Digitális kultúra

Digitális kultúra

Függvénytár

Tartalomjegyzék

Ebben a dokumentumban a címével ellentétben, nem csak függvények szerepelnek. Három témakörhöz tényleg függvények szerepelnek, egy negyedikhez – az utolsó két részben – azonban már nem függvények vannak összegyűjtve.

Úgy lehet erre a dokumentumra tekinteni, mint matematikából a Négyjegyű függvénytáblázatra: összegyűjti a Python függvényeit, amelyek szerepeltek vagy szerepelhettek volna órákon; a táblázatkezelő függvényeit, amelyek szerepeltek órán vagy szerepelhetnek érettségin; adatbáziskezeléshez az SQL függvényeket; valamint a HTML nyelv elemeit és az azok formázásához használható CSS tulajdonságokat.

Python függvények

1. Beépített Python függvények

abs(x)

Visszaadja a paraméterben szereplő számérték abszolút értékét. A paraméter lehet egész, lebegőpontos szám, vagy komplex szám.

aiter(async_iterable)

A paraméterében szereplő aszinkron iterálható értékhez ad egy aszinkron iterátort. Az **iter()** függvénytől eltérően nincs kétparaméteres változata.

all(iterable)

A paraméterében szereplő iterálható érték minden elemének igazként értelmezhető értéke esetén igaz (True) értéket ad vissza, különben hamisat. Akkor is igaz értéket ad, ha a paraméterben megadott iterálható érték üres.

anext(async_iterator[, default])

Az első paraméterben szereplő aszinkron iterátor következő értékét adja vissza. Amennyiben az iterátornak nincs több eleme és szerepel a második paraméter, akkor azt adja eredményül. (Ez a **next()** függvény aszinkron megfelelője.) Ha az iterátor kiürült és nincs második paraméter, akkor a *StopAsyncIterator* hiba keletkezik.

any(iterable)

Igaz értéket ad vissza, ha a paraméterben szereplő iterátor érték bármely eleme igazként értelmezhető. Amennyiben a paraméterben levő értéknek nincs eleme, akkor hamis (False) értéket ad vissza.

ascii(object)

Egy str szöveget ad vissza, amely a paraméterben megadott objektum szöveges reprezentációja, de a **repr()** függvénytől eltérően a nem ASCII karaktereket azok escape sorozatával helyettesíti.

bin(x)

A paraméterben szereplő egész szám bináris megfelelőjét tartalmazó karaktersorozatot adja vissza, amely a „0b” szöveggel kezdődik. Ez a karaktersorozat egy legális Python kifejezés. Amennyiben a paraméter nem *int* típusú, akkor a függvény meghívásakor az értéke *int* típusra konvertálódik – amennyiben az lehetséges.

bool(x)

A bool típus konstruktorfüggvénye. Amennyiben az *x* értéke igaz értékként értelmezhető, akkor True értéket eredményez. Ha *x* hamisként értelmezhető vagy hiányzik, akkor a False értéket eredményezi.

bytearray(...)

Az azonos nevű típus konstruktorfüggvénye. A *bytearray* 0 <= x < 256 értékek módosítható tömbje.

bytes(...)

Az azonos nevtű típus konstruktorfüggvénye. A *bytes* az *str*-hez hasonló, nem módosítható tömb, amelynek értékei byte-ok, azaz 0 <= x < 256 értékű egészek lehetnek.

callable(object)

Igaz értéket ad eredményül, ha a paraméterében szereplő objektum egy függvényként meghívható objektum, hamis értéket különben.

chr(i)

Visszaadja a paraméterben megadott kódú karaktert (egy karakteres *str* értékként) a Unicode kódtábla szerint. Például a chr(97) értéke 'a', míg a chr(8364) értéke a '€' karakter. A függvény az **ord()** függvény inverze.

complex(real=0, imag=0)

complex(string)

Az azonos nevű típus konstruktorfüggvénye. A létrejövő komplex száma *real* + *imag***i* alakú lesz. A második formában a konvertálandó szövegnek a megfelelő alakúnak kell lennie. Amennyiben egy paraméter szerepel, akkor az lehet szám (a *real* kapja az értéket) vagy szöveg (a *string* kapja az értéket), a második paraméternek mindig számnak kell lennie (és ha szerepel, akkor az elsőnek is). A szám lehet *int* vagy *float* típusú. Ha nincs paramétere, akkor a `0j` értéket adja.

dict()

A szótár adattípus konstruktorfüggvénye. Paraméter nélkül egy üres szótárat hoz létre.

dir()

Paraméter nélkül az aktuális lokális tér neveinek listáját adja vissza. Paramétere megadása esetén annak az objektumnak az attribútumaiból álló listát.

divmod(a, b)

A két (nem komplex) paraméter egész osztását végzi el. A visszaadott tuple első eleme az osztás eredménye, a második a maradéka (azaz a visszaadott érték: `(a // b, a % b)`) Amennyiben a paraméterek lebegőpontos számok, akkor a visszaadott érték `(q, a%b)` lesz, ahol *q* a `math.floor(a/b)` eredménye.

enumerate(iterable, start=0)

Ez a függvény egy szekvenciából vagy más iterálható típusú adatból egy iterátort állít elő, amelynek elemei két elemű tuple-k, ahol az első elem a szekvencia indexe, a második pedig az értéke.

filter(function, iterable)

Egy iterátort hoz létre, amelybe az *iterable* azon elemei kerülnek bele, amelyre a *function* paraméternél megadott függvény True értéket ad eredményül. A függvénynek csak a nevét kell megadni, és olyannak kell lennie, amely egy paramétert vár – ez lesz az iterálható érték vizsgált elemének értéke.

Ha a *function* értéke None, akkor azok az elemek maradnak meg az *iterable* elemei közül, amelyek értéke igaz.

A függvény hívás azzal egyenlő, mintha az alábbi generáló kifejezést adnánk meg:

```
(item for item in iterable if function(item))
```

Illetve ha a function paraméter értéke None:

```
(item for item in iterable if item)
```

float()

A lebegőpontos számokat tároló típus konstruktorfüggvénye.

format(value, format_spec='')

A *value* paraméter értékét a *format_spec* által megadott formába konvertálja. A formátumszöveg megadására a **str.format()** függvénnyel azonos szabályok érvényesek.

frozenset(iterable)

Az azonos nevű, nem módosítható halmazt megvalósító típus konstruktorfüggvénye.

hash(object)

A paraméterben szereplő objektum értékének *hash*-értékét adja, ami mindig egy egész szám. Ezeket az értékeket lehet felhasználni például két szótár-kulcs összehasonlítására.

A halmaz elemei, a szótár kulcsai, az egész, valós számok, a *tuple*-k mind rendelkeznek egy-egy *hash* értékkel, ezért nevezzük őket h*ash-elhető típus*oknak. A halmaz elemeire illetve a szótár kulcsaira például előírás, hogy *hash-elhető típusúnak* kell lennie. Numerikus értékek, amelyek összehasonlítva azonos értékűnek számítaniak (mint például az 1 és az 1.0) azonos *hash* értékkel rendelkeznek.

help()

Az interpreter interaktív használata során ezzel a függvénnyel lehet segítséget kérni. Paraméter hiányában az interaktív súgó rendszer indul el a hatására. Paraméter megadása esetén az adott névvel rendelkező objektumról kapunk leírást. Akár saját függvényről is kaphatunk leírást, ha annak a dokumentálását megfelelően elvégeztük.

hex(x)

A paraméterében megadott egész számot az azt tizenhatos számrendszerben reprezentáló szöveggé konvertálja, a „0x” prefixummal együtt.

id(object)

Visszaadja az objektumot azonosító egész számot. Ez garantáltan egyedi azonosítója az objektumnak. Azonban ha van két olyan objektuma a programnak, amelyeken egy időben biztosan nem léteznek, azoknak lehet ugyanaz az azonosítójuk.

A **CPython** esetében ez az egész szám valójában az adott objektum memóriacíme.

input(prompt)

Amennyiben szerepel a *prompt* paraméter, akkor azt, mint szöveget kiírja a szabványos kimenetre (általában a képernyőre), majd utána várja a választ a szabványos bemenetről (általában a billentyűzetről). A bemenetről az azt lezáró újsor jelig érkezett karaktereket *str* típusú szövegként visszaadja az újsor jel nélkül.

int()

Az egész számok tárolására szolgáló típus konstruktorfüggvénye.

iter(iterable)

A függvény egy iterátort hoz létre a paraméterében megadott iterálható típusú értékből.

len(s)

Visszaadja a paraméterében levő összetett típusú érték elemszámát.

list()

A lista adattípus konstruktorfüggvénye.

*map(function, iterable, *iterables)*

Egy iterátort hoz létre, amely a *funtcion* paraméterrel megadott függvényt alkalmazza az *iterable* paraméter minden elemére. Amennyiben további iterálható értékű paraméterek is vannak, akkor a function függvénynek annyi paramétert kell fogadnia, ahány paraméter szerepel és minden paraméterből mindig a soron következő értéket fogja paraméterként megkapni. A legkisebb elemszámú paraméter elfogyásakor a függvény által adott iterátor is véget ér.

max()

A paraméterében megadott értékek maximumát adja eredményül. A paraméterek lehetnek iterálható értékek (egy vagy több), vagy lehet konkrét értékek felsorolása önálló paraméterekként.

open(filename, mode, encoding=None)

A függvény az első paraméterében megadott nevű állományt kísérli meg megnyitni. Ha sikerül, akkor egy állomány-objektumot ad eredményül, amelyen az állomány a második paraméterben megadott módon lehetséges műveletekkel elérhető.

A második paraméter az állomány megnyitásának módját határozza meg.

- 'r' vagy a paraméter hiánya esetén az állomány szöveges állományként olvasásra kerül megnyitásra. Ekkor hibát eredményez, ha az állomány nem található.
- 'w' esetén az állomány szöveges állományként írásra kerül megnyitásra. Ekkor ha az állomány már létezik, a tartalma törlésre kerül, ha nem létezik, akkor létrejön.
- 'a' esetén az állomány szöveges állományként úgy kerül írásra megnyitásra, hogy az újonnan beleírt tartalom a már meglevő tartalom mögé kerül hozzáfűzésre (append).
- '+' esetén az állomány írásra és olvasásra is használhatóan kerül megnyitásra, ekkor lehet az állományban szabadon a pozíciót változtatni a **.seek()** metódussal.
- Amennyiben a paraméterben a b karakter is szerepel, akkor az állomány bináris állomány kerül megnyitásra. Ha a b helyett a t karakter szerepel, vagy nem szerepel ezek egyike sem, akkor az állomány szöveges állományként kerül megnyitásra.

A harmadik paraméternek szöveges állományok esetén van értelme: a karakterkódolást lehet megadni, ha az eltér az *utf-8* karakterkódolástól, ami az alapértelmezett.

ord(c)

A paraméterben megadott szöveg, amelynek egyetlen Unicode karaktert kell tartalmaznia, a Unicode kódtábla szerinti sorszámát, más néven *karakterkódját* adja eredményül. Ez a függvény a **chr()** függvény inverze.

pow(base, exp, mod=None)

A függvénnyel hatványozni lehet. Amennyiben a harmadik paramétert nem használjuk, akkor a függvény eredménye megegyezik a *base****exp* kifejezés értékével. A harmadik paraméter akkor az abban szereplő számmal osztva keletkező maradékát adja a hatványnak (modulo művelet). Ez gyorsabban végrehajtva a `pow(`*base*`,` *exp*`)` `%` *mod* kifejezés értékét jelenti.

*print(*objects, sep=' ', end='\n', file=None, flush=False)*

A függvény kiírja az *objects* kifejezések értékét egy szöveges állományba. Az értékek közötti elválasztásra a *sep* paraméter értékét is kiírja (ez alapértelmezésben egy szóköz karakter). Az utolsó érték kiírása után az *end* paraméterrel megadott karaktersorozat is kiírásra kerül, amely alapértelmezésben egy sortörés, így alaphelyzetben a **print()** a kiírás végén új sort is kezd.

A szöveges állomány, amelybe a kiírás történik, a *file* paraméterrel adható meg. Amennyiben ezt nem adjuk meg, az alapértelmezett None érték azt jelenti, hogy a kiírás a szabványos kimenetre történik, amely általában a képernyő – grafikus felületen az a terminálablak, amelyben a program fut. Amennyiben a függvénynek nincs paramétere, akkor csak az újsorjelet írja ki.

A sep, end, file és flush paramétereken kívül minden más paraméter értéke **str** típussá konvertálódik az **str()** konstruktorfüggvénnyel.

A *flush* paraméter arra szolgál, hogy amennyiben a kiírás célja egy valódi állomány, akkor a kiírást a paraméternek adott True értékkel ki lehessen kényszeríteni. Ebben az esetben, ha az állomány egyébként pufferelt módban van – azaz nem azonnal történik meg a kiírás, hanem amikor a puffer megtelt –, akkor is azonnal megtörténik a kiírás. False érték esetén az állomány alap működése érvényesül, azaz ha van puffer, akkor annak megtelésekor történik majd csak meg a fizikai kiírása a tartalomnak.

Amennyiben a *file* paramétert nem használva a kimenet célja a képernyő, akkor a *flush* paraméternek sincs hatása, hiszen a kiírás a képernyőre nincs késleltetve.

class property(fget=None, fset=None, fdel=None, doc=None)

Ez az objektumokon belül használható az objektum egy belső tulajdonságának elérésére az objektum-orientált programozásnál használt módon. *fget* egy függvény, amellyel a tulajdonság értékét le lehet kérdezni (*getter*), *fset* egy függvény, amellyel a tulajdonság értékét be lehet állítani (*setter*), *fdel* egy függvény, amellyel a tulajdonság értékét törölni lehet, *doc* pedig egy dokumentáló szöveg (*docstring*) megadására ad lehetőséget.

A függvény pontos használatáról a hivatalos dokumentációban lehet tájékozódni, itt az objektum-orientált programozással nem foglalkozunk...

class range(stop)

class range(start, stop, step=1)

Bár függvényként használjuk, igazából ez a *range* típus konstruktora, amely a paramétereivel megadott értéksorozatot adó iterátort adja. Egyparaméteres változatban a *start* értéke 0 lesz.

