Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar
Python
reguláris kifejezések

Dr. Schuster György

2017. november 13.

Alapfogalmak

A reguláris kifejezések a következő célra szolgálnak:

Alapfogalmak

A reguláris kifejezések a következő célra szolgálnak:

- egy adott bemenet megfelel-e egy adott formátumnak,

Alapfogalmak

A reguláris kifejezések a következő célra szolgálnak:

- egy adott bemenet megfelel-e egy adott formátumnak,
- egy adott szövegrészletben egy adott minta előfordul-e,

Alapfogalmak

A reguláris kifejezések a következő célra szolgálnak:

- egy adott bemenet megfelel-e egy adott formátumnak,
- egy adott szövegrészletben egy adott minta előfordul-e,
- egy adott szövegből, egy adott mintát ki lehessen emelni,

Alapfogalmak

A reguláris kifejezések a következő célra szolgálnak:

- egy adott bemenet megfelel-e egy adott formátumnak,
- egy adott szövegrészletben egy adott minta előfordul-e,
- egy adott szövegből, egy adott mintát ki lehessen emelni,
- egy adott szöveg elemet, ki lehessen cserélni egy másikra,

Alapfogalmak

A reguláris kifejezések a következő célra szolgálnak:

- egy adott bemenet megfelel-e egy adott formátumnak,
- egy adott szövegrészletben egy adott minta előfordul-e,
- egy adott szövegből, egy adott mintát ki lehessen emelni,
- egy adott szöveg elemet, ki lehessen cserélni egy másikra,
- egy szöveget valamilyen szabályok szerint kisebb részekre lehessen bontani.

re modul függvényei

Ha reguláris kifejezéseket szeretnénk használni, akkor be kell olvasni a **re** modult.

re modul függvényei

Ha reguláris kifejezéseket szeretnénk használni, akkor be kell olvasni a **re** modult.

A **re** modul függvényeiről és változóiról most csak egy rövid leírást adunk. A későbbiekben részletesen példákkal kerülnek bemutatásra.

re modul függvényei

Ha reguláris kifejezéseket szeretnénk használni, akkor be kell olvasni a re modult.

A re modul függvényei:

re modul függvényei

Ha reguláris kifejezéseket szeretnénk használni, akkor be kell olvasni a **re** modult.

A **re** modul függvényei:

re.compile a reguláris kifejezést lefordítja egy objektummá, ami a későbbiekben felhasználható. Az előállított objektum függvényei **match()** és a **search()** függvények.

re modul függvényei

Ha reguláris kifejezéseket szeretnénk használni, akkor be kell olvasni a **re** modult.

A **re** modul függvényei:

re.compile a reguláris kifejezést lefordítja egy objektummá, ami a későbbiekben felhasználható. Az előállított objektum függvényei **match()** és a **search()** függvények.

A **compile** függvény használata:

re modul függvényei

Ha reguláris kifejezéseket szeretnénk használni, akkor be kell olvasni a **re** modult.

A **re** modul függvényei:

re.compile a reguláris kifejezést lefordítja egy objektummá, ami a későbbiekben felhasználható. Az előállított objektum függvényei **match()** és a **search()** függvények.

A **compile** függvény használata:

A generált objektum.

```
object=   .        (        ,        )
```

re modul függvényei

Ha reguláris kifejezéseket szeretnénk használni, akkor be kell olvasni a **re** modult.

A **re** modul függvényei:

re.compile a reguláris kifejezést lefordítja egy objektummá, ami a későbbiekben felhasználható. Az előállított objektum függvényei **match()** és a **search()** függvények.

A **compile** függvény használata:

```
object=re.          (          ,          )
```

A generált objektum. A modul.

re modul függvényei

Ha reguláris kifejezéseket szeretnénk használni, akkor be kell olvasni a **re** modult.
A **re** modul függvényei:

re.compile a reguláris kifejezést lefordítja egy objektummá, ami a későbbiekben felhasználható. Az előállított objektum függvényei **match()** és a **search()** függvények.
A **compile** függvény használata:

```
object=re.compile(          ,          )
```

A generált objektum. A modul.
A függvény.

re modul függvényei

Ha reguláris kifejezéseket szeretnénk használni, akkor be kell olvasni a **re** modult.
A **re** modul függvényei:

re.compile a reguláris kifejezést lefordítja egy objektummá, ami a későbbiekben felhasználható. Az előállított objektum függvényei **match()** és a **search()** függvények.
A **compile** függvény használata:

```
object=re.compile(pattern,        )
```

A generált objektum.
A modul.
A függvény.
A reguláris kifejezés.

re modul függvényei

Ha reguláris kifejezéseket szeretnénk használni, akkor be kell olvasni a **re** modult.
A **re** modul függvényei:

re.compile a reguláris kifejezést lefordítja egy objektummá, ami a későbbiekben felhasználható. Az előállított objektum függvényei **match()** és a **search()** függvények.
A **compile** függvény használata:

```
object=re.compile(pattern,flags)
```

A generált objektum. A modul.
A függvény.
A reguláris kifejezés.
Módosító flag-ek.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.
A **match** függvény használata:

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=   .      (        ,        ,      )
```

Az eredmény objektum.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.      (       ,       )
```

Az eredmény objektum.

A modul.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.match(          ,          ,          )
```

Az eredmény objektum.
A modul.
A függvény.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.match(pattern,      ,      )
```

Az eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.match(pattern,string,      )
```

Az eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.
A sztring, amiben keresünk.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.match(pattern,string,flags)
```

Az eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.
A sztring, amiben keresünk.
Módosító flag-ek.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.match(pattern,string,flags)
```

Az eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.
A sztring, amiben keresünk.
Módosító flag-ek.

re.search() Az egész sztringet végignézi és az első egyezés adataival tér vissza. Ha nem talál egyezést akkor a visszatérési értéke **None**.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.match(pattern,string,flags)
```

Az eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.
A sztring, amiben keresünk.
Módosító flag-ek.

re.search() Az egész sztringet végignézi és az első egyezés adataival tér vissza. Ha nem talál egyezést akkor a visszatérési értéke **None**.

A **search** függvény használata:

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.match(pattern,string,flags)
```

Az eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.
A sztring, amiben keresünk.
Módosító flag-ek.

re.search() Az egész sztringet végignézi és az első egyezés adataival tér vissza. Ha nem talál egyezést akkor a visszatérési értéke **None**.

A **search** függvény használata:

```
object=   .       (       ,       ,       )
```

A eredmény objektum.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.match(pattern,string,flags)
```

Az eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.
A sztring, amiben keresünk.
Módosító flag-ek.

re.search() Az egész sztringet végignézi és az első egyezés adataival tér vissza. Ha nem talál egyezést akkor a visszatérési értéke **None**.

A **search** függvény használata:

```
object=re.       (       ,       )
```

A eredmény objektum.
A modul.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.match(pattern,string,flags)
```

Az eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.
A sztring, amiben keresünk.
Módosító flag-ek.

re.search() Az egész sztringet végignézi és az első egyezés adataival tér vissza. Ha nem talál egyezést akkor a visszatérési értéke **None**.

A **search** függvény használata:

```
object=re.search(        ,        ,        )
```

A eredmény objektum.
A modul.
A függvény.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.match(pattern,string,flags)
```

Az eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.
A sztring, amiben keresünk.
Módosító flag-ek.

re.search() Az egész sztringet végignézi és az első egyezés adataival tér vissza. Ha nem talál egyezést akkor a visszatérési értéke **None**.

A **search** függvény használata:

```
object=re.search(pattern,      ,      )
```

A eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.match(pattern,string,flags)
```

Az eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.
A sztring, amiben keresünk.
Módosító flag-ek.

re.search() Az egész sztringet végignézi és az első egyezés adataival tér vissza. Ha nem talál egyezést akkor a visszatérési értéke **None**.

A **search** függvény használata:

```
object=re.search(pattern,string,      )
```

A eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.
A sztring, amiben keresünk.

re modul függvényei

re.match() Ha nulla, vagy több karakter megfelel a keresési mintának a sztring elején, akkor visszatér a megfelelő keresési objektummal. Ha nem, akkor a visszatérési érték **None**.

A **match** függvény használata:

```
object=re.match(pattern,string,flags)
```

Az eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.
A sztring, amiben keresünk.
Módosító flag-ek.

re.search() Az egész sztringet végignézi és az első egyezés adataival tér vissza. Ha nem talál egyezést akkor a visszatérési értéke **None**.

A **search** függvény használata:

```
object=re.search(pattern,string,flags)
```

A eredmény objektum.
A modul.
A függvény.
A reguláris kifejezés.
A sztring, amiben keresünk.
Módosító flag-ek.

re modul függvényei

A match.object felépítése, példa:

re modul függvényei

A match.object felépítése, példa:

```
>>> a=re.search("bc","abcdabde")
>>> print(a)
```

re modul függvényei

A match.object felépítése, példa:

```
>>> a=re.search("bc","abcdabde")
>>> print(a)
```

A kiprintelt match.object:

re modul függvényei

A match.object felépítése, példa:

```
>>> a=re.search("bc","abcdabde")
>>> print(a)
```

A kiprintelt match.object:

```
<_sre.SRE_Match object; span=(1, 3), match='bc'>
```

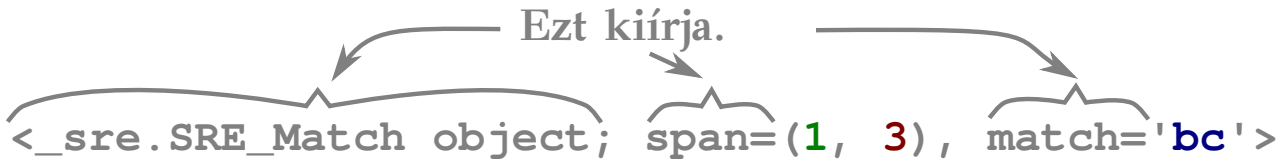
re modul függvényei

A match.object felépítése, példa:

```
>>> a=re.search("bc","abcdabde")
>>> print(a)
```

Nem túl informatív!!

A kiprintelt match.object:



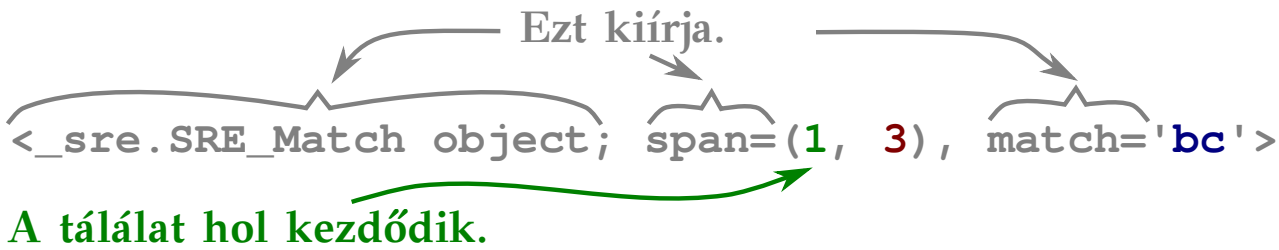
re modul függvényei

A match.object felépítése, példa:

