Elágazások és Ciklusok

Oliver Kiss
Central European University

August 10, 2021

1 Elágazások

Egyes parancsok feltételhez kötött végrehajtása.

[1]:
```
a = 5
```

[2]:
```
if a>0:
    print('a értéke pozitív')
else:
    print('a értéke nem pozitív')
```

```
a értéke pozitív
```

[3]:
```
# A végrehajtandó parancsokat indentálni kell 4 space karakterrel
```

[4]:
```
a = 0
```

[5]:
```
if a>0:
    print('a értéke pozitív')
else:
    print('a értéke nem pozitív')
```

```
a értéke nem pozitív
```

[6]:
```
if a>0:
    print('a értéke pozitív')
elif a==0:
    print('a értéke zéró')
else:
    print('a értéke negatív')
```

```
a értéke zéró
```

[7]:
```
a = 5
```

[8]:
```
if a>10:
    print('a nagyobb, mint 10')
```

```
elif a>7:
    print('a nagyobb, mint 7')
elif a>3:
    print('a nagyobb, mint 3')
elif a>1:
    print('a nagyobb, mint 1')
else:
    print('a nem nagyobb, mint 1')
```

```
a nagyobb, mint 3
```

[9]:
```
if a>10:
    print('a nagyobb, mint 10')
elif a>7:
    print('a nagyobb, mint 7')
elif a>1:
    print('a nagyobb, mint 1')
elif a>3:
    print('a nagyobb, mint 3')
else:
    print('a nem nagyobb, mint 1')
```

```
a nagyobb, mint 1
```

[10]:
```
# Az első feltétel teljesüléséhez tartozó parancsok kerülnek végrehajtásra
```

2 For ciklus

Véges számú elemet tartalmazó objektum elemein való műveletek elvégzésére.

[11]:
```
for elem in [1, 2, 3]:
    print(elem+1)
```

```
2
3
4
```

[12]:
```
for elem in ["alma", 2, 4.3]:
    print(elem*2)
```

```
almaalma
4
8.6
```

[13]:
```
# Hasznos a range() parancs, ami adott számú integert generál 0-tól kezdve
for x in range(5):
    print(x)
```

```
0
1
2
3
4
```

[14]:
```
# Példa: Adjuk össze az egymást követő számokat a következő listában.
a = [2, 3, 4, 5, 6, 7, 8]
b = []
for i in range(len(a)-1):
    b.append(a[i]+a[i+1])
b
```

[14]:
```
[5, 7, 9, 11, 13, 15]
```

[15]:
```
# Példa: Fibonacci számok
fib = [1, 1]
for i in range(20):
    fib.append(fib[i]+fib[i+1])
```

[16]:
```
fib
```

[16]:
```
[1,
 1,
 2,
 3,
 5,
 8,
 13,
 21,
 34,
 55,
 89,
 144,
 233,
 377,
 610,
 987,
 1597,
 2584,
 4181,
 6765,
 10946,
 17711]
```

[17]:
```
# Iterálhatunk tuple elemein is
for j in (2, 3, 4):
```

```
    print(2**j)
```

```
4
8
16
```

[18]:
```
# Dictionary esetén több megoldás létezik, de maga a dictionary nem iterálható,␣
 ↪csak belőle származtatott objektumok.
a = {'alma':'piros','körte':'barna'}
```

[19]:
```
# Key-eken iterálás:
for kulcs in a.keys():
    print(kulcs)
```

```
alma
körte
```

[20]:
```
# Value-kon iterálás:
for ertek in a.values():
    print(ertek)
```

```
piros
barna
```

[21]:
```
# Párokon iterálás
a.items()
```

[21]:
```
dict_items([('alma', 'piros'), ('körte', 'barna')])
```

[22]:
```
for x in a.items():
    print(x)
```

```
('alma', 'piros')
('körte', 'barna')
```

[23]:
```
# Ezek tuple-k, de a tuple egyszerűen változókba bontható
for kulcs, ertek in a.items():
    print(kulcs)
    print(ertek)
```

```
alma
piros
körte
barna
```

[24]:
```
# String formatting, amit hasznos tudni:
print("a következő szót be fogom helyettesíteni: {}. És tényleg.".
 ↪format("alma"))
```

a következő szót be fogom helyettesíteni: alma. És tényleg.

[25]:
```
print("a következő szót be fogom helyettesíteni: {}. És tényleg. Még egyszer!␣
↪{}. Hurrá!".format("alma",'narancs'))
```

```
a következő szót be fogom helyettesíteni: alma. És tényleg. Még egyszer!
narancs. Hurrá!
```

[26]:
```
for kulcs, ertek in a.items():
    print('A dictionaryben a {} keyhez tartozó value {}'.format(kulcs,ertek))
```

```
A dictionaryben a alma keyhez tartozó value piros
A dictionaryben a körte keyhez tartozó value barna
```

3 While ciklus

Addig hajtjuk újra és újra végre a ciklusmagot, amíg a feltétel fennáll.