A létrejövő iterátor egymás után fogja *start*-tól addig adni a *step* által megadott növekménnyel az egész számokat, amíg azok kisebbek *stop* értékénél.

repr(object)

Ez a függvény visszaadja a paraméterben megadott objektum szöveges reprezentációját. Sok típus esetén ez gyakorlatilag megegyezik az objektum értékének szöveggé alakított változatával, mint amit az **str()** is adna. De más típusok esetén (mint például amit a **range()** állít elő) a típus nevét és még esetleg egyéb információkat is tartalmazó szöveg lesz az eredmény.

reversed(seq)

A *seq* szekvencia elemeit fordított sorrendben adó iterátort állít elő a függvény.

round(number, ndigitd=None)

A függvény a *number* szám értékét kerekítve adja vissza. A pontosságot adja meg az *ndigit* értéke, amely az alapértelmezett None értékre egész számra kerekítést jelent. Annyi tizedes jegyre történik a kerekítés, amennyi az *ndigit* értéke. Ha a második paraméter nem szerepel, akkor a függvény értéke **int** típusú lesz, különben **float**. A kerekítés a matematikai szabályok szerint történik. A második paraméter értéke lehet negatív szám is, akkor az egésznél nagyobb értékre történik a kerekítés, például a `round(123, -2)` értéke 100 lesz.

class set(iterable)

Egy új, halmaz típusú objektumot ad vissza, amelynek elemei lesznek az opcionálisan megadott *iterable* paraméter elemei.

*sorted(iterable, /, *, key=None, reverse=False)*

Az *iterable* paraméter elemeinek rendezett sorozatából álló listát ad eredményül (ha a paraméter is lista, annak a másolata lesz a rendezett lista, amit visszaad).

A *key* paraméterrel megadható egy függvény, amely alapján a rendezés történik. Ennek a függvénynek egy paramétere kell legyen.

A *reverse* paraméterrel előírható, hogy az alapértelmezett növekvő rendezés helyett csökkenő rendezés történjen. Értéke False (alapértelmezett, növekvő rendezés) vagy True (csökkenő rendezés) lehet.

class str()

Ez az **str** típus konstruktorfüggvénye.

sum(iterable, /, start=0)

A függvény összeadja az *iterable* paraméterben levő számértékeket és opcionáisan megnöveli azt a *start* paraméter értékével.

A függvény másik működési módja, ha *iterable* értékei szövegek. Ekkor nem összeadja, hanem összefűzi azokat. Ennél jobb megoldás a `''.join(sequence)` kifejezés, amely a *sequence* elemeinek összefűzését eredményezi.

class tuple(iterable)

Ez a tuple adattípus konstruktorfüggvénye.

class type(object)

*class type(name, bases, dict, **kwds)*

Egy paraméter esetén, az objektum típusát adja meg.

Háromparaméteres esetben egy új típusú osztályt lehet így dinamikusan létrehozni (ez már megint az objektum-orientált programozás területe, így nem részletezzük itt).

vars(object)

A paraméterében megadott modul, osztály, vagy más objektum `__dict__` attribútumának értékét adja vissza, amely nem más, mint az adott objektumban létező változók listája.

Paraméter nélkül használva megegyezik a **locals()** függvény meghívásával.

*zip(*iterables, strict=False)*

A paraméterében megadott iterálható objektumokon párhuzamosan végighaladva előállítja azon *tuple*-értékek listáját, amelyek első eleme az első paraméterből, második eleme a második paraméterből... áll.

A függvény valójában a tuple-k listáját iterátorként adja, azaz a következő érték akkor keletkezik, amikor arra szükség van (pl. a **for** ciklus következő ciklusmagja indulásakor).

Alapértelmezésben a függvény addig produkálja az elemeket, amíg az első paraméternek van eleme. Amikor az első paraméter elfogy, a függvény nem hoz létre új elemeket.

Amennyiben az iterálható paraméterek elemszáma nem egyenlő, és a *strict* paraméter értéke True, akkor a *ValueError* hiba keletkezik. A *strict* hamis értéke esetén nem keletkezik hiba, de nehezen felderíthető szemantikai hiba keletkezhet eltérő elemszám esetén.

2. Egész típus műveletei

Az alábbiakban az int típusra alkalmazható műveletek felsorolása következik. Az egyes műveletek megadásánál az **int** egy egész típusú kifejezést jelent. Amennyiben a kifejezés nem egy változó, akkor az egyértelműség kedvéért kerek zárójelbe kell tenni.

int.bit_length()

Visszaadja az *int* érték tárolásához szükséges bitek számát, beleértve az előjelbitet és szükséges vezető nullákat is.

```
>>>n = -37
>>>bin(n)
'-0b100101'
>>>n.bit_length()
6
```

int.bit_count()

Az *int* értékének abszolút értékét kettes számrendszerben ábrázolva, az 1-es bitek számát adja meg.

```
>>>n = 19
>>>bin(n)
'0b10011'
>>>n.bit_count()
3
(-n).bit_count()
3
```

*int.to_bytes(length=1, byteorder='big', *, signed=False)*

Az *int* értékét reprezentáló bitsorozatot byte-tömbként, tehát *bytes* típusú értékként adja vissza.

```
>>>(1024).to_bytes(2, byteorder='big')
b'\x04\x00'
>>>(1024).to_bytes(10, byteorder='big')
b'\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00'
>>>(-1024).to_bytes(10, byteorder='big', signed=True)
b'\xff\xff\xff\xff\xff\xff\xff\xff\xfc\x00'
>>>x = 1000
>>>x.to_bytes((x.bit_length() + 7) // 8, byteorder='little')
b'\xe8\x03'
```

Az egész érték *length* byte-on lesz megadva, amennyiben ennyi byte nem alkalmas a számérték megadására, akkor *OverflowError* keletkezik.

A *byteorder* határozza meg, hogy a byte-sorrend a *nagy-elöl* vagy a *kicsi-elöl* formában értendő-e.

A SIGNED paraméter mondja meg a metódusnak, hogy a számot előjeles (True) vagy előjel nélküli (False, alapértelmezés) formában kell-e megvalósítani.

Az alapértelmezett értékekkel könnyedén lehet egy számot egybyte-os értékké alakítani. Azonban ilyenkor az átalakítandó szám nem lehet nagyobb 255-nél!

```
>>>(65).to_bytes()
b'A'
```

*classmethod int.from_bytes(bytes, byteorder='big', *, signed=False)*

A *bytes* paraméterben szereplő bytes típusú bitsorozatot egész számmá alakítja. Használata során közvetlenül az **int** típusnév után kell a függvénynevét írni (ennek okába az objektum-orientált programozás miatt nem célszerű belemenni).

```
>>>int.from_bytes(b'\x00\x10', byteorder='big')
16
>>>int.from_bytes(b'\x00\x10', byteorder='little')
4096
>>>int.from_bytes(b'\xfc\x00', byteorder='big', signed=True)
-1024
>>>int.from_bytes(b'\xfc\x00', byteorder='big', signed=False)
64512
>>>int.from_bytes([255, 0, 0], byteorder='big')
16711680
```

A *byteorder* és a *signed* paraméterek az **int.to_bytes()** azonos nevű paramétereivel megegyező jelentésűek.

int.as_integer_ratio()

Két egész értéket ad eredményül, amelyekből törtszámot képezve, azok hányadosa megegyezik az eredeti szám és egy egész nevező hányadosával.

```
>>>a = 3
>>>a.as_integer_ratio()
(3, 1)
>>>a = -3
>>>a.as_integer_ratio()
(-3, 1)
```

Ugyanez a függvény float típusra is létezik, ahol már érdekesebb eredményeket tud adni.

3. Valós számok műveletei

A következő metódusok float típusú értékekkel végezhetőek el. A függvényfejléc megadásában a float minden esetben egy **float** vagy **int** típusú kifejezést jelentenek. Amennyiben a kifejezés nem egy változó, akkor kerek zárójelek közé kell írni.

float.as_integer_ration()

Két int értékből álló tuple lesz a függvény (metódus) eredménye. A két egész szám azt a legkisebb két egész számot jelenti, amelyek hányadosa megegyezik a float értékével. A számláló mindig pozitív lesz.

```
>>> a=3.14
>>> a.as_integer_ratio()
(7070651414971679, 2251799813685248)
>>> a = -0.1
>>> a.as_integer_ratio()
(-3602879701896397, 36028797018963968)
>>> a = 0.5
>>> a.as_integer_ratio()
(1, 2)
```

Mint a fenti példák mutatják, az irracionális számoknak is lesz eredménye, hiszen a számítógépes számábrázolás véges számú bit felhasználásával történik. A második példa esetében meglepő lehet az eredmény (a (-1, 10) párt várnánk), de ennek az az oka, hogy az 1/10 kettes számrendszerben végtelen szakaszos tört lenne, így csak egy ahhoz közeli érték tárolható a számítógépen.

float.is_integer()

Ezzel lehet ellenőrizni, hogy a *float* értéke egész szám-e:

```
>>> (-2.0).is_integer()
True
>>> (3.2).is_integer()
False
```

float.hex()

Visszaadja a float lebegőpontos érték hexadecimális reprezentációját (szöveg formában).

```
>>> (15.0).hex()
'0x1.e000000000000p+3'
>>> (3.14).hex()
'0x1.91eb851eb851fp+1'
```

classmethod float.fromhex(s)

Az s szövegben levő hexadecimális értéket annak lebegőpontos szám megfelelőjévé alakítja.

```
>>> float.fromhex('0x1.91eb851eb851fp+1')
3.14
>>> float.fromhex('0xaaff00')
11206400.0
```

4. Az str típus metódusai

Az alábbiakban az *str* típus metódusainak egy része következik. Ezek olyan függvények, amelynek a szöveg, amellyel dolgoznak nem paraméterei, hanem a feldolgozandó szöveg után egy pontot követve kell a függvény nevét és a paramétereit megadni.

str.capitalize()

A szöveg olyan módon átalakított változatát adja vissza, amelyben a szavak nagybetűvel kezdődnek – ahogyan az anngol nyelvű címekben szokás írni őket.

str.casefold()

Ez a metódus a szöveg olyan változatát adja vissza, amely jól használható a betűmérettől független (*caseless*) összehasonlításokhoz. Az eredmény hasonló a kisbetűsítést végző **.lower()** metódushoz, de még aggresszívebb, mivel például a ß karaktert is átalakítja 'ss' karaktersorozattá.

str.center(width[, fillchar])

A *width* által megadott szélességű szövegben középre igazítja az eredeti szöveget. A *fillchar* alapértelmezett értéke egy szóköz, ezt használja a balról és jobbról szükséges kitöltő karakternek. A függvény az eredeti szöveget adja vissza, ha az hosszabb, mint a *width* értéke.

str.endswith(suffix[, start[, end]])

Igaz értéket ad eredményül, ha *str* a *suffix* szövegre végződik. A *suffix* paraméterben egy *tuple* is megadható, így több eltérő végződésre is végezhető egyszerre ellenőrzés. Amennyiben a *start* paraméter szerepel, akkor az abban megadott indexen kezdi az ellenőrzést. Az *end* megadásával meghatározható, melyik indexű elemnél érjen véget az ellenőrzés.

str.find(sub[, start[, end]])

Azt a legkisebb indexet adja eredményül, amely pozíciótól *str* tartalmazza a *sub* szöveget. A *start* és *end* paraméterekkel a keresést az *str*[*start*:*end*] szeletre lehet korlátozni. Ez arra lehet jó, ha egy szövegben ugyanazt a résszöveget egymás után többször is meg kell keresni: az első keresés az elejétől indul, majd minden soron következő a megtalált pozíció után indulhat.

A függvény értéke -1, ha *sub* nem található meg *str*-ben (vagy annak vizsgált részében). Tehát a többszörös keresést addig érdemes ismételni, amíg nem -1 a kapott érték.

str.format()

A függvény egy olyan szöveget eredményez, amelybe a kapcsos zárójelekkel jelölt részekbe a függvény paramétereiben szereplő kifejezések értékei helyettesítődnek be. A függvény majdnem úgy működik, mint a formázott szöveg (lásd: Hiba: A hivatkozás forrása nem található. Hiba: A hivatkozás forrása nem található). Az eltérés abban van, hogy a kapcsos zárójelek között nem magát a kifejezést kell szerepeltetni – mert azt a függvény paraméterében kell megadni –, hanem a paraméter indexét, amit ott kell megjeleníteni. Az egyetlen hely talán, ahol érdemes ezt a megoldást alkalmazni az f"…" megoldás helyett, amikor ugyanabban a szövegben ugyanazt a kifejezést többször is meg kell jeleníteni – esetleg más kiírási formátumban –, amit ez a függvény lehetővé tesz, hiszen nincs korlátozva, hogy egy paraméter indexe hányszor jelenhet meg az alapszövegben. Amennyiben csak egy paraméter van, akkor a kapcsos zárójelek közül a 0 index el is hagyható.