```
>>> a=re.search("bc","abcdabde")
>>> print(a)
```

Itt kedődik a találat.

A kiprintelt match.object:



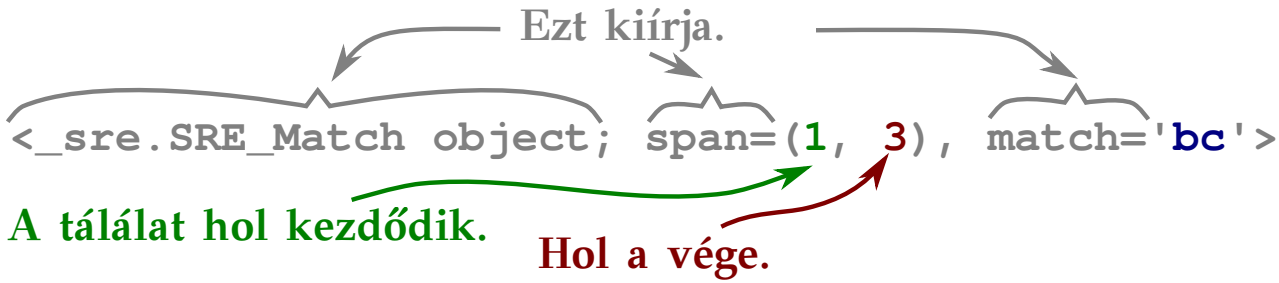
re modul függvényei

A match.object felépítése, példa:

```
>>> a=re.search("bc","abcdabde")
>>> print(a)
```

Ez már nincs a találatban.

A kiprintelt match.object:



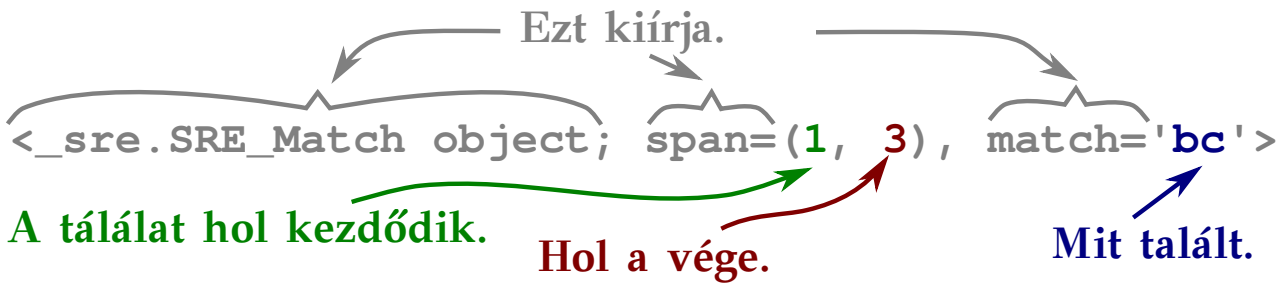
re modul függvényei

A match.object felépítése, példa:

```
>>> a=re.search("bc","abcdabde")
>>> print(a)
```

Ezt kerestük.

A kiprintelt match.object:

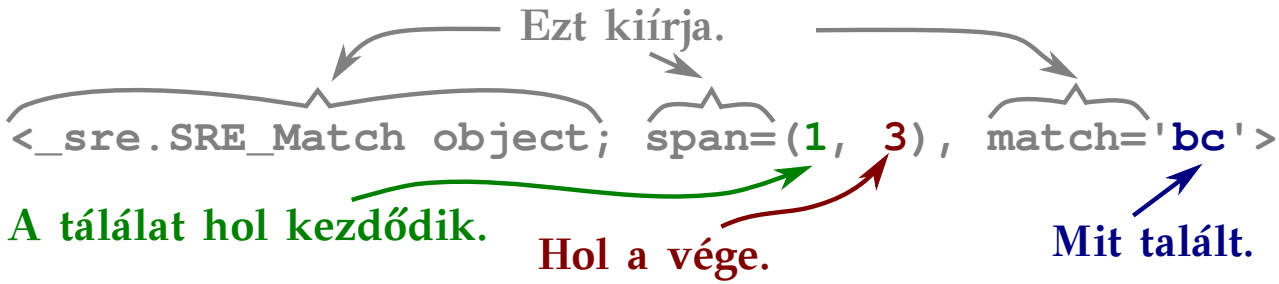


re modul függvényei

A match.object felépítése, példa:

```
>>> a=re.search("bc","abcdabde")
>>> print(a)
```

A kiprintelt match.object:



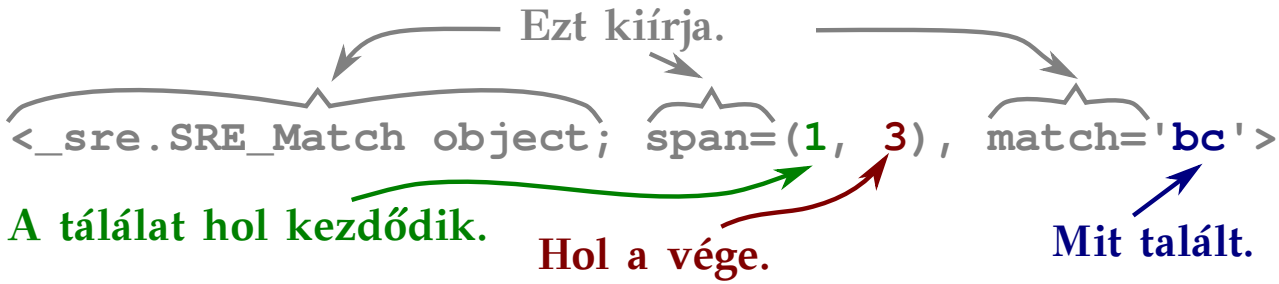
A match.object-nek is vannak függvényei (ne kelljen bogarászni).

re modul függvényei

A match.object felépítése, példa:

```
>>> a=re.search("bc","abcdabde")
>>> print(a)
```

A kiprintelt match.object:



A match.object-nek is vannak függvényei (ne kelljen bogarászni).
Kicsit később.

re modul függvényei

`re.split()` A sztringet a megadott minta szerint szétdarabolja.

re modul függvényei

re.split() A sztringet a megadott minta szerint szétdarabolja.
A **split** függvény használata:

re modul függvényei

`re.split()` A sztringet a megadott minta szerint szétdarabolja.
A `split` függvény használata:

```
list=   .      (        ,        ,        ,      )
```

A eredmény tömb.

re modul függvényei

`re.split()` A sztringet a megadott minta szerint szétdarabolja.
A `split` függvény használata:

```
list=re.      (        ,        ,          ,      )
```

A eredmény tömb.
A modul.

re modul függvényei

`re.split()` A sztringet a megadott minta szerint szétdarabolja.
A `split` függvény használata:

```
list=re.split(        ,        ,          ,       )
```

A eredmény tömb.
A modul.
A függvény.

re modul függvényei

re.split() A sztringet a megadott minta szerint szétdarabolja.
A **split** függvény használata:

```
list=re.split(pattern,        ,          ,       )
```

A eredmény tömb.
A modul.
A függvény.
A reguláris kifejezés, ami a határoló minta.

re modul függvényei

`re.split()` A sztringet a megadott minta szerint szétdarabolja.
A `split` függvény használata:

```
list=re.split(pattern,string,          ,      )
```

A eredmény tömb.
A modul.
A függvény.
A reguláris kifejezés, ami a határoló minta.
A sztring, amit szeletelünk.

re modul függvényei

`re.split()` A sztringet a megadott minta szerint szétdarabolja.
A `split` függvény használata:

```
list=re.split(pattern,string,maxsplit,      )
```

A eredmény tömb.
A modul.
A függvény.
A reguláris kifejezés, ami a határoló minta.
A sztring, amit szeletelünk.
Ha nem 0, akkor max annyi elemre szeletel.

re modul függvényei

re.split() A sztringet a megadott minta szerint szétdarabolja.
A **split** függvény használata:

```
list=re.split(pattern,string,maxsplit,flags)
```

A eredmény tömb.
A modul.
A függvény.
A reguláris kifejezés, ami a határoló minta.
A sztring, amit szeletelünk.
Ha nem 0, akkor max annyi elemre szeletel.
Módosító flag-ek

re modul függvényei

re.split() A sztringet a megadott minta szerint szétdarabolja.
A **split** függvény használata:

```
list=re.split(pattern,string,maxsplit,flags)
```

A eredmény tömb.
A modul.
A függvény.
A reguláris kifejezés, ami a határoló minta.
A sztring, amit szeletelünk.
Ha nem 0, akkor max annyi elemre szeletel.
Módosító flag-ek

split példa:

```
>>> import re
>>> re.split('a','abababababababab')
['', 'b', 'b', 'b', 'b', 'b', 'b', 'b']
```

re modul függvényei

`re.split()` A sztringet a megadott minta szerint szétdarabolja.
A `split` függvény használata:

```
list=re.split(pattern,string,maxsplit,flags)
```

A eredmény tömb.
A modul.
A függvény.
A reguláris kifejezés, ami a határoló minta.
A sztring, amit szeletelünk.
Ha nem 0, akkor max annyi elemre szeletel.
Módosító flag-ek

split példa maxsplit-tel:

```
>>> import re
>>> re.split('a','abababababababab',2)
['', 'b', 'bababababab']
```

re modul függvényei

`re.sub()` Megadott rész sztringet egy másikra cserél.

re modul függvényei

re.sub() Megadott rész sztringet egy másikra cserél.

A **sub** függvény használata:

re modul függvényei

`re.sub()` Megadott rész sztringet egy másikra cserél.

A `sub` függvény használata:

`re.  (  ,  ,  ,  ,  )`

A modul.

re modul függvényei

`re.sub()` Megadott rész sztringet egy másikra cserél.

A `sub` függvény használata:

```
re.sub(          ,       ,          ,       ,       )
```

A modul.
A függvény.

re modul függvényei

re.sub() Megadott rész sztringet egy másikra cserél.

A **sub** függvény használata:

```
re.sub(pattern,      ,      ,      ,      )
```

A modul.
A függvény.
Amit cserélünk.

re modul függvényei

re.sub() Megadott rész sztringet egy másikra cserél.

A **sub** függvény használata:

```
re.sub(pattern,repl,        ,        ,        )
```

A modul.
A függvény.
Amit cserélünk.
Amire cserélünk.

re modul függvényei

re.sub() Megadott rész sztringet egy másikra cserél.

A **sub** függvény használata:

```
re.sub(pattern,repl,string,      ,      )
```

A modul.
A függvény.
Amit cserélünk.
Amire cserélünk.
A sztring, amiben dolgozunk.

re modul függvényei

re.sub() Megadott rész sztringet egy másikra cserél.

A **sub** függvény használata:

```
re.sub(pattern,repl,string,count,      )
```

A modul.
A függvény.
Amit cserélünk.
Amire cserélünk.
A sztring, amiben dolgozunk.
Hány darabot cserélünk.

re modul függvényei

re.sub() Megadott rész sztringet egy másikra cserél.

A **sub** függvény használata:

```
re.sub(pattern,repl,string,count,flags)
```

A modul.
A függvény.
Amit cserélünk.
Amire cserélünk.
A sztring, amiben dolgozunk.
Hány darabot cserélünk.
Módosító flag-ek.

re modul függvényei

re.sub() Megadott rész sztringet egy másikra cserél.

A **sub** függvény használata:

```
re.sub(pattern,repl,string,count,flags)
```

A modul.
A függvény.
Amit cserélünk.
Amire cserélünk.
A sztring, amiben dolgozunk.
Hány darabot cserélünk.
Módosító flag-ek.

sub példa:

