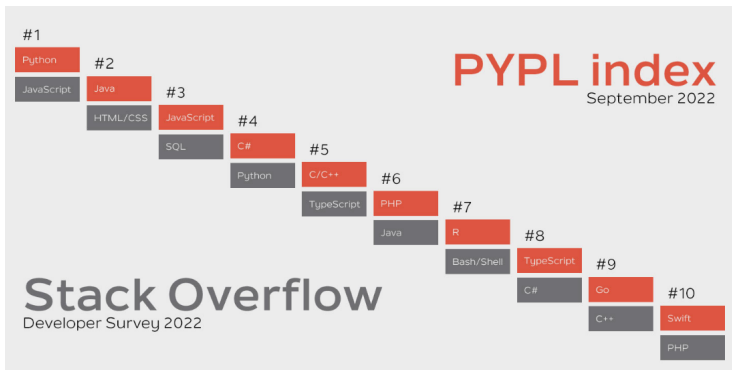
Výhody

- 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 (a naopak)
- Pomalý (pokud nejsou výpočty přes kompilované knihovny)
- Umožňuje zápisy, jejichž pochopení vyžaduje detailní znalosti Pythonu (nevhodné pro začátečníky)

- Python je velmi populární jazyk
- Tiobe index (<https://www.tiobe.com/tiobe-index/>): 1. místo (2023)
- PYPL index (<https://pypl.github.io/PYPL.html>): 1. místo (2023)



- Python je velmi populární jazyk
- Tiobe index (<https://www.tiobe.com/tiobe-index/>): 1. místo (2023)
- PYPL index (<https://pypl.github.io/PYPL.html>): 1. místo (2023)

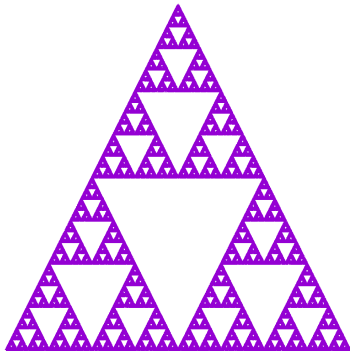
Jul 2023	Jul 2022	Change	Programming Language	Ratings	Change
1	1		 Python	13.42%	-0.01%
2	2		 C	11.56%	-1.57%
3	4	▲	 C++	10.80%	+0.79%
4	3	▼	 Java	10.50%	-1.09%
5	5		 C#	6.87%	+1.21%
6	7	▲	 JavaScript	3.11%	+1.34%
7	6	▼	 Visual Basic	2.90%	-2.07%
8	9	▲	 SQL	1.48%	-0.16%
9	11	▲	 PHP	1.41%	+0.21%
10	20	▲	 MATLAB	1.26%	+0.53%
11	18	▲	 Fortran	1.25%	+0.49%

- Python je velmi populární jazyk
- Tiobe index (<https://www.tiobe.com/tiobe-index/>): 1. místo (2023)
- PYPL index (<https://pypl.github.io/PYPL.html>): 1. místo (2023)

Worldwide, Jul 2023 :

Rank	Change	Language	Share	1-year trend
1		Python	27.43 %	-0.2 %
2		Java	16.19 %	-1.0 %
3		JavaScript	9.4 %	-0.1 %
4		C#	6.77 %	-0.3 %
5		C/C++	6.44 %	+0.2 %
6		PHP	5.03 %	-0.4 %
7		R	4.45 %	+0.1 %
8		TypeScript	3.02 %	+0.3 %
9	↑	Swift	2.42 %	+0.4 %
10	↑↑↑	Rust	2.15 %	+0.6 %
11	↓↓	Objective-C	2.13 %	+0.0 %
12	↓	Go	2.01 %	+0.0 %

```
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)
```



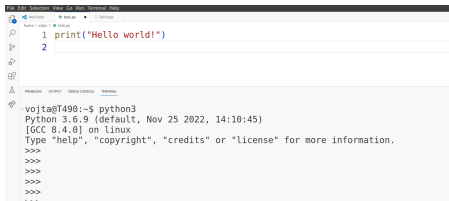
MY CODE WORKS

(I HAVE NO IDEA WHY)

Interaktivní režim (interpret)

- Příkazy jsou přímo zadávány (z klávesnice) do interpreteru, a ihned vyhodnoceny
- Testování krátkých 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!
>>> █
```

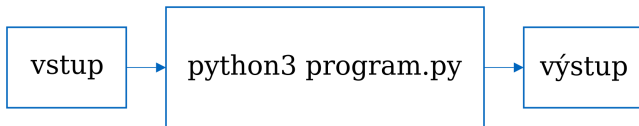