Például:

```
>>> "The sum of 1 + 2 is {0}".format(1+2)
'The sum of 1 + 2 is 3'
```

str.index(sub[, start[, end]])

Mint a .find(), de ez nem -1 értékkel, hanem a ValueError hibával jelzi, ha nincs benne a *sub* az *str*-ben.

str.isalnum()

Igaz értéket ad vissza, ha *str* csak alfanumerikus karaktereket tartalmaz és *len*(*str*)>0.
Alfanumerikus karakter: számjegy, betű lehet.

str.isalpha()

Igaz értéket ad vissza, ha *str* csak betűket tartalmaz és *len*(*str*)>0.

str.isascii()

Igaz értéket ad vissza, ha str minden karaktere az eredeti ASCII kódtáblába tartozik, azaz kódja legfeljebb 127 és *len*(*str*)>0.

str.isdecimal(), str.isdigit()

Bár a két függvény nem teljesen ugyanazokat a karaktereket fogadja el, de számunkra nem lényeges a különbség. A lényeg: akkor igaz, ha minden karakter számjegynek számít és *len*(*str*)>0.

str.isidentifier()

Igaz értéket ad, ha az *str* egy olyan szót tartalmaz, amely megfelel az azonosítóra vonatkozó szabályoknak (betűvel kezdődik és betűvel vagy számmal folytatódik).

str.islower()

Igaz értéket ad, amennyiben legalább egy karaktert tartalmaz az *str* szöveg és minden karakterre, amely betűnek minősül, kisbetű.

str.isnumeric()

Igaz értéket ad, ha *str* nem üres és minden karaktere számkarakter.

str.isprintable()

Igaz értéket abban az esetben ad, ha *str* nem üres és minden benne szereplő karakter nyomtatható karakter.

str.isspace()

Igaz értéket ad, ha csak whitespace karaktereket tartalmaz a szöveg.

str.istitle()

Igaz értéket ad, ha a szöveg szavai mind nagybetűvel kezdődnek, ahogy az angol nyelvben a címeknél szokás.

str.isupper()

Igaz értéket ad, amennyiben legalább egy karaktert tartalmaz a szöveg és minden karaktere, amely betűnek minősül, nagybetű.

str.join(iterable)

A paraméterül kapott iterable elemeit egymás után fűzve tartalmazó szöveget adja eredményül. Az elemek közé az összefűzés során bekerül az *str* szövege. TypeError hiba keletkezik, ha az elemek nem szöveg típusúak.

str.ljust(width[, fillchar])

Visszaadja az *str* szöveg balraigazított változatát *width* karakter szélességben. A jobbról szükséges kitöltés karakterét adja a *fillchar*, amely alapértelmezésben egy szóköz. Amennyiben str hosszabb, mint a width értéke, akkor a visszaadott érték maga az *str* tartalma lesz.

str.lower()

A szöveg kisbetűsített változatát adja eredményül. Ez azt jelenti, hogy a szövegben szereplő minden olyan karaktert, amely betűnek számít, a kisbetűs megfelelőjére cserél ki.

str.lstrip([chars])

A szöveg olyan másolatát adja vissza. Amelynek elejéről a kezdő karakterek eltávolításra kerültek. Hogy mik ezek a *kezdő karakterek*, azt a paraméter határozza meg: A paraméter egy szöveg, amelyben szereplő karakterek alkotják azt a halmazt, amelynek elemeit el kell távolítani a szöveg elejéről. Amennyiben nem szerepel a paraméter vagy *None* az értéke, akkor a *whitespace karakterek* kerülnek eltávolításra.

str.maketrans(x[, y[, z]])

Ez a statikus metódus[1], amely az **str.translate()** függvény számára képes egy fordítótáblát létrehozni. Amennyiben egy paramétere van, annak egy olyan szótárnak kell lennie, amely Unicode értékek (egészek) vagy karakterek és Unicode karakterek közti kulcs-érték párokat ad meg. Amennyiben két paramétere szerepel, annak két azonos elemszámú szövegnek kell lennie. Ebben minden első paraméterbeli karakternek a második paraméterbeli karakter lesz a megfelelője. Amennyiben harmadik paraméter is szerepel, annak is karakterláncnak kell lennie, amelyeket a None értékre kell fordítani.

A végeredményként létrejövő szótárt tudja az **str.translate()** függvény felhasználni arra, hogy egy szövegben a megfelelő karaktercseréket végrehajtsa.

str.partition(sep)

Az *str* szöveget a *sep* első előfordulásánál három részre osztja, amit egy 3 elemű *tuple*-ban ad vissza. Az első elem az *str* sep előtti része, a második maga a *sep*, a harmadik elem pedig a sep utáni része *str*-nek.

str.removeprefix(prefix)

Amennyiben *str* a *prefix* szöveggel kezdődik, akkor *str* ezt követő része, azaz *str*[len(*prefix*):] az eredmény, egyébként az eredeti szöveg. Tehát ez a metódus eltávolítja *str* elejéről a *prefix* prefixumot.

str.removesuffix(suffix)

Amennyiben *str* a *suffix* szöveggel végződik, akkor azt eltávolítja az *str* végéről és az így kapott szöveget adja eredményül. Amennyiben *str* nem *suffix* szöveggel végződik vagy *suffix* üres szöveg, akkor az eredmény *str* egy másolata.

1 A statikus metódus kifejezés az objektum-orientált programozás egyik kifejezése. Arra utal, hogy bár egy adott típusú értéken (objektumon) végezhető a művelet, valójában az értéknek nem kell léteznie a művelet eléréséhez, ellentétben a normál metódusokkal.

str.replace(old, new[, count])

Az *str* olyan másolatát adja eredményül, ahol a szövegben *old* valamennyi előfordulása helyett a *new* szöveg szerepel. Amennyiben az opcionális harmadik, *count* paraméter is szerepel, akkor csak *old* első *count* előfordulásának cseréjére kerül sor.

str.rfind(sub[, start[, end]])

A legnagyobb olyan indexet adja meg, amelyen *str*-ben előfordul a *sub* résszöveg. Amennyiben a két opcionális paraméter szerepel – beleértve azt is, ha csak a *start* –, akkor a keresés csak az *str*[*start*:*end*] szeletben történik. Ha a *sub* nem található a keresett szövegben vagy szövegrészben, akkor -1 értéket ad eredményül.

str.rindex(sub[, start[, end]])

Mint az **str.rfind()**, de amennyiben a keresett *sub* nem található, akkor a *ValueError* hiba keletkezik.

str.rjust(width[, fillchar])

Egy width hosszúságú szöveget ad eredményül, amelyben *str* jobbra igazítva szerepel. Az ehhez a szöveg elejére szükség szerint beszúrandó új karaktereket a *fillchar* határozza meg, amely alapértelmezésben szóköz. Amennyiben *str* hosszabb, mint *width*, akkor eredményül *str*-t kapjuk.

str.rpartition(sep)

Három részre osztja az *str* szöveget, és a három részt egy 3 elemű *tuple* formájában adja vissza, ahol a *tuple* utolsó eleme *sep* utolsó előfordulása utáni része *str*-nek, a második elem maga *sep*, az első pedig az *str*-nek a *sep* utolsó előfordulása előtti része. Amennyiben *str* nem tartalmazza a *sep* paraméterben megadott szövegrészt, akkor a visszaadott *tuple* utolsó eleme tartalmazza a teljes *str* szöveget, míg a *tuple* első két eleme egy-egy üres sztring.

str.rsplit(sep=None, maxsplit=-1)

Egy listát ad vissza, amelynek elemei az *str* azon szeletei, amelyeket egymástól *sep* választja el. Amennyiben *sep* nem szerepel vagy értéke *None* (ez az alapértéke), akkor bármely *whitespace karakter* elválasztó karakternek számít. A *maxsplit* paraméter határozza meg, hogy jobbról indulva mennyi elválasztójelnél történjen meg az elválasztás. Alapértelmezett értéke -1, azt jelenti, hogy minden elválasztásra sor kerül. Ebben az esetben a **str.split()** eredményével azonos eredményt kapunk.

str.rstrip([chars])

Az *str* olyan másolatát kapjuk eredményül, amelynek jobboldaláról a *chars* paraméterrel megadott karakterek, a paraméter hiányában (vagy *None* megadása esetén) a *whitespace karakterek* eltávolításra kerülnek. *chars* értékét mint karakterek halmazát kell értelmezni, azaz a megadott karakterek bármelyike, tetszőleges sorrendben szerepel str végén, az eltávolításra kerül.

str.split(sep=None, maxsplit=-1)

Egy listát ad eredményül, amely *str* azon szeleteit tartalmazza, amelyeket a *sep* által megadott szöveg választja el egymástól. Amennyiben *sep* nem szerepel, vagy az alapértelmezett *None* az értéke, akkor elválasztó karakternek számít bármely *whitespace karakter.* Amennyiben a *maxsplit* paraméter szerepel, akkor a szöveg elejétől kezdve a szétválasztás annyi elválasztó karakterig történik, amennyit a paraméter megad. A -1 alapértelmezett érték jelentése: minden elválasztás megtörténik a szöveg végéig.

Ha a *sep* paramétert megadjuk, akkor ha az ott szereplő karakter közvetlenül egymás után szerepel az *str* szövegben, akkor a köztük levő üres szöveg a lista soron következő eleme lesz, azaz

több elválasztókarakter nem vonódik össze (kivéve az alapértelmezett whitespace karakterek használata esetén).

str.splitlines(keepends=False)

A több sornyi szöveget tartalmazó *str* sorait tartalmazó listát adja eredményül. Alapesetben a sorokat lezáró újsor karaktereket nem tartalmazzák a lista elemei, de ez a *keepends* paraméternek adott True értékkel felülbírálható.

Amennyiben két újsor karakter szerepel az *str* szövegben egymás után, akkor abból a listában egy üres szöveget tartalmazó elem lesz.

Az **str.split()** függvénnyel ellentétben, ha a szöveg végén egy újsor karakter van, akkor a keletkező listába nem kerül bele ezután egy üres szöveg.

str.startswith(prefix[, start[, end]])

Igaz értéket ad eredményül, ha *str* a *prefix* szöveggel kezdődik, illetve *str*[*start*:*end*] a *prefix* szöveggel kezdődik. A *prefix* paraméter lehet egy szöveg vagy egy szövegeket tartalmazó *tuple* is. Utóbbi esetben bármelyik szöveggel kezdődhet a karakterlánc (vagy annak a megadott szelete).

str.strip([chars])

Az *str* szöveg olyan másolatát adja, amelynek elejéről és végéről el vannak távolítva a *chars* paraméterben szereplő karakterek. Amennyiben a paraméter nem szerepel vagy értéke *None*, akkor a *whitespace karakterek* kerülnek eltávolításra. Amennyiben több karakter is szerepel a paraméterben, akkor azt a függvény a karakterek halmazának tekinti, amely karaktereket el kell távolítani. A karakterek eltávolítása az *str* karakterlánc elejéről és végéről az első olyan karakternél ér véget, amely karakter nem szerepel a paraméterben. Ezt követően ha újra szerepel egy olyan karakter, amely a paraméterben benne van, akkor ott már nem kezdődik újra annak eltávolítása.

str.swapcase()

Az *str* olyan másolatát adja eredményül, amelyben a betűk ki vannak cserélve: nagybetű a kisbetűs megfelelőjére, kisbetű a nagybetűs megfelelőjére. Ebből is következik, hogy *s.swapcase().swapcase() == s*.

str.title()

Az *str* olyan másolatát adja eredményül, amelyben a szavak kezdőbetűi nagybetűre vannak cserélve, így a szöveg az angol címek írásmódjának fog megfelelni.

str.translate(table)

Az *str* olyan másolatát adja eredményül, amelyben egyes karakterek a *table* által meghatározott módon kicserélésre kerültek. A table tartalmának előállítására lehet használni az **str.maketrans()** függvényt. A *table* paraméternek kell megmondania, melyik karaktert melyikre kell kicserélni. A paraméter értékének értékpárokat kell tartalmaznia (ez lehet szótár, lista stb.). Így ez a függvény jól használható abban az esetben, ha több karaktert kell egy-egy meghatározottra cserélni (például a magyar ékezetes karaktereket az angol ábécébeli megfelelőjükre.).

str.upper()

Az *str* olyan másolatát adja eredményül, amelyben minden betűnek számító karakter a nagybetűs változatára van cserélve.

str.zfill(width)

Az *str* olyan változatát adjha, amelynek baloldalát az ASCII ’0’ karakterre tölti fel *width* szélességre. Amennyiben van vezető negatív vagy pozitív előjel, azt megfelelően kezeli a függvény. Amennyiben a *width* értéke eleve nem több *str* hosszánál, akkor maga az str lesz az eredmény.

5. Listák függvényei és metódusai

list([iterable])

Ez a konstruktorfüggvény létrehoz egy **list** objektumot (azaz egy listát). Amennyiben van paramétere, az egy szekvencia vagy egy halmaz jellegű adat kell legyen, amelynek elemei lesznek a létrejövő lista elemei. Ha a paraméter elemein értelmezett egy rendezés (a szekvenciák esetében), akkor az az által megadott sorrendben lesznek az elemei a lista elemei.

*sorted(iterable, /, *, key=None, reverse=None)*

Ez a függvény egy listát ad vissza, amelyben az *iterable* elemei találhatóak az elemeken definiált < művelet alapján sorba rendezve.

A függvénynek két opcionális paramétere van, amelyeket a nevükkel lehet megadni:

A *key* segítségével megadható egy egyparaméteres függvény, amely segítségével az egyes elemekből az összehasonlításban résztvevő érték kinyerhető. (pl. ha szövegeket szeretnénk úgy rendezni, hogy a nagybetű a kisbetűvel azonosnak számítson, akkor használható: `key=str.lower`)

A *reverse* paraméter igaz értékre állításával az alapértelmezett növekvő rendezés csökkenő rendezéssé módosítható.

Megjegyzendő, hogy nem minden lista rendezhető, hiszen ha két érték között – azok eltérő típusa miatt – nincs definiálva sorrend, akkor az a lista, amelyben ilyen elemek találhatóak, nem rendezhetőek.

list.append(x)

Az *x* értéket tartalmazó új elemet fűzi a lista végére. Ekvivalens a `list[len(list):] = [x]` utasítással.

list.extend(iterable)

A listát kibővíti a végén az *iterable* paraméterből származó elemekkel. Azonos a `list[len(ist):] = iterable` utasítással.

list.insert(i, x)

A listába beszúrja az *x* értékű elemet úgy, hogy annak indexe *i* lesz. A korábbi *i* indexű elem indexe ebből fakadóan *i*+1 lesz.

list.remove(x)

Az első *x* értékű elemet eltávolítja a listából. Amennyiben a listának nincs *x* értékű eleme, akkor *ValueError* hiba keletkezik.

list.pop([i])

Eltávolítja a paraméterben megadott sorszámú elemet a listából és visszaadja annak értékét. Amennyiben nincs paramétere, akkor a lista utolsó elemét távolítja el és annak értékét adja vissza.[2]

list.clear()

A lista összes elemét eltávolítja. Azonos a `del list[:]` utasítással.

2 A paraméterben a szögletes zárójel annak jelzésére használatos, hogy a tartalma opcionális. A szögletes zárójeleket nem kell szerepeltetni a kódban!

list.index(x[, start[, end]])

Visszaadja az első *x* értékű elem indexét. Amennyiben a listának nincs *x* értékű eleme, akkor a *ValueError* hiba keletkezik.