```
>>> import re
str="abcdabcd"
>>> re.sub('a','A',str)
'AbcdAbcd'
```

re modul függvényei

re.sub() Megadott rész sztringet egy másikra cserél.

A **sub** függvény használata:

```
re.sub(pattern,repl,string,count,flags)
```

A modul.
A függvény.
Amit cserélünk.
Amire cserélünk.
A sztring, amiben dolgozunk.
Hány darabot cserélünk.
Módosító flag-ek.

sub példa count-tal:

```
>>> import re
str="abcdabcd"
>>> re.sub('a','A',str,1)
'Abcdabcd'
```

Módosító flag-ek

A flag-ek a reguláris kifejezést módosítják és a találat értelmezését módosítják.

Módosító flag-ek

A flag-ek a reguláris kifejezést módosítják és a találat értelmezését módosítják.

Flag-ek:

Módosító flag-ek

A flag-ek a reguláris kifejezést módosítják és a találat értelmezését módosítják.

Flag-ek:

`re.A`	helyettesítő karakterek (`\w, \W, \b, \B, \d, \D,`
`re.ASCII`	`\s és \S`) esetén csak az ASCII mintákra illeszkedik.

Módosító flag-ek

A flag-ek a reguláris kifejezést módosítják és a találat értelmezését módosítják.

Flag-ek:

`re.A`	helyettesítő karakterek (`\w, \W, \b, \B, \d, \D, \s és \S`) esetén csak az ASCII mintákra illeszkedik.
`re.ASCII`	
`re.I`	nem tesz különbséget a nagy és kisbetűk között.
`re.IGNORECASE`	

Módosító flag-ek

A flag-ek a reguláris kifejezést módosítják és a találat értelmezését módosítják.

Flag-ek:

`re.A`	helyettesítő karakterek (`\w, \W, \b, \B, \d, \D, \s és \S`) esetén csak az ASCII mintákra illeszkedik.
`re.ASCII`	
`re.I`	nem tesz különbséget a nagy és kisbetűk között.
`re.IGNORECASE`	
`re.M`	a sztring kezdete (^) és a sztring vége (`$`) jelzést több sor esetén minden sorra alkalmazza.
`re.MULTILINE`	

Módosító flag-ek

A flag-ek a reguláris kifejezést módosítják és a találat értelmezését módosítják.

Flag-ek:

`re.A`	helyettesítő karakterek (`\w, \W, \b, \B, \d, \D, \s és \S`) esetén csak az ASCII mintákra illeszkedik.
`re.ASCII`	
`re.I`	nem tesz különbséget a nagy és kisbetűk között.
`re.IGNORECASE`	
`re.M`	a sztring kezdete (`^`) és a sztring vége (`$`) jelzést több sor esetén minden sorra alkalmazza.
`re.MULTILINE`	
`re.S`	az általános helyettesítő karaktert (`.`) az újsor karakterre (`\n`) is értelmezi.
`re.DOTALL`	

Lefordított regex

A **`re.compile`** függvény egy lefordított objektumot ad vissza.

Lefordított regex

A **`re.compile`** függvény egy lefordított objektumot ad vissza.

A fordítás menete:

Lefordított regex

A **`re.compile`** függvény egy lefordított objektumot ad vissza.

A fordítás menete:

A lefordított objektum.

`regex=` `.` `( " " )`

Lefordított regex

A **re.compile** függvény egy lefordított objektumot ad vissza.

A fordítás menete:

```
regex=re.          (  "       ")
```

A lefordított objektum.
A **re** csomag.

Lefordított regex

A **re.compile** függvény egy lefordított objektumot ad vissza.

A fordítás menete:

```
regex=re.compile( "       ")
```

A lefordított objektum.
A **re** csomag.
a **compile** függvény.

Lefordított regex

A **re.compile** függvény egy lefordított objektumot ad vissza.

A fordítás menete:

```
regex=re.compile(r"        ")
```

A lefordított objektum.
A **re** csomag.
a **compile** függvény.
Azt jelzi, hogy ”raw” sztring jön.

Lefordított regex

A **re.compile** függvény egy lefordított objektumot ad vissza.

A fordítás menete:

```
regex=re.compile(r"regex")
```

A lefordított objektum.
A **re** csomag.
a **compile** függvény.
Azt jelzi, hogy ”raw” sztring jön.
A reguláris kifejezés.

Lefordított regex

A **re.compile** függvény egy lefordított objektumot ad vissza.

A fordítás menete:

```
regex=re.compile(r"regex")
```

A lefordított objektum.
A **re** csomag.
a **compile** függvény.
Azt jelzi, hogy ”raw” sztring jön.
A reguláris kifejezés.

Példa:

```
>>> import re
>>> m=re.compile(r"Hello")
```

Lefordított regex függvényei

`regex.match()` A sztring adott pozíciójától vizsgálja a mintának megfelelést. Ha találat van a **`match.object`**-ben vannak a jellemzők, ha nincs a visszatérési érték **`None`**.

Lefordított regex függvényei

`regex.match()` A sztring adott pozíciójától vizsgálja a mintának megfelelést. Ha találat van a **`match.object`**-ben vannak a jellemzők, ha nincs a visszatérési érték **`None`**.

A **`match`** függvény használata:

Lefordított regex függvényei

`regex.match()` A sztring adott pozíciójától vizsgálja a mintának megfelelést. Ha találat van a **`match.object`**-ben vannak a jellemzők, ha nincs a visszatérési érték **`None`**.

A **`match`** függvény használata:

A eredmény objektum.

```
object=      .      (      ,    ,      )
```

Lefordított regex függvényei

regex.match() A sztring adott pozíciójától vizsgálja a mintának megfelelést. Ha találat van a **match.object**-ben vannak a jellemzők, ha nincs a visszatérési érték **None**.

A **match** függvény használata:

A eredmény objektum.
A lefordított reguláris kifejezés.

```
object=regex.      (      ,    ,      )
```

Lefordított regex függvényei

`regex.match()` A sztring adott pozíciójától vizsgálja a mintának megfelelést. Ha találat van a **`match.object`**-ben vannak a jellemzők, ha nincs a visszatérési érték **`None`**.

A **`match`** függvény használata:

```
object=regex.match(          ,     ,          )
```

A eredmény objektum.
A lefordított reguláris kifejezés.
A függvény.

Lefordított regex függvényei

regex.match() A sztring adott pozíciójától vizsgálja a mintának megfelelést. Ha találat van a **match.object**-ben vannak a jellemzők, ha nincs a visszatérési érték **None**.

A **match** függvény használata:

```
object=regex.match(string,     ,          )
```

A eredmény objektum.
A lefordított reguláris kifejezés.
A függvény.
A sztring.

Lefordított regex függvényei

regex.match() A sztring adott pozíciójától vizsgálja a mintának megfelelést. Ha találat van a **match.object**-ben vannak a jellemzők, ha nincs a visszatérési érték **None**.

A **match** függvény használata:

```
object=regex.match(string,pos,          )
```

A eredmény objektum.
A lefordított reguláris kifejezés.
A függvény.
A sztring.
Honnan vizsgál (nem kötelező).

Lefordított regex függvényei

regex.match() A sztring adott pozíciójától vizsgálja a mintának megfelelést. Ha találat van a **match.object**-ben vannak a jellemzők, ha nincs a visszatérési érték **None**.

A **match** függvény használata:

```
object=regex.match(string,pos,endpos)
```

A eredmény objektum.
A lefordított reguláris kifejezés.
A függvény.
A sztring.
Honnan vizsgál (nem kötelező).
Meddig vizsgál (nem kötelező).

Lefordított regex függvényei

regex.match() A sztring adott pozíciójától vizsgálja a mintának megfelelést. Ha találat van a **match.object**-ben vannak a jellemzők, ha nincs a visszatérési érték **None**.

A **match** függvény használata:

```
object=regex.match(string,pos,endpos)
```

A eredmény objektum.
A lefordított reguláris kifejezés.
A függvény.
A sztring.

match példa:

```
>>> import re
>>> regex=re.compile(r"u")
>>> regex.search("kutya",1)
<_sre.SRE_Match object; span=(1, 2), match='u'>
```

Lefordított regex függvényei

regex.match() A sztring adott pozíciójától vizsgálja a mintának megfelelést. Ha találat van a **match.object**-ben vannak a jellemzők, ha nincs a visszatérési érték **None**.

A **match** függvény használata:

```
object=regex.match(string,pos,endpos)
```

A eredmény objektum.
A lefordított reguláris kifejezés.
A függvény.
A sztring.

match példa:

```
>>> import re
>>> regex=re.compile(r"u")
>>> regex.search("kutya")
>>>
```

Lefordított regex függvényei

`regex.search()` A sztringet végignézi (ha nem adunk meg határokat) és az első egyezés adataival tér vissza. Ha nem talál egyezést, akkor a visszatérési értéke **`None`**.

Lefordított regex függvényei

`regex.search()` A sztringet végignézi (ha nem adunk meg határokat) és az első egyezés adataival tér vissza. Ha nem talál egyezést, akkor a visszatérési értéke **`None`**.

A **`search`** függvény használata:

Lefordított regex függvényei

`regex.search()` A sztringet végignézi (ha nem adunk meg határokat) és az első egyezés adataival tér vissza. Ha nem talál egyezést, akkor a visszatérési értéke **`None`**.

A **`search`** függvény használata:

A eredmény objektum.

```
object=      .      (      ,    ,      )
```

Lefordított regex függvényei

regex.search() A sztringet végignézi (ha nem adunk meg határokat) és az első egyezés adataival tér vissza. Ha nem talál egyezést, akkor a visszatérési értéke **None**.

A **search** függvény használata:

A eredmény objektum.
A lefordított reguláris kifejezés.

```
object=regex.        (       ,    ,       )
```

Lefordított regex függvényei

`regex.search()` A sztringet végignézi (ha nem adunk meg határokat) és az első egyezés adataival tér vissza. Ha nem talál egyezést, akkor a visszatérési értéke **`None`**.

A **`search`** függvény használata:

```
object=regex.search(          ,     ,          )
```

A eredmény objektum.
A lefordított reguláris kifejezés.
A függvény.

Lefordított regex függvényei

regex.search() A sztringet végignézi (ha nem adunk meg határokat) és az első egyezés adataival tér vissza. Ha nem talál egyezést, akkor a visszatérési értéke **None**.

A **search** függvény használata:

```
object=regex.search(string,     ,          )
```

A eredmény objektum.
A lefordított reguláris kifejezés.
A függvény.
A sztring.

Lefordított regex függvényei

`regex.search()` A sztringet végignézi (ha nem adunk meg határokat) és az első egyezés adataival tér vissza. Ha nem talál egyezést, akkor a visszatérési értéke **`None`**.

A **`search`** függvény használata:

```
object=regex.search(string,pos,      )
```

A eredmény objektum.
A lefordított reguláris kifejezés.
A függvény.
A sztring.
Honnan vizsgál (nem kötelező).

Lefordított regex függvényei

`regex.search()` A sztringet végignézi (ha nem adunk meg határokat) és az első egyezés adataival tér vissza. Ha nem talál egyezést, akkor a visszatérési értéke **`None`**.

A **`search`** függvény használata:

```
object=regex.search(string,pos,endpos)
```

A eredmény objektum.
A lefordított reguláris kifejezés.
A függvény.
A sztring.
Honnan vizsgál (nem kötelező).
Meddig vizsgál (nem kötelező).

Lefordított regex függvényei

regex.search() A sztringet végignézi (ha nem adunk meg határokat) és az első egyezés adataival tér vissza. Ha nem talál egyezést, akkor a visszatérési értéke **None**.

A **search** függvény használata:

```
object=regex.search(string,pos,endpos)
```

A eredmény objektum.
A lefordított reguláris kifejezés.
A függvény.
A sztring.

search példa:

```
>>> import re
regex=re.compile(r"u")
>>> regex.search("kutya")
<_sre.SRE_Match object; span=(1, 2), match='u'>
```

Lefordított regex függvényei

`regex.findall()` A megfelelő kifejezést keresi a teljes sztingben vagy az adott tartományban. A visszatérési érték egy lista, amiben a találatok szerepelnek. Ha nincs találat a lista üres.

Lefordított regex függvényei

`regex.findall()` A megfelelő kifejezést keresi a teljes sztingben vagy az adott tartományban. A visszatérési érték egy lista, amiben a találatok szerepelnek. Ha nincs találat a lista üres.

A **`findall`** függvény használata:

Lefordított regex függvényei

`regex.findall()` A megfelelő kifejezést keresi a teljes sztingben vagy az adott tartományban. A visszatérési érték egy lista, amiben a találatok szerepelnek. Ha nincs találat a lista üres.

A **`findall`** függvény használata:

A eredmény lista.