[27]:
```
i = 0
while i<10:
    i = i+1
    print('még futok...')
```

```
még futok…
még futok…
még futok…
még futok…
még futok…
még futok…
még futok…
még futok…
még futok…
még futok…
```

[28]:
```
i
```

[28]:
```
10
```

[29]:
```
# Vigyázz: végtelen ciklus veszélye. Megszakíthatod a ctrl+c kombinációval, de␣
↪egyes esetekben gyorsan megtöltheti a számítógép memóriáját és csak hard␣
↪reset segít. Óvatosan vele. Példa:
i = 0
while i<10:
    i=1
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipykernel_11984/3193064053.py in <module>
```

```
      1 # Vigyázz: végtelen ciklus veszélye. Megszakíthatod a ctrl+c␣
 ↪kombinációval, de egyes esetekben gyorsan megtöltheti a számítógép memóriáját␣
 ↪és csak hard reset segít. Óvatosan vele. Példa:
      2 i = 0
----> 3 while i<10:
      4     i=1

KeyboardInterrupt:
```

[30]:
```
# Példa: Találd meg az első 20 négyzetszámot
negyzetszamok = []
i = 1
while len(negyzetszamok)<20:
    if pow(i,0.5)%1==0:
        negyzetszamok.append(i)
    else:
        pass
    i = i+1
```

[31]:
```
negyzetszamok
```

[31]:
```
[1,
 4,
 9,
 16,
 25,
 36,
 49,
 64,
 81,
 100,
 121,
 144,
 169,
 196,
 225,
 256,
 289,
 324,
 361,
 400]
```

4 Break és continue statementek

[32]:
```
# Break megszakítja a legbelső for vagy while ciklust
for i in range(100):
    if i>10:
        break
    else:
        print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
```

[33]:
```
i = 0
while True:
    i = i+1
    if i>4:
        break
    else:
        pass
```

[34]:
```
i
```

[34]: 5

[35]:
```
# A continue ciklus a következő cikluselem végrehajtására utasítja a programot
i = 0
while True:
    i = i+1
    if i>4:
        break
    else:
        pass
    print(i)
```

```
1
2
3
4
```

[36]:
```
# A continue ciklus a következő cikluselem végrehajtására utasítja a programot
i = 0
while True:
    i = i+1
    if i>4:
        break
    else:
        continue
    print(i)
```

5 Egy különlegesség: for-if-end

[37]:
```
# Egy for ciklusba ágyazott feltétel, amely ha egyik elemre sem teljesül, az␣
 ↪else statement kerül végrehajtásra.
for i in range(10):
    if i>10:
        print('{} nagyobb, mint 10'.format(i))
    else:
        print('{} nem nagyobb, mint 10'.format(i))
```

```
0 nem nagyobb, mint 10
1 nem nagyobb, mint 10
2 nem nagyobb, mint 10
3 nem nagyobb, mint 10
4 nem nagyobb, mint 10
5 nem nagyobb, mint 10
6 nem nagyobb, mint 10
7 nem nagyobb, mint 10
8 nem nagyobb, mint 10
9 nem nagyobb, mint 10
```

[38]:
```
# Egy for ciklusba ágyazott feltétel, amely ha egyik elemre sem teljesül, az␣
 ↪else statement kerül végrehajtásra.
for i in range(10):
    if i>10:
        print('{} nagyobb, mint 10'.format(i))
else:
    print('egyik szám sem nem nagyobb, mint 10')
```

```
egyik szám sem nem nagyobb, mint 10
```

[39]:
```
# Példa: Keressük meg 2 és 30 között a prímeket
for n in range(2, 30):
    for x in range(2, n):
        if n % x == 0:
            print('{}={}*{}'.format(n,x,n//x))
```

```
            break
    else:
        print('{} egy prím'.format(n))
```

```
2 egy prím
3 egy prím
4=2*2
5 egy prím
6=2*3
7 egy prím
8=2*4
9=3*3
10=2*5
11 egy prím
12=2*6
13 egy prím
14=2*7
15=3*5
16=2*8
17 egy prím
18=2*9
19 egy prím
20=2*10
21=3*7
22=2*11
23 egy prím
24=2*12
25=5*5
26=2*13
27=3*9
28=2*14
29 egy prím
```

6 Egy hasznos dolog: list comprehension

[40]:
```
# Egy soros, egészen komplikált lista átalakítások
a = [2, 3, 4]
```

[41]:
```
[x*2 for x in a]
```

[41]:
```
[4, 6, 8]
```

[42]:
```
[x+1 for x in a]
```

[42]:
```
[3, 4, 5]
```

[43]:
```
[x for x in a if x>2]
```

[43]:
```
[3, 4]
```

[44]:
```
b = [-2, -1, 0, 1, 2]
```

[45]:
```
[x if x>0 else 0 for x in b]
```

[45]:
```
[0, 0, 0, 1, 2]
```

[46]:
```
[a[i]+a[i+1] for i in range(len(a)-1)]
```

[46]:
```
[5, 7]
```

[47]:
```
[a[i]+b[i] for i in range(min([len(a),len(b)]))]
```

[47]:
```
[0, 2, 4]
```

[48]:
```
# Egy bonyolultabb példa: szorzótábla
for i in range(11):
    sor = []
    for j in range(11):
        sor.append(i*j)
    print(sor)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
[0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40]
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
[0, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60]
[0, 7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
[0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80]
[0, 9, 18, 27, 36, 45, 54, 63, 72, 81, 90]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

[49]:
```
[print([i*j for i in range(11)]) for j in range(11)]
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
[0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40]
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
[0, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60]
[0, 7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
[0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80]
```

```
[0, 9, 18, 27, 36, 45, 54, 63, 72, 81, 90]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

[49]: `[None, None, None, None, None, None, None, None, None, None, None]`