Amennyiben a *start* paraméter szerepel, akkor az elem keresése ennél az indexnél indul. Amennyiben az *end* paraméter szerepel, akkor ennél az indexnél véget ér a keresés. Azaz úgy értelmezendő ezen paraméterek használata, mintha a velük megadott `list[start:end]` szeleten történne az elem keresése.

list.count(x)

Megadja, hány olyan eleme van a listának, amelynek értéke *x*.

l.sort(, key=None, reverse=False)*

Ez a metódus a **sorted()** függvényhez nagyon hasonlóan működik: Az *l* listát rendezi. Az eredmény helyben keletkezik, nem új listaként (tehát ha az eredeti listát is meg akarjuk tartani, akkor helyette a **sorted()** függvényt kell használni). Paraméterezése megegyezik a **sorted()** függvényével, eltekintve attól, hogy a rendezendő listát nem kell megadni, mivel az *l* lista lesz rendezve.

Ez a metódus helyben módosítja az *l* listát, nem adja vissza azt eredményként.

list.reverse()

Helyben megcseréli a lista elemeinek sorrendjét.

list.copy()

Egy teljes másolatot készít a listáról és azt adja vissza eredményként. A teljes másolatot úgy kell érteni, hogy a két lista egymástól teljesen független lesz a memóriában, így egyik megváltoztatása nincs hatással a másik listára. Ekvivalens a `list[:]` kifejezéssel.

6. Halmazok metódusai

set.isdisjoint(other)

Igaz értéket ad vissza, ha a *set* egyetlen eleme sem szerepel az *other* halmazban.

set.issubset(other)

Azonos a `set <= other` művelettel (*set* és *other* is halmaz típusú). Igaz, ha *set* részhalmaza *other* halmaznak.

set.issuperset(other)

Igaz, ha `other <= set` igaz.

*set.union(*others)*

Visszaadja a *set* és a paraméterben felsorolt halmazok unióját.

*set.intersection(*others)*

Visszaadja *set* és a paraméterben felsorolt halmazok metszetét.

*set.difference(*others)*

Visszaadja azon elemek halmazát, amelyek *set*-ben benne vannak, de nincsenek benne egyetlen olyan halmazban sem, ami a paraméterek között szerepel.

set.symmetric_difference(other)

A *set* és az *other* halmazok szimmetrikus metszetét adja eredményül, azaz azokat az elemeket, amelyek pontosan az egyik halmaznak elemei.

set.copy()

A *set* halmaz teljes másolatát adja eredményül.

A fentebb felsorolt változatai a halmazműveleteknek paraméterként bármely iterálható típusú értéket elfogadnak (amit természetesen szükség esetén halmazzá alakítva használnak). Ellenben ugyanezeknek a műveleteknek az operátorral megvalósított változatai csak halmaz típusú adattal működnek.

6.2. Csak módosítható halmazok műveletei

*set.update(*others)*

A *set* halmazhoz hozzáadja a paraméterben megadott halmazok elemeit.

*set.intersection_update(*others)*

A *set* halmaznak csak azon elemeit tartja meg, amelyek a paraméterben szerepelő valamennyi halmaznak is elemei.

*set.difference_update(*others)*

A *set* halmaznak eltávolítja minden olyan elemét, amely a paraméterben szerepelő halmazok bármelyikében is szerepel.

set.symmetric_difference_update(other)

A *set* halmazba azok az elemek kerülnek – az addigiek helyett –, amelyek vagy a *set*, vagy az *other* halmaznak eredetileg elemei voltak, de nem mindkettőnek.

set.add(elem)

Az *elem* elemet hozzáadja a *set* halmazhoz. Amennyiben már benne van ez az elem, akkor nem történik semmi.

set.remove(elem)

A *set* halmazból eltávolítja az *elem* elemet. Ha az elem nincs a halmazban, akkor a *KeyError* hiba történik.

set.discard(elem)

Mint a **.remove()**, de ha az elem nincs a halmazban, nem történik semmi.

set.pop()

Egy elemet eltávolít a halmazból úgy, hogy az eltávolított elemet visszaadja eredményként. Amennyiben a *set* üres halmaz, akkor a *KeyError* hiba keletkezik.

set.clear()

Minden elemet eltávolít a halmazból. Tehát a *set* üres halmazzá válik.

7. Szótárral végezhető műveletek

A következő műveletek végezhetőek el a *dict* típusú adatokkal.

list(d)

Visszaadja a *d* szótárban szereplő valamennyi kulcsot tartalmazó listát.

len(d)

Visszaadja a *d* szótárban szereplő elemek számát.

d.clear()

Törli a *d* szótár valamennyi elemét.

d.copy()

A *d* szótár teljes másolatát adja vissza.

classmethod fromkeys(iterable [,value])

Egy új szótárt hoz létre a paraméterében szereplő iterálható értékekkel, mint kulcsokkal, amelyeknek a *value* értéket állítja be. Amennyiben a második paraméter nem szerepel, akkor a None értéket kapják a szótár elemei.

d.get(key [, default])

Visszaadja a *key* kulcsú elem értékét vagy ha az nem létezik, akkor a második paraméterrel megadott értéket. Amennyiben nincs második paraméter és a keresett kulcsú elem nincs a szótárban, akkor a None értéket adja vissza, vagyis soha nem keletkezik KeyError hiba.

d.items()

A szótár (kulcs, érték) párjait tartalmazó nézetet ad vissza. (A szótár nézeteiről a szótár adattípus bemutatásánál van szó a 11.osztaly.pdf-ben.)

d.keys()

Visszaadja a szótár kulcsait tartalmazó nézetet.

d.pop(key [, default])

Amennyiben a *key* kulcsú elem benne van a szótárban, akkor annak értékét adja vissza és egyben az elemet eltávolítja a listából. Amennyiben a keresett elem nincs a szótárban, de a második paraméter meg van adva, akkor annak eredményét adja eredményül, különben a KeyError hiba történik.

d.popitem()

A szótárból eltávolít egy elemet, amelynek (kulcs, érték) párját adja eredményül. Üres szótár esetén KeyError hibát eredményez.

reversed(d)

Szótárra hívva egy olyan iterátort ad, amely a szótár elemeit az **d.keys()** függvénnyel ellentétes sorrendben adja.

d.setdefault(key[, default])

A *key* kulcsú elemet a *default* értékre, annak hiányában None értékre állítja. Amennyiben az adott kulccsal nem létezik elem, akkor létrehozza.

d.update([other])

Frissíti a *d* szótárt az *other* szótár elemeivel: Azon elemek, amelyek mindkét szótárban szerepelnek, az *other*-beli értéket kapják; a *d*-ben nem szereplő *other*-beli elemek pedig bekerülnek a d szótárba.

d.values()

A szótárban szereplő elemekben levő értékeket adja nézetként.

8. A math modul

Ebben a modulban matematikai számításokhoz hasznos dolgok találhatóak. Az egyes matematikai függvények, amelyek ebben a modulban találhatóak, a C nyelvben kialakult szabványt követik.

A modul függvényei nem használhatók komplex számokkal, azokhoz a *cmath* modulbeli megfelelő függvényeket lehetne használni – ha mi egyáltalán foglalkoznánk komplex számokkal.

A következőkben a teljesség igénye nélkül szerepelnek a modulban definiált függvények és egyéb objektumok. A modul teljes dokumentációja angolul a https://docs.python.org/3/library/math.html címen érhető el.

Amennyiben a függvény leírásában nem szerepel, a függvény visszatérési értéke mindig *float* típusú.

8.1.1. Számelméleti függvények

math.ceil(x)

Visszaadja a legkisebb egész számot, amely nagyobb vagy egyenlő *x* értékénél. A függvény *int* típusú értéket ad eredményül.

math.fabs(x)

x abszolút értékét adja eredményül, ahogyan a modul nélkül is használható **abs(x)** függvény is. A különbség a két függvény között, hogy az **abs()** függvény egész paraméter esetén egész értéket ad eredményül, míg a **math.fabs()** mindig *float* típusú értéket ad vissza (erre utal a nevében az *f* betű is).

math.factorial(x)

Visszaadja az *x* egész érték faktoriálisát egész számként. Amennyiben *x* nem egy nemnegatív egész szám, akkor hiba keletkezik.

math.floor(x)

Visszaadja a legnagyobb egész számot, amely kisebb vagy egyenlő *x* értékével. A függvény egész értéket ad vissza.

math.fsum(iterable)

Visszaadja a paraméterében megadott *iterable* típusú érték összegét.

Az *iterable* típusról a listák és a **for** utasítás kapcsán lesz szó. Elöljáróban, a függvény megértése céljából annyit érdemes tudni, hogy leginkább számok valamilyen módon előállított sorozataként lehet értelmezni. Így tehát a függvény a paramétere által megadott számsorozat összegét fogja valós értékként előállítani.

math.gcd(egész értékek)

A paraméterben szereplő egész számok legnagyobb közös osztóját adja a függvény egész értékként eredményül, ha minden paraméter nullától különböző értékű. Ha minden paramétere nulla, vagy nincs a függvénynek paramétere, akkor nullát ad eredményül.

math.isfinite(x), math.isinf(x), math.isnan(x)

Ezzel a három függvénnyel a lebegőpontos típus speciális értékeit lehet letesztelni. A **math.isfinite(x)** akkor ad `True` (igaz) értéket, ha *x* egy érvényes véges számérték.

Ellenben a **math.isinf(x)** akkor ad `True` értéket, ha *x* a negatív végtelennek vagy a pozitív végtelennek számító lebegőpontos értéket tartalmazza.[3]

A **math.isnan(x)** függvény akkor lesz igaz, ha az *x* egy olyan értéket tartalmaz, amely a lebegőpontos számok ábrázolására vonatkozó szabvány szerint nem érvényes szám (*NaN*, azaz *N*ot *a N*umber a hivatalos neve az ilyen értékeknek).

math.sqrt(x) és math.isqrt(n)

Ez a két függvény az *x* érték nemnegatív négyzetgyökét számolja ki. A legfőbb különbség a két függvény között, hogy a **math.sqrt(x)** minden esetben *float* típusú, a **math.isqrt(n)** minden esetben *int* típusú eredményt ad. Utóbbi esetében *n* is *int* típusú kell legyen és az eredményül kapott érték a `math.floor(math.sqrt(n)` függvényhívással azonos érték lesz, azaz az eredmény a legnagyobb *a* egész szám lesz, amelyre az $a^2 \leqslant n$ reláció igaz.

math.lcm(egész értékek)

Amennyiben a paraméterek pozitív egész számok, akkor azok legkisebb közös többszörösét adja a függvény. Ha bármelyik paraméter nulla, akkor a visszaadott érték nulla, ha a függvénynek nincs paramétere, akkor 1 a visszaadott érték.

math.prod(iterable)

Ez a függvény a **math.sum()** függvényhez hasonlóan viselkedik, de eredményül az *iterable* típusú érték által adott számértékek szorzatát adja. Van egy speciális paramétere, a *start*, amelynek alapértelmezett értéke 1. Ez a szorzás kezdőértéke. Amennyiben nem szerepel paraméter, akkor ezt az értéket adja eredményül a függvény.

math.trunc(x)

A paraméterben szereplő *x* valós szám egész részét adja *int* típusú értékként.

8.1.2. Hatványozási és logaritmikus függvények

math.exp(x)

A függvény eredménye az e^x hatványozás eredménye, ahol *e* a természetes logaritmus alapszáma (*e* = 2,7182...).

math.exp1(x)

A függvény eredménye az $e^x - 1$ hatványozás eredménye. Ennek a függvények azért van értelme, mert ugyanerre a számításra a `math.exp(x)-1` kifejezés komoly pontosságvesztést okozhat.

math.log(x, base)

A függvény egy vagy két paraméteres változatban is használható. Ha csak az *x* szerepel, akkor a *base* paraméter értéke automatikusan az *e* lesz.

Amennyiben a *base* paraméter is szerepel, akkor ez lesz a logaritmus alapja.

math.log1p(x)

A függvény 1+*x* természetes alapú logaritmusát számolja ki.

3 Ha többet akarsz tudni a lebegőpontos számokról, mint ami ebben a dokumentumban szerepel, akkor a kutatást elkezdheted a Wikipédia https://hu.wikipedia.org/wiki/Lebegőpontos_számábrázolás címen elérhető szócikkén vagy ennek angol változatán: https://en.wikipedia.org/wiki/Floating-point_arithmetic.

math.log2(x), math.log10(x)

Az első függvény *x* kettes alapú logaritmusát adja eredményül. A második függvény *x* tízes alapú logaritmusát adja eredményül.

math.pow(x, y)

A függvény x^y eredményét adja. A C szabvány alapján pow(1.0, x) és pow(x, 0.0) mindig 1.0 értéket eredményez, még akkor is, ha *x* nulla vagy NaN. Ha mind *x*, mind *y* véges, *x* negatív és *y* nem egész, akkor a függvény a *ValueError* nevű hibát eredményezi.

8.1.3. Trigonometrikus függvények

Az itt szereplő függvényekben a szögeket alapértelmezésben radiánban megadva kell érteni.

math.acos(x)

A függvény eredménye radiánban az a szög, amelynek koszinusza *x*, azaz az *r* szög, amelyre $\cos(r)=x$. Az eredmény a [0; π] intervallumba esik.

math.asin(x)

A függvény eredménye radiánban az az *r* szög, amelyre $\sin(r)=x$ és *r* a [$-\frac{\pi}{2}$; $\frac{\pi}{2}$] intervallumba esik.

math.atan(x)

A függvény radiánban azt az *r* szöget adja meg, amelyre $\tan(r)=x$ és *r* a [$-\frac{\pi}{2}$; $\frac{\pi}{2}$] intervallumba esik.

math.cos(x), math.sin(x) és math.tan(x)

Ezek a függvények pontosan azt csinálják, amit mondanak: a cos(*x*), sin(*x*) és tan(*x*) szögfüggvények matematikából ismert értékét adják (természetesen radiánban értve).

Átváltás radián és szög értékek között

A **math.degrees(x)** függvény a radiánban megadott *x* értékre megadja, hogy az hány foknak felel meg. Az ellenkező irányt valósítja meg a **math.radians(x)** függvény: megadja radiánban a szögben megadott *x* értékét.