```
list=      .        (      ,    ,      )
```

Lefordított regex függvényei

`regex.findall()` A megfelelő kifejezést keresi a teljes sztingben vagy az adott tartományban. A visszatérési érték egy lista, amiben a találatok szerepelnek. Ha nincs találat a lista üres.

A **`findall`** függvény használata:

A eredmény lista.
A lefordított reguláris kifejezés.

```
list=regex.        (       ,    ,       )
```

Lefordított regex függvényei

`regex.findall()` A megfelelő kifejezést keresi a teljes sztingben vagy az adott tartományban. A visszatérési érték egy lista, amiben a találatok szerepelnek. Ha nincs találat a lista üres.

A **`findall`** függvény használata:

```
list=regex.findall(        ,    ,        )
```

A eredmény lista.
A lefordított reguláris kifejezés.
A függvény.

Lefordított regex függvényei

regex.findall() A megfelelő kifejezést keresi a teljes sztingben vagy az adott tartományban. A visszatérési érték egy lista, amiben a találatok szerepelnek. Ha nincs találat a lista üres.

A **findall** függvény használata:

```
list=regex.findall(string,     ,          )
```

A eredmény lista.
A lefordított reguláris kifejezés.
A függvény.
A sztring.

Lefordított regex függvényei

regex.findall() A megfelelő kifejezést keresi a teljes sztingben vagy az adott tartományban. A visszatérési érték egy lista, amiben a találatok szerepelnek. Ha nincs találat a lista üres.

A **findall** függvény használata:

```
list=regex.findall(string,pos,          )
```

A eredmény lista.
A lefordított reguláris kifejezés.
A függvény.
A sztring.
Honnan vizsgál (nem kötelező).

Lefordított regex függvényei

`regex.findall()` A megfelelő kifejezést keresi a teljes sztingben vagy az adott tartományban. A visszatérési érték egy lista, amiben a találatok szerepelnek. Ha nincs találat a lista üres.

A **`findall`** függvény használata:

```
list=regex.findall(string,pos,endpos)
```

A eredmény lista.
A lefordított reguláris kifejezés.
A függvény.
A sztring.
Honnan vizsgál (nem kötelező).
Meddig vizsgál (nem kötelező).

Lefordított regex függvényei

regex.findall() A megfelelő kifejezést keresi a teljes sztingben vagy az adott tartományban. A visszatérési érték egy lista, amiben a találatok szerepelnek. Ha nincs találat a lista üres.

A **findall** függvény használata:

A eredmény lista.
A lefordított reguláris kifejezés.
A függvény.

```
list=regex.findall(string,pos,endpos)
```

findall példa:

```
>>> import re
>>> regex=re.compile(r"u")
>>> list=regex.findall("kutyakutyakutya")
>>> print(list)
['u', 'u', 'u']
```

Lefordított regex függvényei

regex.findall() A megfelelő kifejezést keresi a teljes sztingben vagy az adott tartományban. A visszatérési érték egy lista, amiben a találatok szerepelnek. Ha nincs találat a lista üres.

A **findall** függvény használata:

A eredmény lista.
A lefordított reguláris kifejezés.
A függvény.

```
list=regex.findall(string,pos,endpos)
```

findall példa:

```
>>> import re
>>> regex=re.compile(r"i")
>>> list=regex.findall("kutyakutyakutya")
>>> print(list)
[]
```

Lefordított regex függvényei

`regex.sub()` Működése megegyezik a **`re.sub()`** függvény működésével.

Lefordított regex függvényei

`regex.sub()` Működése megegyezik a **`re.sub()`** függvény működésével.

A **`sub`** függvény használata:

Lefordított regex függvényei

`regex.sub()` Működése megegyezik a **`re.sub()`** függvény működésével.

A **`sub`** függvény használata:

A lefordított reguláris kifejezés.

`regex.` `(` `,` `,` `)`

Lefordított regex függvényei

`regex.sub()` Működése megegyezik a **`re.sub()`** függvény működésével.

A **`sub`** függvény használata:

A lefordított reguláris kifejezés.
A függvény.

```
regex.sub(      ,       ,       )
```

Lefordított regex függvényei

`regex.sub()` Működése megegyezik a **`re.sub()`** függvény működésével.

A **`sub`** függvény használata:

```
regex.sub(repl,        ,        )
```

A lefordított reguláris kifejezés.
A függvény.
A helyettesítő minta.

Lefordított regex függvényei

`regex.sub()` Működése megegyezik a **`re.sub()`** függvény működésével.

A **`sub`** függvény használata:

```
regex.sub(repl,string,      )
```

A lefordított reguláris kifejezés.
A függvény.
A helyettesítő minta.
A sztring.

Lefordított regex függvényei

`regex.sub()` Működése megegyezik a **`re.sub()`** függvény működésével.

A **`sub`** függvény használata:

```
regex.sub(repl,string,count)
```

A lefordított reguláris kifejezés.
A függvény.
A helyettesítő minta.
A sztring.
Hány helyettesítés legyen (nem kötelező).

Lefordított regex függvényei

`regex.sub()` Működése megegyezik a **`re.sub()`** függvény működésével.

A **`sub`** függvény használata:

```
regex.sub(repl,string,count)
```

A lefordított reguláris kifejezés.
A függvény.
A helyettesítő minta.
A sztring.
Hány helyettesítés legyen

`sub` példa:

```
>>> import re
>>> a="abcdabcdabcd"
>>> regex=re.compile(r"ab")
>>> regex.sub("AB",a)
'ABcdABcdABcd'
```

Lefordított regex függvényei

`regex.sub()` Működése megegyezik a **`re.sub()`** függvény működésével.

A **`sub`** függvény használata:

```
regex.sub(repl,string,count)
```

A lefordított reguláris kifejezés.
A függvény.
A helyettesítő minta.
A sztring.
Hány helyettesítés legyen

`sub` példa:

```
>>> import re
>>> a="abcdabcdabcd"
>>> regex=re.compile(r"ab")
>>> regex.sub("AB",a,2)
'ABcdABcdabcd'
```

A `match object` függvényei

A **`re`** modul és a lefordított reguláris kifejezés kimenete egy találati objektum **`match object`**. Ez a **`match object`**amellett, hogy rendelkezik egy BOOL jellegű értékkel, több függvényt is implementál.

A match object függvényei

A **re** modul és a lefordított reguláris kifejezés kimenete egy találati objektum **match object**. Ez a **match object**amellett, hogy rendelkezik egy BOOL jellegű értékkel, több függvényt is implementál.

Ezek a függvények könnyebbé teszik az eredmény értelmezését.

A `match object` függvényei

A **`re`** modul és a lefordított reguláris kifejezés kimenete egy találati objektum **`match object`**. Ez a **`match object`**amellett, hogy rendelkezik egy BOOL jellegű értékkel, több függvényt is implementál.

Ezek a függvények könnyebbé teszik az eredmény értelmezését.

A `match object`"előállítása"

```
>>> import re
>>> match=re.search(...)
```

A match object függvényei

A **re** modul és a lefordított reguláris kifejezés kimenete egy találati objektum **match object**. Ez a **match object**amellett, hogy rendelkezik egy BOOL jellegű értékkel, több függvényt is implementál.

Ezek a függvények könnyebbé teszik az eredmény értelmezését.

A match object"előállítása"

```
>>> import re
>>> regex=re.compile(...)
>>> match=regex.search(...)
```

A match object függvényei

`match.group()` A találatok sztring(elemeivel) tér vissza.

A match object függvényei

match.group() A találatok sztring(elemeivel) tér vissza.

A **group** függvény használata:

A match object függvényei

`match.group()` A találatok sztring(elemeivel) tér vissza.

A `group` függvény használata:

Az eredmény.

`m=  .  (  )`

A match object függvényei

match.group() A találatok sztring(elemeivel) tér vissza.

A **group** függvény használata:

```
m=match.          (                )
```

Az eredmény.

A **match object**.

A match object függvényei

match.group() A találatok sztring(elemeivel) tér vissza.

A **group** függvény használata:

```
m=match.group(          )
```

Az eredmény.
A **match object**.
A függvény

A match object függvényei

`match.group()` A találatok sztring(elemeivel) tér vissza.

A `group` függvény használata:

```
m=match.group(group,...)
```

Az eredmény.
A `match object`.
A függvény
A találati paraméterek (lásd alábbi példa).

A match object függvényei

match.group() A találatok sztring(elemeivel) tér vissza.

A group függvény használata:

group példa:

```
>>> import re
>>> a="Humpty Dumpty sat on the wall"
>>> m=re.match(r"(\w+) (\w+)" )
>>> m.group(0)
'Humpty Dumpty'
>>> m.group(1)
'Humpty'
>>> m.group(2)
'Dumpty'
>>> m.group(1,2)
'Humpty', 'Dumpty'
```

ek (lásd

A match object függvényei

match.groupdict() Egy aszociációs tömbbel tér vissza, amely tartalmazza a találat rész sztringeket.

A match object függvényei

match.groupdict() Egy aszociációs tömbbel tér vissza, amely tartalmazza a találat rész sztringeket.

A **groupdict** függvény használata:

A match object függvényei

match.groupdict() Egy aszociációs tömbbel tér vissza, amely tartalmazza a találat rész sztringeket.

A **groupdict** függvény használata:

match. **()** A **match object**.

A `match object` függvényei

`match.groupdict()` Egy aszociációs tömbbel tér vissza, amely tartalmazza a találat rész sztringeket.

A **`groupdict`** függvény használata:

```
match.groupdict()
```

A **`match object`**.
A függvény.

A match object függvényei

match.groupdict() Egy aszociációs tömbbel tér vissza, amely tartalmazza a találat rész sztringeket.

A groupdict függvény használata:

match.groupdict()

A match object.
A függvény.

groupdict példa:

```
>>> import re
>>> a=abcd efgh
>>> m=re.match(r"(?P<First>\w+) (?P<Second>\w+)")
>>> m.groupdict()
{'First': 'abcd', 'Second': 'efgh'}
```

A match object függvényei

match.(start)

match.(end) A kérdéses rész sztring kezdeti (**start**) és vég (**end**) indexével tér vissza

A match object függvényei

match.(start)

match.(end) A kérdéses rész sztring kezdeti (**start**) és vég (**end**) indexével tér vissza

A **start** és **end** függvény használata:

A match object függvényei

match.(start)

match.(end) A kérdéses rész sztring kezdeti (**start**) és vég (**end**) indexével tér vissza

A **start** és **end** függvény használata:

A kezdeti és a vég index

```
idx=      .      (      )
idx=      .    (      )
```

A match object függvényei

`match.(start)`

`match.(end)` A kérdéses rész sztring kezdeti (**start**) és vég (**end**) indexével tér vissza

A **start** és **end** függvény használata:

A kezdeti és a vég index

A **match object**.