```
1 print("Hello world!")
2
```

```
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.
>>>
>>>
>>>
>>>
```

Spouštění 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

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 prvkům 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'>`

- 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 ‘_’ (podtržítka)
- Může obsahovat písmena, číslice nebo ‘_’
- Jména proměnných jsou case-sensitive
- Nesmí být stejné jako tzv. klíčová slova:

```
False None True and as assert 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

- 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

$$1+1 \rightarrow 1 \quad 1+1.0 \rightarrow 1.0$$

- Dělení: a/b
 - výsledek je vždy float (v Python3)

$$1/2 \rightarrow 0.5 \quad 1/1 \rightarrow 1.0 \quad 4./2 \rightarrow 2.0$$

- 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: $a//b = \lfloor \frac{a}{b} \rfloor$

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

Co je výsledek? $3**2**3//1000$

- $(3**2)**(3//1000) \rightarrow 1$
 - $3**(2**(3//1000)) \rightarrow 3$
 - $(3**(2**3))//1000 \rightarrow 6$
 - $3**2**3//1000 \rightarrow ?$
-
- Vyhodnocování operátorů dle jejich priority
 - Python používá systém PEMDAS (Parenthesis, Exponentiation, Multiplication, Division, Addition, Substraction)
 - Priority operátorů mohou být v různých jazycích jiné
 - Řešením je nařídit prioritu použitím závorek
 - **Vždy používejte závorky pro určení priority operací**

(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

- `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'
```

- 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'
```

Jaká hodnota bude uložena v proměnné b?

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

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`

- 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)
```


- Pro testování se hodí mít vstup připraven v souboru, např. `testInput.txt`
- Tento soubor lze přeměřovat na tzv. standardní 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
```

- 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

```
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
```

- 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
```

- 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
```

- 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

- 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

- 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 == 10:
4     print("true")
5 else:
6     print("false")
```

false

- 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 umožňují opakované vykonání části programu
 - Předepsaný počet opakování (for cyklus)
 - Počet opakování je ukončen podmínkou (while 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, ...
- Funkce `range()`

Iterable object

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

range()

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

For cyklus: range(n)

range(n)

- Vrací n celých čísel $0, 1, \dots, n - 1$
- 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

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 $a, a + 1, a + 2, \dots, b - 1$
- 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
```

range(a,b)

- Vrací celá čísla $a, a + 1, a + 2, \dots, b - 1$
- 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 $a, a + s, a + 2s, a + 3s \dots$, 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

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
```

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)
```

- 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!

```
1 while condition:  
2     code
```

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

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

```
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")
```

- 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
```

```
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
```

- Klíčové slovo `continue` zajistí další iteraci cyklu

Provést opakování pro všechny hodnoty 0, 1 ..., 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)
```


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

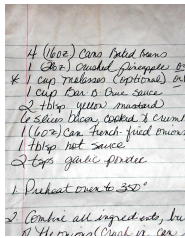
- Postup vyřešení nějaké úlohy
- Slovní popis, rovnice, vývojový diagram, pseudokód
- "Algoritmus" se objevuje i v jiných odvětvích
 - kuchařský recept, chemická reakce, výrobní postup

Implementace

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

Programování

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



Algorithm 1: A*

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

Output: path

```

1 if  $goal(start) = true$  then return  $makePath(start)$ 
2
3  $open \leftarrow start$ 
4  $closed \leftarrow \emptyset$ 
5 while  $open \neq \emptyset$  do
6    $sort(open)$ 
7    $n \leftarrow open.pop()$ 
8    $kids \leftarrow expand(n)$ 
9   forall the  $kid \in kids$  do
10     $kid.f \leftarrow (n.g + 1) + h(kid)$ 
11    if  $goal(kid) = true$  then return  $makePath(kid)$ 
12    if  $kid \cap closed = \emptyset$  then  $open \leftarrow kid$ 
13   $closed \leftarrow n$ 
14 return  $\emptyset$ 

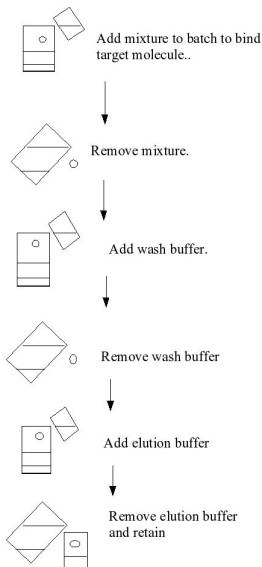
```

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.



- Sekvence instrukcí
 - výpočetní operace
 - práce s pamětí
 - větvení, cykly ...
- Programovací jazyk je způsob zápisu algoritmu pro realizaci na HW
- High-level vs. low-level jazyky

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

```
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
```

- 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
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

- 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)
```

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 (a naopak)
- Pomalý (pokud nejsou výpočty přes kompilované knihovny)
- Umožňuje zápisy, jejichž pochopení vyžaduje detailní znalosti Pythonu (nevhodné pro začátečníky)