Hiperbolikus függvények

A következő hiperbolikus függvényeket ismeri a modul:

- **math.acosh(x)**
- **math.asinh(x)**
- **math.atanh(x)**
- **math.cosh(x)**
- **math.sinh(c)**
- **math.tanh(x)**

8.1.4. A modulban definiált konstansok

math.pi

A számítógépen elérhető pontossággal, lebegőpontos típussal a π matematikai konstans.

math.e

A számítógépen elérhető pontossággal az *e* szám, azaz a természetes alapú logaritmus alapszáma.

math.tau

A számítógépen elérhető pontossággal a τ szám (2π).

math.inf

Ez a konstans a pozitív végtelen értékét tartalmazza a számítógépen használt lebegőpontos számábrázolásnak megfelelően. Az érték egyébként megkapható a `float('inf')` kifejezéssel is. Nyilván ha a negatív végtelen értékre van szükség, akkor az a `-math.inf` kifejezéssel érhető el.

math.nan

A lebegőpontos szabvány által definiált *NaN* értéket tartalmazza, amely azt jelenti, hogy ez nem számérték. Ugyanezt az értéket kapjuk a `float('nan')` kifejezéssel is. A szabvány szerint a *NaN* érték nem egyenlő egyetlen számmal sem. Éppen ezért ellenőrizni, hogy egy lebegőpontos érték *NaN*-e nem az `==` vagy az `is` operátorral lehet, hanem a **math.isnan()** függvénnyel.

8.2. A random modul

Ez a modul minden professzionális felhasználási igényt ki tud elégíteni, így ezúttal is csak egy töredéke kerül felsorolásra a modul lehetőségeinek. A véletlenszám-generátor a Mersenne Twister algoritmust használja, amely 53 bites pontosságú lebegőpontos számokat generál a [0.0; 1.0) intervallumban úgy, hogy a generált számok periódusának hossza $2^{19937}-1$. Ez hatalmas szám, így esély sincs arra, hogy a kidobott számok elkezdjenek ismétlődni. A generált számokat közvetlenül a **random.random()** függvénnyel lehet lekérni. A többi függvény az előállított számot valamilyen más értékké alakítva adja vissza.

A modul hivatalos leírásában (https://docs.python.org/3/library/random.html) szerepel egy megjegyzés, miszerint a modul véletlenszám-generátorát nem szabad használni biztonsági, titkosítási célokra, arra a *secrets* modult kell használni.

8.2.1. A modul néhány függvénye

random.seed(a=None, version=2)

A véletlenszám-generátor inicializálását végző függvény. Bár akkor is kapunk véletlen számokat, ha az inicializálás nem történt meg, a biztonság kedvéért minden olyan program, amelyben véletlen számokat kell előállítani, kezdődjön a **random.seed()** függvény meghívásával!

Bár mint látható, a függvénynek két alapértékkel rendelkező paramétere van, általában egy paramétert szokás adni neki. De lehetne azt is elhagyni, ami az Hiba: A hivatkozás forrása nem található. ábrán levő utasítást egyszerűbbé teszi: definíció szerint ugyanis, ha az *a* paraméter nem szerepel, akkor éppen az aktuális rendszeridőt használja a függvény. De ha az operációs rendszer is biztosít véletlenszám-generátort, akkor elképzelhető, hogy azt használja inkább a függvény a rendszeridő helyett – ekkor pedig egy bizonytalan forrást használunk, tehát mindig adjuk meg paraméterként a rendszeridőt!

Ha *a int* típusú, akkor közvetlenül az értéke lesz használva kezdőértékként. A másik paraméterrel a véletlenszám-generátor verzióját lehet beállítani. Ha ez az alapértelmezett 2 értékű, akkor *str*, *bytes*, *bytearray* típusú értékek *int* típusra konvertálódnak a felhasználáshoz. 1-es verziót használva (amely a Python régebbi változatának szimulálására alkalmas) az *str* és a *bytes* típusú értékek szükebb értékhalmazt adnak, mint a 2-es verzió.

Ha a függvénynek egy konkrét egész számot adunk, azt tesztelésre lehet használni, ugyanis ilyenkor a véletlenszám-generátor garantáltan mindig ugyanazt a számsorozatot állítja elő.

random.random()

Ezt tekinthetjük a véletlenszám-generátor alapfüggvényének, mivel közvetlenül a kidobott lebegőpontos számot adja eredményül, amely a [0.0; 1.0) intervallumba esik.

random.randbytes(n)

n byte véletlen értéket generál.

random.randrange(stop)
random.randrange(start, stop [,step])

Véletlenszerűen választ értéket az azonos paraméterekkel megadott **range()** által előállított iterátor[4] értékei közül.

random.randint(a, b)

A függvény egy *n* egész számot ad vissza, amelyre igaz, hogy *a* <= *n* <= *b*, *a* és *b* egész érték.

random.getrandbits(k)

4 Az iterátor bemutatása a listáknál, azon belül is a **for** utasítás ismertetésénél fog szerepelni.

Egy nemnegatív *k* bites egész véletlen számot ad vissza.

random.choice(szekvencia)

Ez a függvény akkor hasznos, ha egy lista valamelyik elemét kell véletlenszerűen kiválasztani, ugyanis pontosan ezt csinálja. A paraméter bármely szekvencia-típus lehet (karakterlánc, lista, iterátor, tuple), annak az egyik elemét adja eredményül.

random.shuffle(x)

A függvény véletlenszerűen összekeveri a paraméterében szereplő, módosítható szekvencia elemeit. Csak olyan szekvencia típusra működik, amely módosítható és amelynek az elemei sorbarendezettek (azaz az elemeknek egyértelmű sorszámuk van). A nem módosítható listák esetén a következő **random.sample()** függvényt célszerű használni, amely ugyan kicsit bonyolultabban paraméterezendő, de az összekevert szekvenciát visszatérési értékként adja.

random.sample(elemhalmaz, k, …)

Ez a függvény egy *k* elemű listát ad eredményül, amelynek elemei az *elemhalmaz*ból származnak úgy, hogy minden elemet csak egyszer vesz ki a függvény az *elemhalmaz*ból. Ha minden elemet ki akarunk választani, de véletlen sorrendben, mint a **random.shuffle()** esetén, akkor a *k* értékének az elemhalmaz számosságát kell megadni a **len()** függvény révén.

Ezen felül a függvénynek még van egy *count* paramétere is, amelynek egy listával meg lehet adni, hogy az *elemhalmaz* egyes elemeit hányszor lehet kiválasztani (azaz hányszor szereplőnek kell tekinteni).

8.3. A time modul

Ez a modul is jóval többet tud annál, ami itt bemutatásra kerül. Vannak még más időkezelést segítő modulok is, például a *datetime* és a *calendar*, amelyekkel nem foglalkozunk. Bár a modul minden Python értelmezőben elérhető, a benne ténylegesen elérhető funkciók az adott számítógépes rendszertől jelentősen függenek.

Mielőtt a modullal megismerkedünk, néhány dolgot a számítógépes időkezelésről érdemes tudni. Először is, a számítógépeken az időt az általunk ismert formában nem lenne túl kényelmes kezelni. Sokkal egyszerűbb az elmúlt időt folyamatosan mérni egy bizonyos kezdőponttól. Ezt az eltelt időt általában másodpercben mérjük. Vagy annak törtrészében, ha nagyobb pontosság szükséges.

Ez a megoldás azért hasznos, mert így egy időtartam megadása során nem kell bonyolultabb számításokat végezni: egyszerűen az időtartam végéből ki kell vonni az időtartam kezdetét. Hasonlóan egy időponthoz egy időtartamot is nagyon egyszerűen hozzá lehet adni, ha mindent a legkisebb lehetséges mértékben számolunk. Természetesen az ember által érthető formába át kell tudni számolni szükség esetén.

A *time* modul is a fenti módon kezeli természetesen az idő múlását: Egy angolul *epoch* néven nevezett időpontot kezdőidőpontnak tekintve minden időpontra az *epoch*-tól az adott időpontig eltelt másodpercek számát tartja számon. Ezt egyébként a `time.gmtime(0)` függvényhívással be is lehet azonosítani, így hamar kideríthető, hogy Unix rendszerekben ez a kezdőidőpont 1970. január 1. 0 óra (UTC[5] szerint).

Miért ennyi? Mert hivatalosan 1970. január 1. a Unix születésének napja.

Tehát például Linux operációs rendszer alatt futó programokban az aktuális időpontot, mint 1970. január 1. 0 óra óta eltelt másodpercek számát kapjuk meg.

time.process_time()

Ez a függvény az aktuális folyamat (a program) összes futási idejét adja meg másodpercben, lebegőpontos számként. Ez egészen pontosan az az idő, amit a CPU erre a folyamatra fordított a folyamat elindulása óta. Azaz ebbe az időbe semmi olyan eltelt idő nem számít be, amikor a folyamat nem kapott futási időt.

time.process_time_ns()

Az előző függvény egész értéket visszaadó változata. Míg a **time.process_time()** az eltelt folyamatidő lebegőpontos értékként megadása során esetleg pontossági hibát eredményez, a **time.process_time_ns()** függvény ezt elkerüli azzal, hogy egész értéket ad vissza. Viszont ez a függvény nanomásodpercben adja az eredményt éppen a pontosság kedvéért.

A **time.process_time()** és a **time.process_time_ns()** függvények alkalmasak arra, hogy valamely tevékenység időigényét megmérjük, ugyanis az ezek által visszaadott időbe nem piszkál bele az, ha közben az operációs rendszer elveszi a programtól a futási jogot egy időre. Ha például egy függvény végrehajtásához szükséges időt szeretnénk megmérni, akkor a folyamat futási idejét egy-egy változóba el kell tárolni a függvény meghívása előtt és után. A függvény végrehajtására igénybe vett CPU-időt adja a két változóban levő másodperc vagy nanomásodperc különbsége.

time.sleep(secs)

A folyamat futását felfüggeszti a paraméterben megadott másodpercre. A paraméter lehet lebegőpontos érték a másodpercnél pontosabb érték megadása érdekében. A tényleges várakozási idő lehet esetleg a kértnél több is, mivel a várakozást az operációs rendszer úgy valósítja meg, hogy más folyamatnak adja a CPU-t, a folyamat folytatását úgy ütemezi, hogy addig legalább *secs* másodperc

5 UTC: a korábban Greenwich középidejekéng ismert (GMT = Greenwich Mean Time) idő.

elteljen. De ha a többi folyamat futása éppen úgy alakul, akkor lehetséges, hogy az ütemezett időpontnál később kap a folyamat újra futási lehetőséget.

Visszatérve az előző két függvényhez, az azok által adott futási időbe logikusan nem számít bele a **time.sleep()** miatti várakozás ideje, hiszen ekkor a folyamat nem fut.

time.time() és **time.time_ns()**

Ez a két függvény az aktuális időt, mint a hivatalos kezdőidőtől (Windows és Unix esetén 1970. január 1. 0 órától) eltelt időt adja vissza. A **time.time()** lebegőpontos értékként az eltelt másodperceket, a **time.time_ns()** pedig egész értékként az eltelt nanomásodperceket adja eredményül.[6]

Bár a függvények hivatalosan 1 másodpercnél pontosabb értéket adnak vissza, a tényleges pontosság az adott számítógép órájának pontosságától függően nem biztos, hogy másodpercnél pontosabb adatot eredményezhet. Ami ennél is fontosabb, ha a program futása közben egy másik folyamat átállítja a gép óráját, akkor előfordulhat az is, hogy egy későbbi függvényhívás a korábbinál időben korábbi értéket ad eredményül.

A modul további elemei megtalálhatóak a modul hivatalos dokumentációjában az alábbi címen: https://docs.python.org/3/library/time.html

6 A modul `_ns` végű névvel rendelkező függvényei a Python 3.7-es verzió óta léteznek.

9. A sys modul

Ez a modul olyan változókhoz és függvényekhez ad hozzáférést, amelyekkel az értelmező szoftverkörnyezetével lehet kapcsolatot teremteni. A modul funkcionalitása bőven túlmutat az általunk vizsgált kereteken, így itt csupán egy jelentősen rövidített kivonata szerepel a modulnak. A modul teljes, angol nyelvű dokumentációja a https://docs.python.org/3/library/sys.html címen érhető el.

sys.argv

Elsősorban ez az a változó, ami miatt itt egyáltalán szóba kerül a modul. A lecke egy korábbi példájában már szerepelt a használata, a változó teljes megértéséhez szükséges lista adattípus pedig egy későbbi leckében fog szerepelni.

Amennyiben parancssorból indítunk egy programot, akkor annak szóközzel elválasztva különböző paramétereket lehet adni. Éppen ez az a hatalmas előnye a karakteres felületnek a grafikus felülettel szemben, amit számítógép működésébe beleszólni kívánó szakemberek kifejezetten értékelnek: ezekkel a paraméterekkel pontosan meg lehet fogalmazni az elvárt tevékenységet. Természetesen ha a grafikus felület használatával indul el egy program, akkor is lehetnek paraméterei, de azokat nem a felhasználó határozza meg, hanem a grafikus felület mögötti programkód maga.

A **sys.argv** változóval ezekhez a paraméterekhez tud a Python program hozzáférni. A változó egy lista típusú értéket tartalmaz. Ennek első eleme, amely a `sys.argv[0]` megadásával érhető el, magának a programnak az indítására használt név. Ennek a Unix alapú rendszereken van igazán értelme, ahol ugyanaz az állomány több néven is elérhető lehet, így vannak olyan programok, amelyek esetében már az is befolyásolja a működést, hogy melyik nevet használva történik az elindításuk.

A lista további elemei a program indítására kiadott parancs paraméterei a megadásuk sorrendjében. Előretekintve a listákra, úgy lehet tekinteni erre a változóra, mintha az értelmező a programot indító parancsra mint szövegre alkalmazta volna a `.split(" ")` műveletet és az eredményül kapott listát tárolta volna el a változóban (valószínűleg tényleg ezt teszi).