```
idx=match.      (      )
idx=match.    (      )
```

A match object függvényei

`match.(start)`

`match.(end)` A kérdéses rész sztring kezdeti (**start**) és vég (**end**) indexével tér vissza

A **start** és **end** függvény használata:

```
idx=match.start(      )
idx=match.end(      )
```

A kezdeti és a vég index

A **match object**.

A függvények

A match object függvényei

`match.(start)`

`match.(end)` A kérdéses rész sztring kezdeti (**start**) és vég (**end**) indexével tér vissza

A **start** és **end** függvény használata:

```
idx=match.start(group)
idx=match.end(group)
```

A kezdeti és a vég index
A **match object**.
A függvények
A kérdéses csoport, aminek az indexeit kérdezzük.

A match object függvényei

match.(start)

match.(end) A kérdéses rész sztring kezdeti (**start**) és vég (**end**) indexével tér vissza

A **start** és **end** függvény használata:

A kezdeti és a vég index

aminek
k.

start end példa:

```
>>> import re
>>> a=abcxyzdefg
>>> m=re.search("xyz",a)
>>> b=a[:m.start()]+a[m.end():]
>>> print(b)
'abcdefg'
```

A match object függvényei

match.(start)

match.(end) A kérdéses rész sztring kezdeti (**start**) és vég (**end**) indexével tér vissza

A **start** és **end** függvény használata:

A kezdeti és a vég index

start end példa:

```
>>> import re
>>> a=abcxyzdefg
>>> m=re.search("xyz",a)
>>> b=a[:m.start()]+a[m.end():]
>>> print(b)
'abcdefg'
```

aminek

Ezt vennénk ki.

A match object függvényei

match.(start)

match.(end) A kérdéses rész sztring kezdeti (**start**) és vég (**end**) indexével tér vissza

A **start** és **end** függvény használata:

A kezdeti és a vég index

aminek

k.

start end példa:

```
>>> import re
>>> a=abcxyzdefg
>>> m=re.search("xyz",a)
>>> b=a[:m.start()]+a[m.end():]
>>> print(b)
'abcdefg'
```

Eddig tedd be!

A match object függvényei

match.(start)

match.(end) A kérdéses rész sztring kezdeti (**start**) és vég (**end**) indexével tér vissza

A **start** és **end** függvény használata:

A kezdeti és a vég index

aminek
k.

start end példa:

```
>>> import re
>>> a=abcxyzdefg
>>> m=re.search("xyz",a)
>>> b=a[:m.start()]+a[m.end():]
>>> print(b)
'abcdefg'
```

Innen tovább tedd be!

A match object függvényei

match.(start)

match.(end) A kérdéses rész sztring kezdeti (**start**) és vég (**end**) indexével tér vissza

A **start** és **end** függvény használata:

A kezdeti és a vég index

aminek
k.

start end példa:

```
>>> import re
>>> a=abcxyzdefg
>>> m=re.search("xyz",a)
>>> b=a[:m.start()]+a[m.end():]
>>> print(b)
'abcdefg'
```

A match object függvényei

`match.span()` A kérdéses rész sztring kezdeti és vég indexével tér vissza egy tuple-ben.

A match object függvényei

match.span() A kérdéses rész sztring kezdeti és vég indexével tér vissza egy tuple-ben.

A **span** és **end** függvény használata:

A match object függvényei

match.span() A kérdéses rész sztring kezdeti és vég indexével tér vissza egy tuple-ben.

A **span** és **end** függvény használata:

Az eredmény tuple.

```
tpl=      .     (      )
```

A match object függvényei

match.span() A kérdéses rész sztring kezdeti és vég indexével tér vissza egy tuple-ben.

A **span** és **end** függvény használata:

Az eredmény tuple.

A **match object**.

```
tpl=match.      (       )
```

A match object függvényei

match.span() A kérdéses rész sztring kezdeti és vég indexével tér vissza egy tuple-ben.

A **span** és **end** függvény használata:

```
tpl=match.span(      )
```

Az eredmény tuple.
A **match object**.
A függvény.

A match object függvényei

`match.span()` A kérdéses rész sztring kezdeti és vég indexével tér vissza egy tuple-ben.

A **span** és **end** függvény használata:

```
tpl=match.span(group)
```

Az eredmény tuple.
A **match object**.
A függvény.
A kérdéses csoport, aminek az indexeit kérdezzük.

A match object függvényei

match.span() A kérdéses rész sztring kezdeti és vég indexével tér vissza egy tuple-ben.

A **span** és **end** függvény használata:

Az eredmény tuple.

start end példa:

```
>>> import re
>>> a=abcxyzdefg
>>> m=re.search("xyz",a)
>>> tpl=m.span()
>>> print(tpl)
(3, 7)
```

aminek
k.

A match object függvényei

match.span() A kérdéses rész sztring kezdeti és vég indexével tér vissza egy tuple-ben.

A **span** és **end** függvény használata:

Az eredmény tuple.

start end példa:

```
>>> import re
>>> a=abcxyzdefg
>>> m=re.search("xyz",a)
>>> tpl=m.span()
>>> print(tpl)
(3, 7)
```

aminek
k.

Ezt keressük.

A match object függvényei

match.span() A kérdéses rész sztring kezdeti és vég indexével tér vissza egy tuple-ben.

A **span** és **end** függvény használata:

Az eredmény tuple.

start end példa:

```
>>> import re
>>> a=abcxyzdefg
>>> m=re.search("xyz",a)
>>> tpl=m.span()
>>> print(tpl)
(3, 7)
```

aminek
k.

Reguláris kifejezések

Egy karakter keresése. Az '**a**'-t keressük.

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Bárhol:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Bárhol:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Ezt már ismerjük.

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Bárhol:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Ezt már ismerjük.
Ebből a tömbből válogatunk.

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Bárhol:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Ezt már ismerjük.
Ebből a tömbből válogatunk.
A tömbelemek olvasása a `c` változba.

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Bárhol:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Ezt már ismerjük.
Ebből a tömbből válogatunk.
A tömbelemek olvasása a `c` változba.

A reguláris kifejezés kiértékelése.

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Bárhol:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Ezt már ismerjük.
Ebből a tömbből válogatunk.
A tömbelemek olvasása a `c` változba.

A reguláris kifejezés kiértékelése.
Volt-e találat.

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Bárhol:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Ezt már ismerjük.
Ebből a tömbből válogatunk.
A tömbelemek olvasása a `c` változba.

A reguláris kifejezés kiértékelése.
Volt-e találat.
Ha volt, akkor kiírjuk.

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

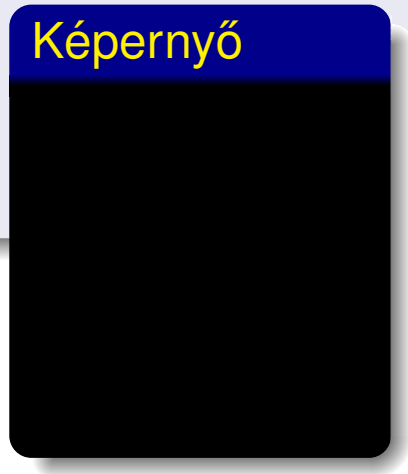
Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő



Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

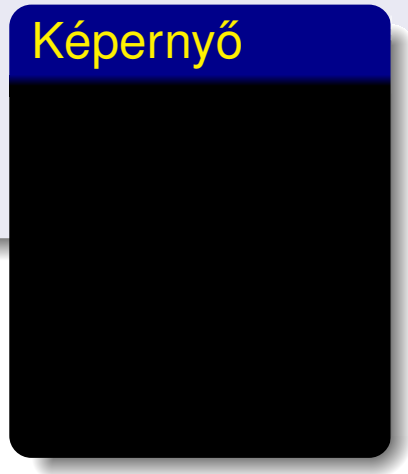
Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő



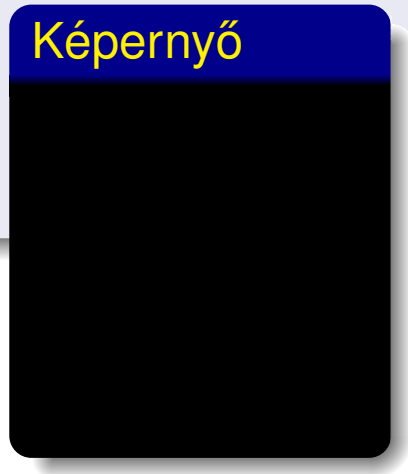
Reguláris kifejezések

Egy karakter keresése. Az '**a**'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő



Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
```

✔

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
```

✔

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
```

✔

Reguláris kifejezések

Egy karakter keresése. Az '**a**'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

✔

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az '**a**'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
```

Reguláris kifejezések

Egy karakter keresése. Az '**a**'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
```

✔

Reguláris kifejezések

Egy karakter keresése. Az '**a**'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
```

Reguláris kifejezések

Egy karakter keresése. Az '**a**'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
bba
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
bba
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
bba
```

Reguláris kifejezések

Egy karakter keresése. Az '**a**'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
bba
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük.

Futtassuk (bárhol):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
bba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Elöl:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Elöl:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Ezt már ismerjük.

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Elöl:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Ezt már ismerjük.
A reguláris kifejezés kiértékelése ^ jelzi, hogy a sztring kezdetén keresünk.

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Elöl:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Ezt már ismerjük.
A reguláris kifejezés kiértékelése ^ jelzi, hogy a sztring kezdetén keresünk.
Ez sem újdonság.

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük elől.

Futtassuk (elöl):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("^a",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aab
aba
abb
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Hátul:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
   m=re.search("a$",c)
   if(m):
      print(c)
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Hátul:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Ezt már ismerjük.

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Hátul:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Ezt már ismerjük.
A reguláris kifejezés kiértékelése $ jelzi, hogy a sztring végén keresünk.

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Hátul:

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Ezt már ismerjük.
A reguláris kifejezés kiértékelése $ jelzi, hogy a sztring végén keresünk.
Ez sem újdonság.

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
   m=re.search("a$",c)
   if(m):
      print(c)
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
bba
```

Reguláris kifejezések

Egy karakter keresése. Az 'a'-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
bba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
bba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
bba
```

Reguláris kifejezések

Egy karakter keresése. Az ’a’-t keressük hátul.

Futtassuk (hátul):

```
#!/usr/bin/python3
import re
a=["aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
for c in a:
    m=re.search("a$",c)
    if(m):
        print(c)
```

Képernyő

```
aaa
aba
baa
bba
```

Karakter osztályok

- egyetlen karakter helyettesítése. A pont helyén bármilyen karakter lehet.