A Linux több rendszerparancsa is úgy van megvalósítva, hogy néhány parancs mögött ugyanaz a program található. Például a különböző állományrendszerre történő formázásra az mkfs parancs olyan változatai is vannak, amelyek neve egy pont után az állomány rendszer nevével van kiegészítve (pl. mkfs.ext, mkfs.fat stb.). Ezeket beírva parancsnévként valójában az mkfs program indul el, de az indítására használt programnévből tudja, hogy melyik típusú állományrendszerre kell a formázást végeznie.

sys.exit(paraméter), sys._exit(paraméter)

A **sys.exit()** függvény egy **SystemExit** nevű *kivételt* generál. Amennyiben ez a kivétel nem kerül kezelésre, illetve ha a függvény meghívása a főprogramban történik (ahonnan már nem kezelhető a kivétel), akkor a program véget ér. A paraméterben levő érték a program hibakódja lesz. Amennyiben ez az érték 0, az a program szabályos befejezését jelzi, az ettől eltérő értékkel valamilyen hibát lehet jelezni. De a paraméter lehet egy szöveg is, amely ebben az esetben a programot indító parancsértelmezőben megjelenő hibaüzenet is lehet.

Mivel ennek a függvénynek a meghívása nem jelenti egyértelműen a program befejezését, ha valahol mindenképpen a program befejezését kell elérni, akkor általában a **sys._exit()** függvény meghívását szokták alkalmazni, amely nem kivételt generál, hanem azonnal befejezi a programot, ugyanígy a paraméterben megadott értéket a program hibakódjaként használva. A két függvény között még az a különbség, hogy a **sys.exit()** paramétere elhagyható, míg a **sys._exit()** függvény paraméterét kötelező megadni.

sys.float_info

Ez a változó a *float* típusról ad információkat egy olyan *tuple* segítségével, amelyben az egyes értékeknek neve is van. Tehát ezen változón keresztül ki lehet deríteni, hogy a programot futtató értelmező pontosan milyen pontosságú lebegőpontos számokat tud használni.

sys.int_info

Hasonlóan, ez a változó az *int* típus adott interpeterbeli megvalósításaira jellemző adatokat tartalmazza.

sys.maxsize

Egy egész értéket tartalmaz, amely legnagyobb érték, amelyet az *int* típus tartalmazhat. Magyarul: az adott interpreter esetén alkalmazható legnagyobb egész szám.

sys.modules

Ez a változó a programba pillanatnyilag beimportált modulok nevét tartalmazza.

sys.path

Ez a változó szövegek listájaként tartalmazza azoknak a számítógépes könyvtáraknak az elérési útvonalait, amelyekből az **import** utasítással betöltendő modulokat az értelmező keresi. A program indításakor a PYTHONPATH környezeti változó értéke másolódik be ebbe a változóba. A program futása során ez a változó szabadon változtatható, de mindenképpen karakterláncok listájának kell lennie az értékének.

sys.stdin, sys.stdout, sys.stderr

Ez a három változó a szabványos bemenet (alapértelmezésben a billentyűzet), a szabványos kimenet (alapértelmezésben a programot hívó karakteres felület) és a szabványos hibakimenet (alapértelmezésben azonos a szabványos kimenettel) meghatározására szolgál. A Python (és gyakorlatilag minden más nyelvű program) a külvilággal állományokon keresztül kommunikál. A legfőbb ilyen állomány ez a három. Például az **input()** függvény a szabványos bemenetről olvassa be a visszaadandó szöveget, a **print()** a szabványos kimenetre írja a kiírandókat, ha az értelmező hibát észlel, akkor a hibaüzenetet a szabványos hibakimenetre írja. Ha például átállítjuk a szabványos kimenetet egy szöveges állományra, akkor a **print()** a képernyő helyett ebbe az állományba fog írni. Másik alkalmazás, amikor bizonyos függvények paraméterben kérik a bemenet vagy a kimenet helyének megadását. Itt alapból a szabványos ki/bemenetet lehet a fenti változókkal megadni.

sys.__stidn__, sys.__stout__, sys.__stderr__

Az előző három változó azon változatai ezek, ahol a változó neve előtt és után két-két aláhúzás jel található. Ezek csak olvasható változók és a program indításakori értékeket tartalmazzák.

sys.version

Ebben a változóban egy karakterlánc található, amely a Python értelmező verzióját azonosítja. Amennyiben az értelmezőt interaktív módban indítjuk, az értelmező kiírja ennek értékét a prompt előtt.

sys.version_info

Az előző helyett ezt a változót célszerű használni, ha az adott értelmező verziószámára van szükség, mivel ez ad biztosabb információt. Ez viszont több érték felsorolása (tuple) formájában adja az információt.

Végezetül érdemes még megjegyezni, hogy a Python értelmező konfigurációjával kapcsolatos dolgokra van egy külön modul *sysconfig* néven. Ott részletesebb információkhoz lehet jutni.

10. Állományműveletek

A következőkben azok a műveletek szerepelnek, amelyek az **open()** függvény által visszaadott állomány-objektumon hajthatóak végre. Az *io* modul további állománytípusokhoz ad hozzáférést, amelyeken nagyjából ugyanezek a műveletek végezhetőek el. Itt csak a szöveges állományokon végezhető műveletek szerepelnek.

file.close()

Bezárja a *file* állományt. Szükség esetén előzőleg gondoskodik a még nem kiírt tartalom kiírásáról (*flush*) is. Az állomány bezárása előtt nem lehetünk biztosak abban, hogy a kiírni szánt tartalom ténylegesen belekült-e az állományba, így minden állomány megnyitásánál gondoskodni kell annak valamikori bezárásáról!

Bezárt állományon a műveletnek nincs hatása. A további írási/olvasási műveletek azonban a *ValueError* hibát eredményezik.

file.closed

Egy változó, amely igaz értéket ad, ha a *file* állomány be van zárva.

file.fileno()

Egy egész számot ad vissza, amely az *állomány leírója*, vagy ha ilyen nincs, akkor az *OSError* hibát eredményezi. Az *állományleíró* az operációs rendszer által használt azonosító a megnyitott állományok nyilvántartására.

file.flush()

Az *írási puffer*ben maradt tartalmat fizikailag is az állományba írja. Olvasásra megnyitott állományokra nincs hatása. Ezzel a művelettel – vagy az ezt a műveletet is végrehajtó **.close()** művelettel – lehet gondoskodni arról, hogy a kiírni szánt tartalom ténylegesen is kiírásra kerüljön.

file.isatty()

Igaz értéket ad, ha a *file* egy interaktív állomány, mint amilyen például a szabványos bemenet.

file.readable()

Igaz értéket ad azokra az állományokra, amelyek úgy vannak megnyitva, hogy lehessen a tartalmukat olvasni. Ha értéke hamis, akkor a **.read()** művelet *OSError* hibát eredményez.

file.read(size=-1)

A paraméterében megadott számú byte-ot, karaktert beolvassa a *file* állományból és visszaadja azt (szöveges állomány esetén *str* típusú értékként). Amennyiben nem szerepel a paraméter, vagy -1 értékű, akkor az olvasás az állomány végég történik. Amennyiben nulla byte beolvasása történt, de a paraméter értéke nem 0, az azt jelzi, hogy az állomány végére ért az aktuális pozíció.

file.readline(size=-1)

Beolvas egy sornyi szöveget a *file* állományból és visszaadja azt. Amennyiben a *size* paraméter értéke pozitív, akkor legfeljebb ennyi karakter beolvasásra kerül. Szöveges állományok esetében elvileg az **open()** függvény *newline* paraméterével felül lehet írni az alapértelmezett ’\n’ értéket, mint újsor karaktert. Ha az olvasási kísérlet az állomány végén történik, akkor az üres szöveget adja eredményül.

file.readlines(hint=-1)

Beolvassa az állomány tartalmát és soronként egy listaelembe téve kapot listaként visszaadja azt. A *hint* paraméterrel lehet jelezni, hogy hány sorra kell számítani: legfeljebb annyi sor kerül beolvasásra, amennyi ennyi karaktert tartalmaz (tehát a maximum beolvasható karakterek számát lehet vele megadni, nem a maximum beolvasandó sorokét).

file.seek(offset, whence)

Az állományon belüli aktuális pozíciót állítja át a megadott értékre. A következő írási/olvasási művelet az új pozícióról fog történni. A *whence* paraméterrel adható meg, hogy honnan kell számolni az *offset* értéket. Ha a *whence* értéke 0, akkor az állomány elejétől kell az *offset* értéket számolni, ez az alapértelmezés. Ha a *whence* értéke 1, akkor az aktuális pozíciótól mérendő az *offset*. Ha *whence* értéke 2, akkor az állomány végétől kell az *offset* értéket mérni, amely ekkor jellemzően egy negatív egész számot tartalmaz.

Az *offset* értéket szöveges állományok esetében nem byte értékként, hanem karakter értékként kell érteni, ami egyes karakterkódolásoknál – így az alapértelmezett utf-8 kódolás esetén is – lehet több byte méretű karakter is. Ennek megfelelően csak 0, vagy a **.tell()** művelet által visszaadott érték használható, minden más érték esetén a metódus viselkedése definiálatlan. Ebből következően a második paraméter 1 és 2 értéke esetén a nullától különböző első paraméter nincs támogatva.

file.seekable()

Igaz értéket ad vissza, ha az állomány támogatja a közvetlen hozzáférést (*random access*), azaz működik rá a **.seek()** művelet. Ha a visszaadott érték hamis, akkor erre az állományra kiadott **.seek()**, **.tell()** és **.truncate()** műveletek *OSError* hibát okoznak.

(Tipikusan az interaktív állományok nem támogatják a közvetlen hozzáférést, mert csak sorban érkeznek róla az adatok, vagy sorban írhatóak ki bele.)

file.tell()

Visszaadja az állomány aktuális pozícióját. Szöveges állományok esetében a visszaadott érték nem feltétlenül byte-okban értendő!

file.truncate(size=None)

Átméretezi az állományt a paraméterben megadott méretre, vagy ha az nincs megadva, akkor az aktuális pozícióra. Az aktuális pozíció nem változik a művelettől. Ez a művelet csökkentheti is és növelheti is az állomány méretét. Utóbbi esetben az újonnan létrejövő tartalom az operációs rendszertől függ, a legtöbb esetben nulla értékű byte-okkal történik a feltöltés. A metódus az új állományméretet adja vissza.

file.writeable()

Igaz értéket ad vissza, ha a *file* egy írásra (is) megnyitott állomány. Hamis értéke esetén az állományra történő **.write()** és **.truncate()** művelet *OSError* hibát eredményez.

file.write(t)

A *t* paraméter tartalmát kiírja a *file* állományba. Szöveges állományok esetén a paraméternek *str* típusúnak kell lennie. A visszaadott érték a ténylegesen kiírt karakterek száma lesz.

file.writelines(lines)

Kiírja a *file* állományba a *lines* paraméterben levő lista elemeit, mint sorokat. A paraméternek egy olyan listának kell lennie, amelynek elemei mind a kiírandó típusnak megfelelően – szöveges állo-

mány esetén *str* típusúak. Mivel a sorok közé sortörés kiírása nem történik, a paraméterben levő lista elemeinek végén el kell az újsor karaktert helyezni.

11. A json modul

Ez Python modul a JSON formátum használatát teszi lehetővé Python programokban. A JSON a JavaScript Object Notation rövidítése, ami arra utal, hogy eredetileg a JavaScript nyelvben használatos adattípusok szöveges megjelenítésére találták ki.

A JSON nyelvfüggetlenül képes a legtöbb programozási nyelvben létező adattípus megjelenítésére. C, C++, C#, Java, JavaScript, Perl, Python és sok egyéb nyelv esetén létezik a használatához szükséges programkönyvtár valamilyen formában. Python esetében ez a programkönyvtár a *json* modul.

*json.dump(obj, fp, *, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw)*

Szerializál egy *obj* objektumot JSON formátumú szöveggé és azt kiírja az *fp* által megadott, írásra megnyitott **szöveges** állományba (a **file.write()** műveletet használva).[7]

Amennyiben a *skipkeys* paraméter értéke igaz (az alapértelmezés a hamis), azok a szótárkulcsok, amelyek típusa nem valamelyik alaptípus (*str*, *int*, *float*, *bool*, None), kihagyásra kerülnek. Hamis érték esetén ezen esetben a *TypeError* hiba keletkezik.

Az *ensure_ascii* paraméter igaz értéke esetén a kimenet biztosan csak olyan karaktereket fog tartalmazni, amelyek az ASCII kódtáblában szerepelnek, a nem-ascii karaktereket azok helyettesítő karaktersorozatával (*escape-sequence*) fogja a keletkező szöveg tartalmazni (ezek a \ karakterrel kezdődő sorozatok).

A *check_circular* paraméterrel lehet az adatban levő körkörös tartalmazás ellenőrzését szabályozni. Az alapértelmezett igaz érték esetén az összetett adattípusokban levő tartalomra megtörténik annak ellenőrzése, hogy nincs-e benne olyan komponens, amely rekurzívan tartalmazza az objektum egy részét. Ezt az ellenőrzést lehet elnyomni a paraméter hamis értékével, azonban ekkor *RecursionError* hiba keletkezik ilyen esetben. Akkor van értelme a hamis értéknek, ha biztosan ilyen önmagát tartalmazó rész az objektumban és gyorsítani szeretnénk a programot az ellenőrzés kihagyásával.

Az *allow_nan* paraméter segítségével szabályozható, hogy mi történjen a *float* típus speciális értékeire, mint a `nan`, `inf` és `-inf`. Alapértelmezett igaz érték esetén ezen értékek JavaScriptbeli megfelelőjére konvertálódnak: NaN, Infinity és -Infinity. Hamis érték esetén ezek az értékek a *ValueError* hibát váltják ki.

Amennyiben az *indent* paraméter egy nemnegatív egész számot kap értékül vagy egy nemüres sztringet, a JSON kifejezés olvashatóbb formázást kap, amelyben a paraméterben megadott bentebb kezdési értékek fognak szerepelni. Ha a paraméter értéke 0 vagy az üres szöveg (`""`), akkor a formázás csak új sorokat szúr be, de bentebb kezdést nem.

Amennyiben a *separators* paraméter szerepel, akkor értéke egy két elemű (*item_separator*, *key_separator*) *tuple* kell legyen. Alapértelmezés a hivatalos JSON elválasztókat eredményezi: `(', ', ': ')`, ezt lehet felülírni – amennyiben az eredményt az adatok reprezentálására használjuk csak, akkor érdemes ezzel a lehetőséggel érni, egyébként a kiírt adatok felhasználhatóságát nehezíti az elválasztók megváltoztatása.