Karakter osztályok

- egyetlen karakter helyettesítése. A pont helyén bármilyen karakter lehet.

```
#!/usr/bin/pyhon3
import re
a=["aaa","aab","aba","abb","baa","bab","bba","bbb"]
for c in a:
   m=re.search("a.",c)
   if(m):
      print(c)
```

Karakter osztályok

- egyetlen karakter helyettesítése. A pont helyén bármilyen karakter lehet.

```
#!/usr/bin/pyhon3
import re
a=["aaa","aab","aba","abb","baa","bab","bba","bbb"]
for c in a:
   m=re.search("a.",c)
   if(m):
      print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
```

Karakter osztályok

. egyetlen karakter helyettesítése. A pont helyén bármilyen karakter lehet.

```
#!/usr/bin/pyhon3
import re
a=["aaa","aab","aba","abb","baa","bab","bba","bbb"]
for c in a:
   m=re.search("a.",c)
   if(m):
      print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
```

[...] felsorolásszerűen megadható a szűrendő karaker halmaz.

Karakter osztályok

. egyetlen karakter helyettesítése. A pont helyén bármilyen karakter lehet.

```
#!/usr/bin/pyhon3
import re
a=["aaa","aab","aba","abb","baa","bab","bba","bbb"]
for c in a:
   m=re.search("a.",c)
   if(m):
      print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
```

[...] felsorolásszerűen megadható a szűrendő karaker halmaz. Megadás módja:

Karakter osztályok

. egyetlen karakter helyettesítése. A pont helyén bármilyen karakter lehet.

```
#!/usr/bin/pyhon3
import re
a=["aaa","aab","aba","abb","baa","bab","bba","bbb"]
for c in a:
   m=re.search("a.",c)
   if(m):
      print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
```

`[...]` felsorolásszerűen megadható a szűrendő karaker halmaz. Megadás módja:

- felsorolás: `[abc]`,

Karakter osztályok

. egyetlen karakter helyettesítése. A pont helyén bármilyen karakter lehet.

```
#!/usr/bin/pyhon3
import re
a=["aaa","aab","aba","abb","baa","bab","bba","bbb"]
for c in a:
   m=re.search("a.",c)
   if(m):
      print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
```

`[...]` felsorolásszerűen megadható a szűrendő karaker halmaz. Megadás módja:

- felsorolás: `[abc]`,
- tartomány: `[a-z]`,

Karakter osztályok

`.` egyetlen karakter helyettesítése. A pont helyén bármilyen karakter lehet.

```
#!/usr/bin/pyhon3
import re
a=["aaa","aab","aba","abb","baa","bab","bba","bbb"]
for c in a:
   m=re.search("a.",c)
   if(m):
      print(c)
```

Képernyő

```
aaa
aab
aba
abb
baa
bab
```

`[...]` felsorolásszerűen megadható a szűrendő karaker halmaz. Megadás módja:

- felsorolás: `[abc]`,
- tartomány: `[a-z]`,
- és ezek kombinációja: `[a-z0-9]`.

Karakter osztályok

`[^...]` a megadott karakterhalmaz kizárása. A megadás módja megegyezik az előző pontban leírttal.

Karakter osztályok

`[^...]` a megadott karakterhalmaz kizárása. A megadás módja megegyezik az előző pontban leírttal.

```
#!/usr/bin/pyhon3
import re
a=["aaa","aab","aba","abb","baa","bab","bba","bbb"]
for c in a:
  m=re.search("a[^a]",c)
  if(m):
    print(c)
```

Karakter osztályok

`[^...]` a megadott karakterhalmaz kizárása. A megadás módja megegyezik az előző pontban leírttal.

```
#!/usr/bin/pyhon3
import re
a=["aaa","aab","aba","abb","baa","bab","bba","bbb"]
for c in a:
   m=re.search("a[^a]",c)
   if(m):
      print(c)
```

Képernyő

```
aab
aba
abb
bab
```

Karakter osztályok

`[^...]` a megadott karakterhalmaz kizárása. A megadás módja megegyezik az előző pontban leírttal.

```
#!/usr/bin/pyhon3
import re
a=["aaa","aab","aba",
for c in a:
   m=re.search("a[^a]
   if(m):
      print(c)
```

Ez azt jelenti, hogy a vizsgált sztringben van-e olyan karakter, amely nem egyezik meg a megadott halmazzal.

Képernyő

```
bab
```

Karakter osztályok

`[^...]` a megadott karakterhalmaz kizárása. A megadás módja megegyezik az előző pontban leírttal.

```
#!/usr/bin/pyhon3
import re
a=["aaa","aab","aba","abb","baa","bab","bba","bbb"]
for c in a:
   m=re.search("a[^a]",c)
   if(m):
      print(c)
```

Képernyő

```
aab
aba
abb
bab
```

Karakter osztályok

- `\d` decimális számjeg. Megegyezik a `[0-9]` halmazzal.
- `\D` nem decimális számjeg. Megegyezik a `[^0-9]` halmazzal.

Karakter osztályok

- `\d` decimális számjeg. Megegyezik a `[0-9]` halmazzal.
- `\D` nem decimális számjeg. Megegyezik a `[^0-9]` halmazzal.
- `\s` szóköz jellegű karakter. Megegyezik a `[ \t\n\r\t\f\v]` halmazzal.
- `\S` nemszóköz jellegű karakter. Megegyezik a `[^ \t\n\r\t\f\v]` halmazzal.

Karakter osztályok

- `\d` decimális számjeg. Megegyezik a `[0-9]` halmazzal.
- `\D` nem decimális számjeg. Megegyezik a `[^0-9]` halmazzal.
- `\s` szóköz jellegű karakter. Megegyezik a `[ \t\n\r\t\f\v]` halmazzal.
- `\S` nemszóköz jellegű karakter. Megegyezik a `[^ \t\n\r\t\f\v]` halmazzal.
- `\w` unicode alfanumerikus karakter. Ha az ASCII flag-et használjuk, akkor megegyezik a `[a-zA-Z0-9_]` halmazzal.
- `\W` nem unicode alfanumerikus karakter. Ha az ASCII flag-et használjuk, akkor megegyezik a `[^a-zA-Z0-9_]` halmazzal.

Karakter osztályok

- `\d` decimális számjeg. Megegyezik a `[0-9]` halmazzal.
- `\D` nem decimális számjeg. Megegyezik a `[^0-9]` halmazzal.
- `\s` szóköz jellegű karakter. Megegyezik a `[ \t\n\r\t\f\v]` halmazzal.
- `\S` nemszóköz jellegű karakter. Megegyezik a `[^ \t\n\r\t\f\v]` halmazzal.
- `\w` unicode alfanumerikus karakter. Ha az ASCII flag-et használjuk, akkor megegyezik a `[a-zA-Z0-9_]` halmazzal.
- `\W` nem unicode alfanumerikus karakter. Ha az ASCII flag-et használjuk, akkor megegyezik a `[^a-zA-Z0-9_]` halmazzal.
- `\A` A sztring elejére illeszkedik.
- `\Z` A sztring végére illeszkedik.

Multiplikátorok

- * 0, vagy több karakterre illeszkedik.

Multiplikátorok

- * 0, vagy több karakterre illeszkedik.

```
#!/usr/bin/pyhon3
import re
a=["bb","bab","baab","baaab"]
for c in a:
   m=re.search("a*",c)
   if(m):
      print(c)
```

Multiplikátorok

- * 0, vagy több karakterre illeszkedik.

```
#!/usr/bin/pyhon3
import re
a=["bb","bab","baab","baaab"]
for c in a:
   m=re.search("a*",c)
   if(m):
      print(c)
```

Képernyő

```
bb
bab
baab
baaab
```

Multiplikátorok

- \+ 1, vagy több karakterre illeszkedik.

Multiplikátorok

- \+ 1, vagy több karakterre illeszkedik.

```
#!/usr/bin/pyhon3
import re
a=["bb","bab","baab","baaab"]
for c in a:
   m=re.search("a+",c)
   if(m):
      print(c)
```

Multiplikátorok

- \+ 1, vagy több karakterre illeszkedik.

```
#!/usr/bin/pyhon3
import re
a=["bb","bab","baab","baaab"]
for c in a:
   m=re.search("a+",c)
   if(m):
      print(c)
```

Képernyő

```
bab
baab
baaab
```

Multiplikátorok

- ? 0, vagy 1 karakterre illeszkedik.

Multiplikátorok

- ? 0, vagy 1 karakterre illeszkedik.

```
#!/usr/bin/pyhon3
import re
a=["bb","bab","baab","baaab"]
for c in a:
   m=re.search("a?",c)
   if(m):
      print(c)
```

Multiplikátorok

- ? 0, vagy 1 karakterre illeszkedik.

```
#!/usr/bin/pyhon3
import re
a=["bb","bab","baab","baaab"]
for c in a:
   m=re.search("a?",c)
   if(m):
      print(c)
```

Képernyő

```
bb
bab
baab
baaab
```

Multiplikátorok

? 0, vagy 1 karakterre illeszkedik.

```
#!/usr/bin/pyhon3
import re
a=["bb","bab","baab","baaab"]
for c in a:
   m=re.search("a?",c)
   if(m):
      print(c)
```

Képernyő

```
bb
bab
baab
baaab
```

Van értelme, még ha nehéz is elhinni!

Multiplikátorok

- ? 0, vagy 1 karakterre illeszkedik.

```
#!/usr/bin/pyhon3
import re
a=["bb","bab","baab","baaab"]
for c in a:
   m=re.search("a?",c)
   if(m):
      print(c)
```

Képernyő

```
bb
bab
baab
baaab
```

Multiplikátorok

`{n}` n db. karakterre illeszkedik.

Multiplikátorok

`{n}` n db. karakterre illeszkedik.

```
#!/usr/bin/pyhon3
import re
a=["bb","bab","baab","baaab"]
for c in a:
   m=re.search("a{2}",c)
   if(m):
      print(c)
```

Multiplikátorok

`{n}` n db. karakterre illeszkedik.

```
#!/usr/bin/pyhon3
import re
a=["bb","bab","baab","baaab"]
for c in a:
   m=re.search("a{2}",c)
   if(m):
      print(c)
```

Képernyő

```
baab
baaab
```

Multiplikátorok

`{n}` n db. karakterre illeszkedik.

```
#!/usr/bin/pyhon3
import re
a=["bb","bab","baab","baaab"]
for c in a:
   m=re.search("a{2}",c)
   if(m):
      print(c)
```

Képernyő

```
baab
baaab
```

A **baaab**-ban is van pontosan 2db. 'a' karakter.
Ennek máshol van jelentősége.

Multiplikátorok

`{n}` n db. karakterre illeszkedik.

```
#!/usr/bin/pyhon3
import re
a=["bb","bab","baab","baaab"]
for c in a:
   m=re.search("a{2}",c)
   if(m):
      print(c)
```

Képernyő

```
baab
baaab
```

Multiplikátorok

`{n,}` legalább n db. karakterre illeszkedik.

Multiplikátorok

- `{n,}` legalább n db. karakterre illeszkedik.
- `{,m}` legfeljebb m db. karakterre illeszkedik.

Multiplikátorok

- `{n,}` legalább n db. karakterre illeszkedik.
- `{,m}` legfeljebb m db. karakterre illeszkedik.
- `{n,m}` legalább n, de legfeljebb m db. karakterre illeszkedik.