7 A modul csak az str típust támogatja, így az állománynak szöveges állománynak kell lennie.

A *default* paramétert akkor kell használni, ha az objektum olyan típust tartalmaz, amelynek szerializálására nincs automatikus megoldás. Ekkor a szerializálást végző függvény nevét megadva, az alapértelmezett szerializáló eljárást lehet kicserélni olyanra, amely a problémás objektummal is elbír.

A *sort_keys* paraméter igaz értékével el lehet érni, hogy a szótárak tartalma úgy kerüljön kiírásra, hogy annak kulcsai rendezett sorrendben helyezkednek el.

A szabályos JSON tartalmú állomány csak egy objektum szerializált változatát tartalmazhatja, így a dump() függvényt nem szabad többször meghívni ugyanazzal a célállománnyal!

*json.dumps(obj, *, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw)*

A **dump()** függvény olyan változata, amely nem állományba írja az eredményül kapott *str* szöveget, hanem visszaadja azt. Ezért hiányzik is az *fp* paraméter a fenti specifikációból, míg a többi paramétere a dump() függvényével teljesen azonos módon használandó.

*json.load(fp, *, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw)*

Az *fp* egy olvasásra megnyitott szöveges állomány, amelynek tartalmát JSON formátumú szövegnek tekintve azt deszerializálja a függvény, azaz előállítja a neki megfelelő Python adatszerkezetet.

Az *object_hook* paraméterrel lehet saját dekódoló függvényt alkalmazni az alapértelmezett helyett a szótár típusú adatok feldolgozásához.

Az *object_pair_hook* paraméter is egy függvényt vár, de a lista típusú adatok feldolgozásához.

A *parse_float* paraméternél megadott függvénnyel lehet a lebegőpontos értékek dekódolására szolgáló saját függvényt megadni, hasonlóan, a *parse_int* paraméterrel lehet az egész értékek dekódolására szolgáló függvényt sajátra cserélni. A parse_constants paraméternél megadott függvény az `'-Infinity'`, az `'Infinity'` és a `'NaN'` szövegek feldolgozására szolgál.

Amennyiben a függvény által deszerializálandó szöveg nem helyes JSON szöveg, akkor a *JSONDecodeError* hiba keletkezik.

*json.loads(s, *, cls=None, object_hook=None...)*

A függvény ugyanazt csinálja, mint a **load()** függvény, azzal a különbséggel, hogy az első paraméterében nem egy állományobjektum, hanem a deszerializálandó JSON szöveget tartalmazó *str* típusú érték szerepel. Azaz egy JSON szöveget lehet vele a megfelelő Python adatszerkezetté alakítani.

Weboldalak készítése (HTML + CSS)

A következő fejezetek sorban felsorolják a weboldalak készítése során használt HTML elemeket, azok lehetséges paramétereit, valamint a formázásukhoz használható tulajdonságokat, amelyek értékét a CSS nyelv segítségével lehet módosítani.

12. HTML elemek

HTML

Ez az elem maga a HTML dokumentum. Ennek tartalma két további elem: a HEAD és a BODY. Minden, ami a weboldalhoz tartozik, ebben a két elemben helyezendő el.

Ajánlott mindig megadni a *lang* paraméteren keresztül a dokumentum nyelvét.

Példa az elem használatára (a HTML szabványból másolva):

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Swapping Songs</title>
</head>
<body>
<h1>Swapping Songs</h1>
<p>Tonight I swapped some of the songs I wrote with some friends, who
gave me some of the songs they wrote. I love sharing my music.</p>
</body>
</html>
```

12.2. A dokumentum metaadatai

HEAD

A HEAD elem belsejében adjuk meg a dokumentum megjelenítését végző programnak (azaz a böngészőnek) a megjelenítést segítő metaadatokat. Ezek a TITLE elem tartalmát kivéve nem jelennek meg a weboldalon, de befolyásolhatják annak megjelenítését.

Az általános paraméterekkel rendelkezik.

TITLE

A TITLE elem a HEAD elemen belül helyezendő el. A dokumentum címét határozza meg a tartalma. Alapvetően kötelező elemnek számít, az egyetlen kivétel, ahol nem szerepel, amikor a címet valami meghatározza magasabb szinten, például amikor a HTML dokumentum egy e-mail szövegeként szerepel. Csak egy TITLE elem lehet a dokumentumban.

Az általános paraméterekkel rendelkezik.

BASE

A BASE elem lehetővé teszi a dokumentum alap-URL címének megadását. Amennyiben az elem nem szerepel, akkor a böngésző az aktuális dokumentum címéből állítja ezt elő, ezt lehet az elemmel felülbírálni. Akkor van ennek értelme, ha a dokumentum sok olyan hivatkozást tartalmaz, amely egy adott, másik alapcímmel rendelkezik. Legfeljebb egy BASE elemet szabad a HEAD belsejében alkalmazni, amelynek vagy a *href*, vagy a *target* paramétert (vagy mindkettőt) tartalmaznia kell.

Paraméterei:

- Az általános paraméterek
- `href`: Egy valós URL-t kell tartalmaznia, amely a dokumentumban levő hivatkozások alap URL címe lesz.
- `target`: Meghatározza azt a területet, ahol a hivatkozások alapértelmezésben megnyílnak. (lásd részletesen a paraméter leírásánál).

LINK

A LINK elem a HEAD elemen belül szerepelve lehetővé teszi további erőforrásoknak a HTML dokumentumhoz kapcsolását. Elsősorban külső stíluslap megadására szokás használni, de más jellegű erőforrásokat is meg lehet vele adni. Az elemnek csak nyitótagja van.

Az általános paramétereken felül az alábbi paraméterei lehetnek:

- `href`: Az erőforrás címét tartalmazza (ennélfogva az elemnek nincs jelentése enélkül a paraméter nélkül).
- `crossorigin`
- `rel`: Az aktuális dokumentum és a külső erőforrás közötti kapcsolat jellegét írja le, szintén feltétlenül szükséges paraméter.
- `media`: A dokumentumban használandó média megadása.
- `integrity`
- `hreflang`: a hivatkozott erőforrás nyelve.
- `type`: segítség a megjelenítő eszköznek az erőforrás típusára
- `referrefpolicy`
- `sizes`: Amennyiben a *rel* paraméter értéke „icon”, akkor annak méretét adja meg.
- `imagesrcset`: Különböző helyzetekben használandó képek adhatóak meg az egyes megjelenítő-fajták számára (pl. `rel="preload"` esetére).
- `imagesizes`: A különböző megjelenítési esetekre vonatkozó fenti paraméterben megadott képek mérete.
- `as`: potenciális célpont egy előtöltési kéréshez (pl. `rel="preload"` vagy `rel="modulepreload"` esetére)
- `blocking`: Az elem esetleg blokkolhatja a dokumentum megjelenítését?
- color: Az oldal-ikon testreszabásához használandó szín (`rel="mask-icon"` esetén)
- `disabled`: Megadása esetén az erőforrás nem kerül alkalmazásra (Javascript-en keresztül ez a tulajdonság megváltoztatható, így van értelme az erőforrást ilyen esetben is megadni).
- `title`: Ez a paraméter a LINK elemnél speciális jelentéssel bír. Nem egy buborékszöveget lehet vele megadni, hanem a hivatkozott stíluslapot lehet elnevezni.

Az elem nagyon sokféle erőforrás megadására alkalmas, amit itt nincs értelme részletezni. Külső stíluslap megadásához a következőképpen kell használni:

- *href* paraméternek a stíluslapot tartalmazó állomány címét kell megadni.
- A *rel* paraméter értéke: „`stylesheet`” legyen.
- A *type* paraméter értéke „`text/css`” legyen.

META

Ez az elem különböző metaadatokat tud megadni.

Paraméterei:

- Az alapértelmezett paraméterek

- `name`: a metaadat neve;
- `http-equiv`: HTTP paraméter érték;
- `content`: az elem értéke
- `charset`: a dokumentum karakterkódolásának megadása
- `media`: a médiatípus, amelyre a dokumentum alkalmazható.

Az elem a *name* paraméterrel képes dokumentum-szintű metaadatot megjeleníteni, vagy a *http-equiv* paraméterrel olyan metaadatokat felülírni, amelyeket a webszerver küld a HTTP(S) protokoll fejlécében. Ezen felül a dokumentum karakterkódolását is meg lehet adni a *charset* paraméterhasználatával.

Az elemnek csak nyitótagja van, amelyben pontosan egy *name*, *http-equiv*, *charset* vagy – fent nem felsorolt – *itemprop* paraméternek kell lennie. Amennyiben a name vagy a http-equiv paraméter szerepel, akkor a metaadat értékét a *content* paraméteren keresztül kell megadni, ellenkező esetben ez a paraméter nem szerepelhet.

Egy dokumentumban egy META elem szerepelhet a *charset* paraméterrel, amely így megadja a dokumentumban használt szöveg nyelvét.

A name paraméterben használt nevek a következők lehetnek:

- `application-name`: Az értéknek egy rövid szövegnek kell lennie, amely megadja azt a webes alkalmazást, amelyet az oldal reprezentál.
- `author`: az oldal szerzője (szöveg)
- `description`: az érték az oldalt leíró szöveg.
- generator: ennek értékét általában beállítja az a program, amellyel az oldal előállítása történt. Szöveges adat.
- `keywords`: Valaha jelentéssel bíró metaadat, amellyel sokan meghekkelték az internetes keresőket, így ma már nincs hatása a felsorolt értékeknek.
- `referrer`: a megadott kulcsszó meghatározza a hivatkozói szabályzatot, amely az oldalra érvényes (ezzel részletesebben nem érdemes foglalkoznunk).
- `theme-color`: Az értéknek olyan szövegnek kell lennie, amely a CSS színmegadási szabályának megfelel. Ezt a színt javasolja a dokumentum a megjelenítőnek a tartalom megjelenítéséhez.
- `color-scheme`: Hasonlóan az előzőhöz, az oldal javasolt háttérszínét lehet vele megadni.

A fentieken kívül más neveket is lehet alkalmazni a name paraméteren keresztül.

A http-equiv a következő értékeket kaphatja (csak ezeket):

- `content-language`: A dokumentum alapértelmezett nyelve. Ajánlatosabb helyette a *lang* paramétert használni a HTML elemre.
- `content-type`: A dokumentum szövegénél használt karakterkódolás mellett a dokumentumot leíró nyelv specifikációja adható itt meg. A *content* értéke: „`text/html; charset=`…”, ahol a … helyére a dokumentum szövegkénél használt karakterkódolást kell megadni. A régebbi HTML verziók alkalmazták ezt a bonyolult megadást, helyette a *charset* paramétert érdemes inkább használni.
- `default-style`: A dokumentum alapértelmezett stíluslapkészletét lehet vele megadni. Amennyiben több külső stíluslap is meg van adva, akkor az azokat megadó LINK elemekben adjunk nekik egyedi neveket, majd itt a *content* paraméter értékében szerepeljen az a név, amelyhez tartozó stíluslapot alapértelmezésnek kell tekinteni – a többi alternatív stíluslap lesz, amit a megjelenítő eszközben lehet valahogy beállítani.

- `refresh`: Egy időzített átirányítást lehet vele megadni. Például akkor szokták alkalmazni, ha az oldal máshova költözött és a régi helyen egy szöveggel adnak erről tájékoztatót, majd az oldal (ezzel a metaadattal) egy idő elteltével az új oldalt tölti be maga helyett.
- `set-cookie`: Ennek a lehetőségnek nincs hatása, a böngészőknek figyelmen kívül kell hagyniuk (nyilván a régi HTML verziók használták).
- `x-ua-compatible`: Arra szolgál, hogy az Internet Explorer nevű csodát rávegye a szabvány szorosabb követésére.
- `content-security-policy`: Az oldalra érvényes biztonsági beállításokra vonatkozhat ez az adat…

STYLE

Ezzel az elemmel a HEAD elemen belül belső stíluslapot lehet megadni. Az elem tartalma szintaktikailag megegyezik a külső stíluslapot tartalmazó állomány szintaxisával.

Az alapértelmezett paramétereken felül a következő paraméterei lehetnek:

- `media`: azt a megjelenítési módot lehet vele megadni, amire az itt megadott stíluslapot alkalmazni kell.
- `blocking`: ezzel lehet jelezni a megjelenítő eszköznek, hogy lehet a stíluslapon olyasmi, amit a megjelenítéssel meg kell várni (blokkolhatja a megjelenítést).
- `title`: A hagyományos jelentéssel ellentétben ez a paraméter a stíluslap elnevezésére szolgál (mint a LINK elem esetében is).
- `disabled`: Ez az alapértelmezett paraméter tudja megadni a megjelenítő eszköznek, hogy ne vegye figyelembe a stíluslapot az oldal megjelenítésekor. Mivel interaktív tartalom (pl. Javascript program) képes a paramétert állítani, így utólag bekapcsolható a stíluslap használata.

Az elem type paraméterében szokás megadni a stíluslap nyelvét, ami – mivel nincs más ilyen nyelv – kötelezően „`text/css`”.

12.3. Szakaszok kialakítása

BODY

A BODY elem tartalmazza az összes olyan elemet, amelyet a weboldalon meg kell jeleníteni. Az elem mindig a HTML elem belsejében, a HEAD elem után szerepel.

A BODY elem adja meg a dokumentum tartalmát.

ARTICLE

Az ARTICLE elem egy teljes, szöveget ad meg a dokumentumban. Ez lehet egy fórum bejegyzés, egy cikk, blogbejegyzés, egy hozzászólás vagy más hasonló.

Az ARTICLE elemek egymásba is ágyazhatóak. Ekkor a belső ARTICLE egy olyan tartalmat ad meg, amely valamilyen formában a külső ARTICLE tartalmához kapcsolódik – például a külsővel megadott cikkhez tartozó hozzászólásként.

Az ADDRESS elemmel hozzáadott szerzői információk nem tartoznak az esetleges beágyazott további ARTICLE elemhez.

Amennyiben az egész dokumentum egyetlen cikknek felel meg, akkor az ebben elhelyezett ARTICLE redundánsnak számít, hiszen maga a BODY ilyenkor megfelel az ARTICLE elemnek.

SECTION

A SECTION elem egy dokumentum valamely fejezetét, szakaszát reprezentálja.

NAV

Ez az elem egy olyan területét reprezentálja az oldalnak, amelyen a dokumentum más oldalára mutató hivatkozások találhatóak: navigációs területet képeznek.

ASIDE

Ez az elem az oldal olyan részét reprezentálja, amely az elemet körülvevő tartalomhoz kapcsolódik, de attól különállóként kezelendő. Az ilyen területet nyomtatott dokumentumokban általában oldalsó sávban szokták megjeleníteni.

Az elem jól használható olyan tipográfiai megoldásokhoz, mint az oldalsó sávbeli megjegyzés, navigációs elemek vagy egyéb, a fő tartalomtól elkülönítve oldalt megjelenítendő tartalom.

H1, H2, H3, H4, H5, H6

Ezek az elemek a dokumentum egyes szakaszainak a címeit adját meg.

Az elemeknek van egy-egy rangjuk a címek hierarchiájában. Minden szakaszban a H1 a legmagasabb rangú elem, azután a H2 következik,… A hiearchiát be kell tartani, azaz nem szabad például a H2 elem után a H4 elemet szerepeltetni. Az egymásba ágyazott szakaszok (SECTION elem) a címsor rangját csökkentik, így a fejezetet megadó SECTION H1 eleménél egyel alacsonyabb rangú az ebben elhelyezkedő alfejezet SECTION elemében levő H1 elem stb.

HGROUP

Ez az elem egy címsorcsoportot reprezentál. A H1–H6 elemeket lehet vele csoportosítani, hogy azokkal alcímet lehessen megadni.

HEADER

Az elem egy bevezető vagy navigációs csoportot reprezentál az oldal elején. Alapvetően H1–H6 elemek vagy HGROUP elem szokott lenni a tartalma, de ez nem kötelező. Szokás még egy szakasz tartalomjegyzékének megadására is használni.

Ez az elem nem hoz létre a dokumentumban új szakaszt!

FOOTER

Az elem a dokumentum a hozzá legközelebbi szakasz befejezését adhatja meg. Tipikusan a szakaszhoz kapcsolódó információkat szoktak benne elhelyezni, mint például hogy ki írta a szakaszt, kapcsolódó dokumentumokra mutató hivatkozások, szerzői jog stb.

Amennyiben a FOOTER elem egész szakaszokat tartalmaz, akkor tekinthető mellékletnek, kolofonnak is.

ADDRESS

Ez az elem tartalmazza azokat az információkat a hozzá legközelebbi ARTICLE vagy BODY elemhez, amelyen keresztül annak szerzőét el lehet érni.

12.4. Blokkot létrehozó elemek

P

A P elem egy bekezdést reprezentál. Elsősorban szöveg sorokba tördelve történő megjelenítésére használatos, de vannak olyan elemek, amelyek alapvetően a szövegtördelést tekintve karakterként viselkednek, így szintén bekezdésben helyezendőek el.

Az elemnek mind a nyitó-, mind a zárótagja elhagyható, ha ettől a bekezdés kezdete és vége felismerhető. Ennek ellenére nem jó ötlet a névtelen bekezdések alkalmazása, mivel ezekre nem lehet hivatkozni, így például a formázásukra sincs a stíluslapon keresztül lehetőség.

HR

Ez az elem egy tematikai váltást jelez, például a történetben egy jelenetváltást. A HTML 4-es szabványa szerint még ez egy vízszintes vonalat hozott létre, de a HTML5 szerint a HR elem már nem feltétlenül eredményz vízszintes vonalat, helyette új, szemantikai jelentést kapott az elem!

PRE

Ez az elem egy előreformázott szöveget határoz meg, amelynek felépítését nem elemekkel, hanem a szöveg tipográfiai kialakításával határozzuk meg. Az elem nyitótagját közvetlenül követő újsor karaktert a böngésző lenyeli, de minden más karaktert, legyen az szóköz, újsor, tabulátor is, a megadott módon kell megjeleníteni.

BLOCKQUOTE

Az elem olyan szövegrészt reprezentál, amelyet más forrásból idéztek. Az idézett tartalomra való hivatkozást a *cite* paraméter segítségével lehet megadni.

Amennyiben a *cite* paraméter szerepel, annak értéke kötelezően egy URI, amely az idézett tartalom forrására mutat.

OL

Ez az elem egy számozott listát tartalmaz. A lista elemei az LI elemmel adandóak meg. Az elem nyitó- és zárótagját is kötelező megadni.

Amennyiben a *reversed* paraméter szerepel (érték nélkül, mivel ez egy logikai típusú paraméter), akkor a számozás visszafelé történik úgy, hogy a lista utolsó eleme kapja az 1 értéket. Ha a paraméter nincs megadva, akkor a számozás 1-től indul és növekszik.

A *start* paraméterrel lehet felülírni az 1 kezdőértéket. Értékének egész számként értelmezhetőnek kell lennie.

A számozáshoz használt értékek megjelenési módját a *list-style-type* CSS tulajdonsággal lehet meghatározni (alapértelmezés: arab számok).

UL

Ez elem egy felsorolást (nem számozott listát) tartalmaz. A lista elemei az LI elemmel adandóak meg. Az elem nyitó- és zárótagját is kötelező megadni.

A lista egyes elemei előtt ugyanaz a listajel jelenik meg, amelyet a *list-style-type* CSS tulajdonsággal lehet meghatározni. Egymásba ágyazott listák esetén az első három szintnek van külön-külön listajele, ezért ennél több szintű listánál mindig meg kell határozni az alkalmazandó listajelet.

MENU

Ez az elem egy olyan eszköztárt reprezentál, amelynek tartalma nem számozott lista elemeiként megadva egy menü menüelemeit adják. Az egyes menüelemeket tehát az LI elemmel lehet megadni.

(Ez az elem tehát egyszerűen az UL elem szemantikai alternatíváját jelenti. A kinézet menüszerűvé alakításáról külön kell gondoskodni.)

LI

OL, UL vagy MENU elemen belül annak elemeit adja meg az LI elem. Számozott lista esetén az elem számértékét az elem *value* paraméterével lehet megadni. Értékének egész számként értelmezhetőnek kell lennie.

DL

Ez elem tartalma definíciós lista. A szócikk megadására a DT, a meghatározás megadására a DD elem szolgál: A DL elemnek nulla vagy több név-érték csoportot kell tartalmaznia. Egy ilyen csoport áll egy vagy több DT elemből a szócikk(ek) megadására, amit egy vagy több DT elem követ az ahhoz tartozó meghatározás(ok) megadására.

DT

Az elem a DL lista egy meghatározását tartalmazza.

DD

Az elem a DL lista egy címszavát tartalmazza.

FIGURE

Ez az elem egy ábrát hatátor meg, opcionális ábraszöveggel. Az ábra maga nem kötelezően két, bármely önmagában egységet alkotó dokumentumrész lehet, amelyre az aláírásán keresztül lehet utalni a dokumentum más részében levő szövegben.

Legjellemzőbb felhasználása a kép és a képaláírás összefogása egy objektummá.

FIGCAPTION

Képaláírás megadása egy FIGURE elemben. A tartalma lehet több bekezdésnyi is.

MAIN

Az elem a dokumentum főrészét reprezentálja. Ebből következően egy dokumentumban legfeljebb egy szerepelhet belőle.

DIV

Ennek az elemnek alapvetően semmilyen jelentése nincs. A benne található további elemek összefogására alkalmas. A CSS tulajdonságain keresztül lehet a megjelenését szabályozni, így például szövegdobozként is felhasználható.

12.5. Szövegelemek

A

Az A elem egy link valamelyik végpontját adja meg.

Amennyiben van *href* paramétere, akkor az elem egy hivatkozást ad meg, amely a *href* paraméter értékeként megadott URI-ra mutat. (*ugrópont*)

Amennyiben nincs *href* paramétere, akkor az elem egy célpontja lehet egy másik helyről induló linknek. Ilyenkor valamilyen módon – általában a *name* paraméterrel – beazonosíthatóvá kell tenni az elemet, hogy az az URI-ban megadható legyen.

Amennyiben a *name* paraméter szerepel, akkor annak értéke egy előírt # jellel adja meg azt a címet, amellyel az adott célpont elérhető. A *name* paraméter értéke egyedi azonosító, amely nem szerepelhet a dokumentumban egyetlen másik *name* vagy *id* paraméter értékeként sem, kivéve ugyanannak az A elemnek az *id* paraméterét.

EM

Ez az elem egy szövegbeli kiemelést ad meg. Az elemet önmagával lehet egymásba ágyazni, amelylyel a kiemelés erősségét lehet megadni.

STRONG

Ez az elem egy szöveg fontosságát adja meg. Az elem használható a címekben, képaláírásokban valami fontosságának jelzésére. Használható az elem figyelmeztetés megadására vagy sürgősség jelzésére.

Az EM elemhez hasonlóan, több STRONG elemet egymásba ágyazva a fontosságát lehet növelni a szövegnek.

SMALL

Az elem a szöveget kisebb megjegyzésként jelöli meg.

Nem szabad hosszabb szövegre – például több bekezdésnyi szövegre – alkalmazni.

S

Az elem a tartalmát olyannak jelöli meg, amely többé már nem pontos vagy nem releváns.

CITE

Az elem egy idézett tartalom címét jelöli meg.

Q

Szövegen belüli idézet megadására szolgáló elem. Az elem automatikusan elhelyezi az idézet elé és után a dokumentum nyelve szerint használandó idézőjelekt. Amennyiben a Q elem egy másik Q elembe kerül, akkor a nyelvtani szabályok szerinti belső idézőjelekt fogja használni.

A BLOCKQUTE elemhez hasonlóan a *cite* paraméterrel megadható az idézet forrását adó URI.

DFN

Ez az elem a tartalmát egy fogalom definiálásaként jelöli meg.

ABBR

Az elem a tartalmát, mint rövidítést vagy mozaikszót jelöli meg. A *title* paraméterben megadva a rövidítés kifejtését, azt a böngésző egy buborékban meg tudja jeleníteni.

TIME

Ebben az elemben a gép által olvasható formátumú időadat adható meg. A HTML szabvány időnek tekinti a dátumadatot is, azaz az elemben akár dátum, akár időadat, akár egyszerre mindkettő szerepelhet.

CODE

Ez az elem a tartalmát számítógépes kódrészletnek jelöli meg.

VAR

Ez az elem a tartalmát matematikai vagy informatika változónak jelöli meg.

SAMP

Ez az elem a tartalmát egy számítógépes program által adott kimenetként jelöli meg.

KBD

Ez az elem a tartalmát egy számítógépes program számára szánt bemenetként – begépelendő tartalomként – jelöli meg.

SUB és SUP

Alsó- és felsőindex megadására szolgálnak ezek az elemek.

I

Az elem a tartalmát olyan szövegnek jelöli meg, amely a normál szövegtől eltérőnek számít valamilyen okból, ezért általában dőlt betűvel írnánk.

B

Ez az elem a tartalmát olyan szövegnek jelöli meg, amelyet általában félkövér betűkkel szokás írni.

U

Ez az elem a tartalmát olyan szövegnek jelöli, amelyet aláhúzással kellene kiemelni a környezetéből.

Az aláhúzást weboldalon nem célszerű használni, mert a hivatkozásokat szokás aláhúzással jelezni.

MARK

Az elem a tartalmát olyan szövegnek jelöli, amelyet ki kell emelni. Például idézetben szokás jelezni olyan kiemelést vele, amely nem az eredeti szövegben volt kiemelve, hanem az idézetben kapott kiemelést.

SPAN

Alapvető jelentése nincs az elemnek. A DIV szöveg belsejében alkalmazható párja: a tényleges jelentését a formázással vagy egy JavaScript kódrészen keresztül nyeri el.

BR

A bekezdés megtartásával új sor kezdését írja elő az elem a szövegben.

13. HTML elemek paraméterei

13.1. Általános paraméterek

Az itt felsorolt paraméterek gyakorlatilag minden HTML elem esetén szerepelhetnek. Ezek nem mindegyik szerepel itt részletesen, azért először íme egy teljes lista:

- accesskey
- autocapitalize
- autofocus
- contenteditable
- dir
- draggable
- enterkeyhint
- hidden
- inert
- inputmode
- is
- itemid
- itemprop
- itemref
- itemscope
- itemtype
- lang
- nonce
- spellcheck
- style
- tabindex
- title
- translate

Lang

A *lang* paraméter megadja az adott elem tartalmának nyelvét. Célszerű a HTML elemre beállítani a dokumentum nyelvét, hogy a dokumentumot megjelenítő/felolvasó program helyesen tudja a benne levő szöveget kezelni. Ezen felül ha a dokumentumban valahol más nyelvű tartalom fordul elő, akkor ott kell ezzel a paraméterrel megadni annak nyelvét. A paraméter értéke többek között befolyásolja az idézetek körül megjelenő idézőjelek megjelenését, így például a Q és a BLOCKQUOTE elemnél mindenképpen adjuk meg, ha a környezetétől eltérő nyelvű az idézett szöveg!

13.2. Hivatkozások paraméterei

href

Ez a paraméter mindig egy URI értéket tartalmaz. A elemnél a hivatkozás céljának URI-ja, BASE elem esetén a dokumentumban levő relatív című hivatkozások alap címét adja meg. LINK elem esetén annak az erőforrásnak a címét, amelyet a LINK elem leír.

BASE elem esetén mindig abszolút címet kell megadni, hiszen ez lesz az alapja a dokumentumban levő relatív címeknek. Az A és a LINK elem relatív értékei alapesetben az aktuális dokumentumhoz relatívak, ha a BASE elem szerepel a *href* paraméterrel, akkor ezt módosítja az itt megadott címre.

target

Ez a paraméter a megnyitandó hivatkozás célterületét adja meg. A és BASE elemnél használatos. Utóbbinál megadva az A elem alapértelmezett viselkedését bírálja felül.

Értéke lehet

- egy konkrét név, amely esetben ha olyan nevű ablak van, akkor abban nyílik meg a hivatkozott oldal, különben azzal a névvel létrejövő új ablakban;
- `_BLANK` érték, amely esetén egy új, névtelen ablak jön létre és abban nyílik meg a hivatkozott tartalom;
- `_TOP` érték, amely esetén az esetleg létező kereteket eltüntetve az egész ablakot használja a böngésző az oldal megjelenítésére.
- `_PARENT` érték hasonló a _TOP értékhez, de ha több szintű a keretek használata, akkor csak a legalsó szint helyére kerül az új tartalom.

Tárgymutató