Multiplikátorok

- `{n,}` legalább n db. karakterre illeszkedik.
- `{,m}` legfeljebb m db. karakterre illeszkedik.
- `{n,m}` legalább n, de le

Ennek is máshol van jelentősége.
Ezért nincs itt példa.

Multiplikátorok

- `{n,}` legalább n db. karakterre illeszkedik.
- `{,m}` legfeljebb m db. karakterre illeszkedik.
- `{n,m}` legalább n, de legfeljebb m db. karakterre illeszkedik.

Példa 1. (GPS):

GPS rekord szétbontása:

Példa 1. (GPS):

GPS rekord szétbontása:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

Példa 1. (GPS):

GPS rekord szétbontása:

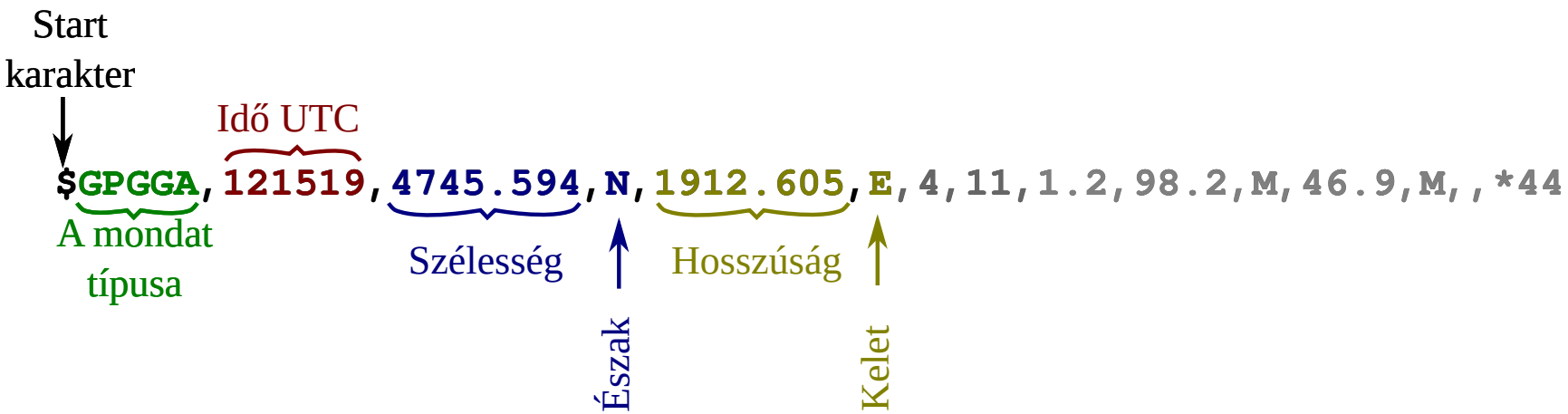
Start karakter

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

A mondat típusa

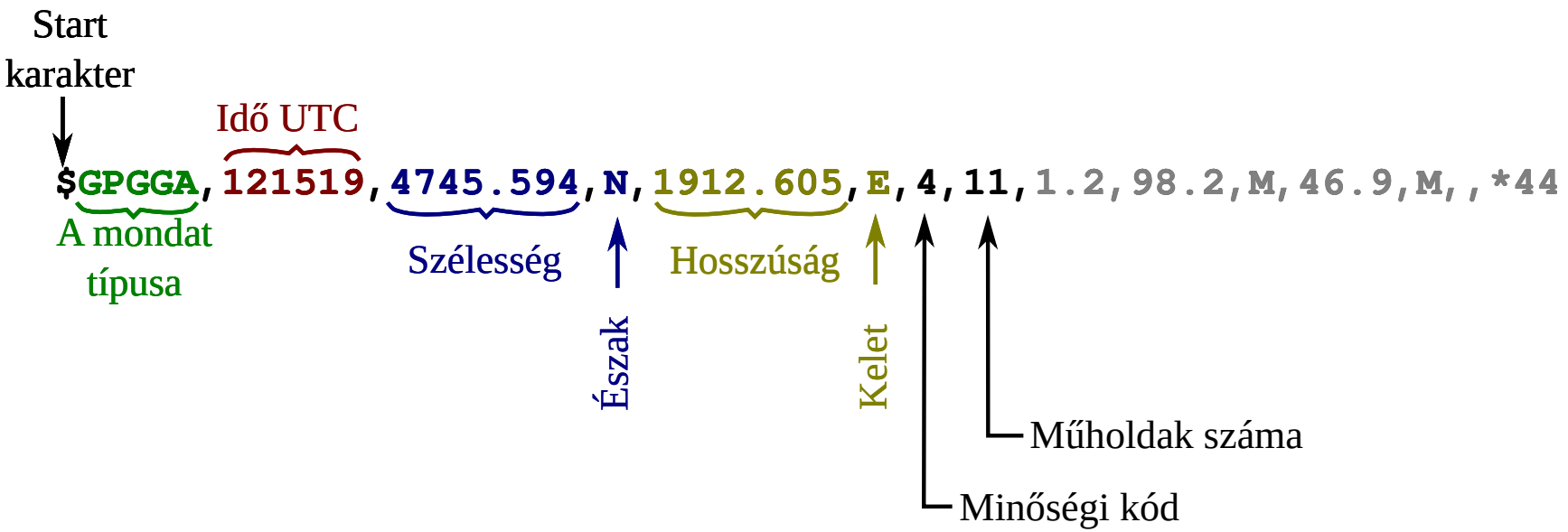
Példa 1. (GPS):

GPS rekord szétbontása:



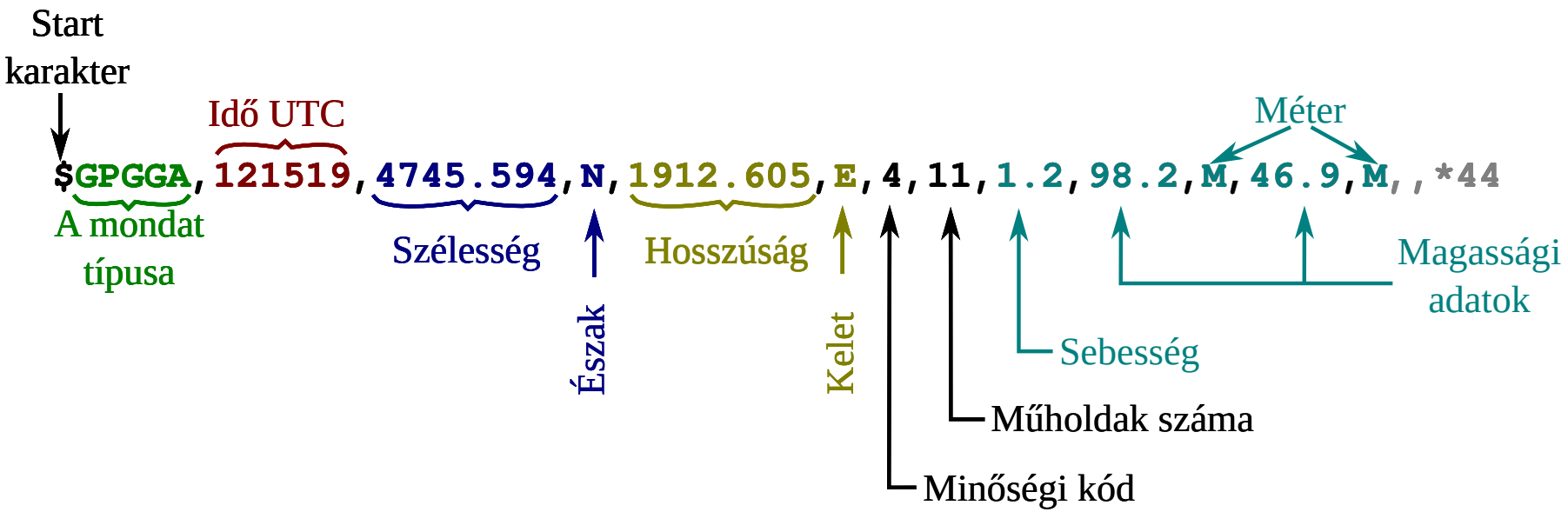
Példa 1. (GPS):

GPS rekord szétbontása:



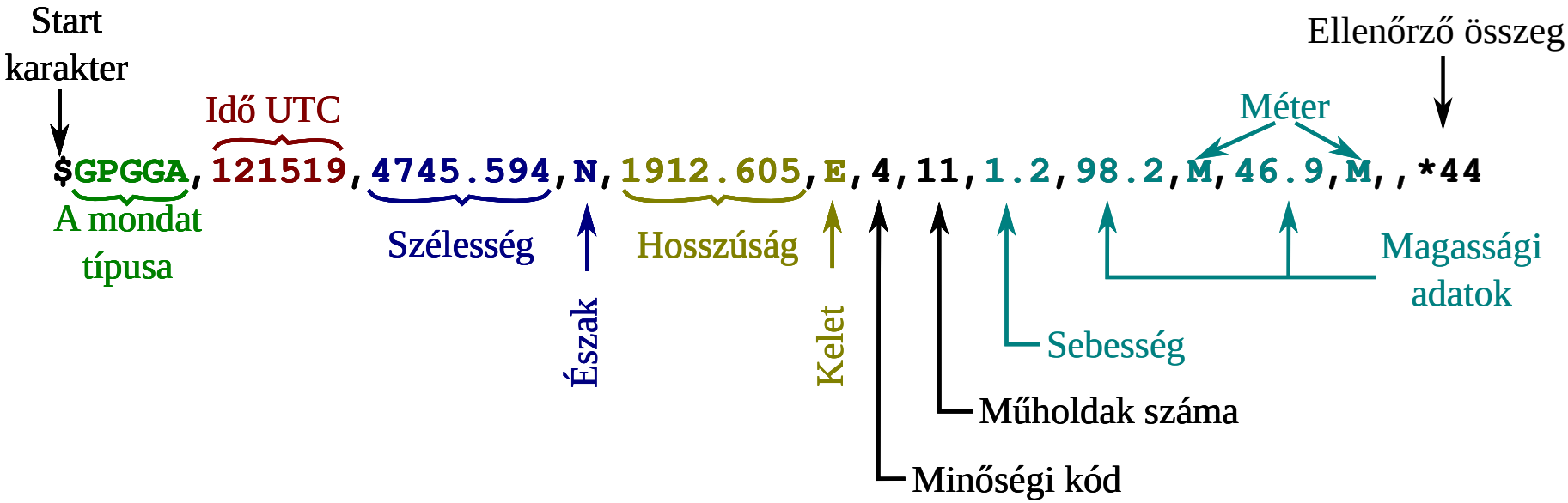
Példa 1. (GPS):

GPS rekord szétbontása:



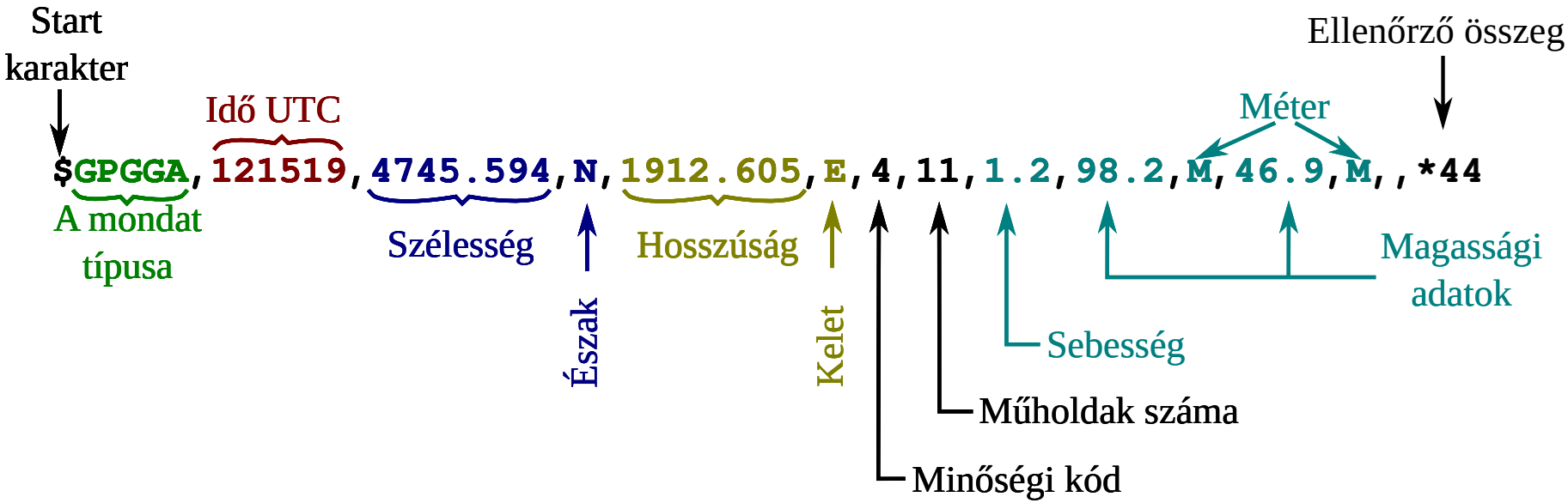
Példa 1. (GPS):

GPS rekord szétbontása:



Példa 1. (GPS):

GPS rekord szétbontása:



Ezt kell szétbontani!

Példa 1. (GPS):

A GPS rekord:

"

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

A reguláris kifejezés:

```
r"                                                    "
```

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

Ez nem kell!

A reguláris kifejezés:

```
r"$\w+,                                                    "
```

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

Ez nem kell!
Az idő (UTC HHMMSS).
Ez kell!

A reguláris kifejezés:

```
r"$\w+,(\d+)                                    "
```

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

Ez nem kell!
Az idő (UTC HHMMSS).
Ez kell!
Egyik vessző sem kell!

A reguláris kifejezés:

```
r"$\w+,(\d+),                                    "
```

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

Ez nem kell!
Az idő (UTC HHMMSS).
Ez kell!
Egyik vessző sem kell!
A szélességi érték kell!

A reguláris kifejezés:

```
r"$\w+,(\d+),(\d+\.\d+)                                    "
```

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

Ez nem kell!
Az idő (UTC HHMMSS).
Ez kell!
Egyik vessző sem kell!
A szélességi érték kell!

A reguláris kifejezés:

```
r"$\w+,(\d+),(\d+\.\d+),                                    "
```

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

Ez nem kell!
Az idő (UTC HHMMSS).
Ez kell!
Egyik vessző sem kell!
A szélességi érték kell!
Északi (N), vagy déli (S) szélesség, kell!

A reguláris kifejezés:

```
r"$\w+,(\d+),(\d+\.\d+),([NS])"
```

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

Ez nem kell!
Az idő (UTC HHMMSS).
Ez kell!
Egyik vessző sem kell!
A szélességi érték kell!
Északi (N), vagy déli (S) szélesség, kell!

A reguláris kifejezés:

```
r"$\w+,(\d+),(\d+\.\d+),([NS]),                    "
```

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

A reguláris kifejezés:

```
r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+)                    "
```

Ez nem kell!
Az idő (UTC HHMMSS).
Ez kell!
Egyik vessző sem kell!
A szélességi érték kell!
Északi (N), vagy déli (S) szélesség, kell!
A hosszúsági érték kell!

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

A reguláris kifejezés:

```
r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+),                    "
```

Ez nem kell!
Az idő (UTC HHMMSS).
Ez kell!
Egyik vessző sem kell!
A szélességi érték kell!
Északi (N), vagy déli (S) szélesség, kell!
A hosszúsági érték kell!

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

A reguláris kifejezés:

```
r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+),([EW])"
```

Ez nem kell!
Az idő (UTC HHMMSS).
Ez kell!
Egyik vessző sem kell!
A szélességi érték kell!
Északi (N), vagy déli (S) szélesség, kell!
A hosszúsági érték kell!
Keleti (E), vagy nyugati (S) hosszúság, kell!

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

Ez nem kell!
Az idő (UTC HHMMSS).
Ez kell!
Egyik vessző sem kell!
A szélességi érték kell!
Északi (N), vagy déli (S) szélesség, kell!
A hosszúsági érték kell!
Keleti (E), vagy nyugati (S) hosszúság, kell!
A többi most nem kell!

A reguláris kifejezés:

```
r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+),([EW])"
```

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

A reguláris kifejezés:

```
r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+),([EW])"
```

Ez nem kell!
Az idő (UTC HHMMSS).
Ez kell!
Egyik vessző sem kell!
A szélességi érték kell!
Északi (N), vagy déli (S) szélesség, kell!
A hosszúsági érték kell!
Keleti (E), vagy nyugati (S) hosszúság, kell!
A többi most nem kell!

Példa 1. (GPS):

A GPS rekord:

```
$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44
```

A reguláris kifejezés:

```
r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+),([EW])"
```

Ez nem kell!
Az idő (UTC HHMMSS).
Ez kell!
Egyik vessző sem kell!
A szélességi érték kell!
Északi (N), vagy déli (S) szélesség, kell!
A hosszúsági érték kell!
Keleti (E), vagy nyugati (S) hosszúság, kell!
A többi most nem kell!

Kész!!!!

Példa 1. (GPS):

A program:

Példa 1. (GPS):

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
m=re.match(r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+),([EW])",g)
print(m.group(1))
print(m.group(2))
print(m.group(3))
print(m.group(4))
print(m.group(5))
```

Példa 1. (GPS):

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
m=re.match(r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+),([EW])",g)
print(m.group(1))
print(m.group(2))
print(m.group(3))
print(m.group(4))
print(m.group(5))
```

Képernyő

```
121519
```

Példa 1. (GPS):

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
m=re.match(r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+),([EW])",g)
print(m.group(1))
print(m.group(2))
print(m.group(3))
print(m.group(4))
print(m.group(5))
```

Képernyő

```
121519
4745.594
```

Példa 1. (GPS):

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
m=re.match(r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+),([EW])",g)
print(m.group(1))
print(m.group(2))
print(m.group(3))
print(m.group(4))
print(m.group(5))
```

Képernyő

```
121519
4745.594
N
```

Példa 1. (GPS):

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
m=re.match(r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+),([EW])",g)
print(m.group(1))
print(m.group(2))
print(m.group(3))
print(m.group(4))
print(m.group(5))
```

Képernyő

```
121519
4745.594
N
1912.605
```

Példa 1. (GPS):

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
m=re.match(r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+),([EW])",g)
print(m.group(1))
print(m.group(2))
print(m.group(3))
print(m.group(4))
print(m.group(5))
```

Képernyő

```
121519
4745.594
N
1912.605
E
```

Példa 1. (GPS):

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
m=re.match(r"$\w+,(\d+),(\d+\.\d+),([NS]),(\d+\.\d+),([EW])",g)
print(m.group(1))
print(m.group(2))
print(m.group(3))
print(m.group(4))
print(m.group(5))
```

Képernyő

```
121519
4745.594
N
1912.605
E
```

Példa 2 (GPS)

Egy másik megoldás a **`split`** függvénnyel:

Példa 2 (GPS)

Egy másik megoldás a **split** függvénnyel:

A GPS rekord:

```
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
```

Példa 2 (GPS)

Egy másik megoldás a **split** függvénnyel:

A GPS rekord:

```
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
```

A mezőket vessző határolja, ez jó a ”szeletelésre”.

Példa 2 (GPS)

Egy másik megoldás a **split** függvénnyel:

A GPS rekord:

```
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
```

A mezőket vessző határolja, ez jó a ”szeletelésre”.

Példa 2 (GPS)

Egy másik megoldás a **split** függvénnyel:

A GPS rekord:

```
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
```

A mezőket vessző határolja, ez jó a ”szeletelésre”.

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
m=re.split(",",g)
print(m[1])
print(m[2])
print(m[3])
print(m[4])
print(m[5])
```

Példa 2 (GPS)

Egy másik megoldás a **split** függvénnyel:

A GPS rekord:

```
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
```

A mezőket vessző határolja, ez jó a ”szeletelésre”.

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,4
m=re.split(",",g)
print(m[1])
print(m[2])
print(m[3])
print(m[4])
print(m[5])
```

Képernyő

Példa 2 (GPS)

Egy másik megoldás a **split** függvénnyel:

A GPS rekord:

```
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
```

A mezőket vessző határolja, ez jó a ”szeletelésre”.

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,4
m=re.split(",",g)
print(m[1])
print(m[2])
print(m[3])
print(m[4])
print(m[5])
```

Képernyő

```
121519
```

Példa 2 (GPS)

Egy másik megoldás a **split** függvénnyel:

A GPS rekord:

```
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
```

A mezőket vessző határolja, ez jó a ”szeletelésre”.

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,4
m=re.split(",",g)
print(m[1])
print(m[2])
print(m[3])
print(m[4])
print(m[5])
```

Képernyő

```
121519
4745.594
```

Példa 2 (GPS)

Egy másik megoldás a **split** függvénnyel:

A GPS rekord:

```
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
```

A mezőket vessző határolja, ez jó a ”szeletelésre”.

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,4
m=re.split(",",g)
print(m[1])
print(m[2])
print(m[3])
print(m[4])
print(m[5])
```

Képernyő

```
121519
4745.594
N
```

Példa 2 (GPS)

Egy másik megoldás a **split** függvénnyel:

A GPS rekord:

```
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
```

A mezőket vessző határolja, ez jó a ”szeletelésre”.

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,4
m=re.split(",",g)
print(m[1])
print(m[2])
print(m[3])
print(m[4])
print(m[5])
```

Képernyő

```
121519
4745.594
N
1912.605
```

Példa 2 (GPS)

Egy másik megoldás a **split** függvénnyel:

A GPS rekord:

```
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
```

A mezőket vessző határolja, ez jó a ”szeletelésre”.

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,4
m=re.split(",",g)
print(m[1])
print(m[2])
print(m[3])
print(m[4])
print(m[5])
```

Képernyő

```
121519
4745.594
N
1912.605
E
```

Példa 2 (GPS)

Egy másik megoldás a **split** függvénnyel:

A GPS rekord:

```
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,46.9,M,,*44"
```

A mezőket vessző határolja, ez jó a ”szeletelésre”.

A program:

```
#!/usr/bin/python3
import re
g="$GPGGA,121519,4745.594,N,1912.605,E,4,11,1.2,98.2,M,4
m=re.split(",",g)
print(m[1])
print(m[2])
print(m[3])
print(m[4])
print(m[5])
```

Képernyő

```
121519
4745.594
N
1912.605
E
```