<-- vissza

Tartalomjegyzék

- 2.12 email
- 2.13 smtplib
- 2.14 poplib
- 2.15 threading
- 2.16 zipfile
- 2.17 logging
- 2.18 weakref
- 2.19 gc
- 2.20 array
- 2.21 collections
- 2.22 bisect
- 2.23 heapsq
- 2.24 ...

kivételek

Az el?z? órán tanultuk hogyan tudunk kivételeket kezelni, most megnézzük a f?bb kivételeket és azok hierarchiáját.

BaseException - az ?se minden kivételnek, ajánlatos nem ?t örökíteni. Létezik az **args** változója ami egy tuple és ebben lehet paramétereket tárolni.

Exception - minden nem rendszerszint? kivétel ett?l származik le. A fejleszt? által definiált kivételek ?se.

ArithmeticError - alap osztálya az aritmetikai kivételeknek mint pl.: **OverflowError** - túlcsordulás, **ZeroDivisionError** - nullával való osztás, **FloatingPointError**.

```
import math
1-math.exp(-4*1000000*-0.0641515994108)
...
OverflowError: math range error
```

LookupError - alap osztálya az lista címzési kivételeknek mint a **IndexError** - indexelési hiba, **KeyError** - címzési hiba.

```
d = {}
d['alma']
...
KeyError: 'alma'
```

AssertionError - Amikor egy assert kifejezés elbukik.

AttributeError - Nemlétez? attribútum lekérése vagy érték adásnál.

EOFError - amikor az file végét eléri egy beépített fv.

IOError - input, output operátorok elbukásakor pl.: a file nem található vagy megtelik a disk

ImportError - keletkezik, amikor a modul vagy azon belüli elem nem található

TabError - az IndentationError (nem megfelel? a behúzás) gyereke, akkor dobódik, amikor a behúzás nem konzisztens

StopIteration - amikor az iterátoron már nem tudunk tovább haladni

NameError - amikor lokálisan vagy globálisan a név nem található

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |    |    +-- FloatingPointError
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
      |    +-- EnvironmentError
      |    |    +-- IOError
      |    |    +-- OSError
      |    |         +-- WindowsError (Windows)
      |    |         +-- VMSError (VMS)
      |    +-- EOFError
      |    +-- ImportError
      |    +-- LookupError
      |    |    +-- IndexError
      |    |    +-- KeyError
      |    +-- MemoryError
      |    +-- NameError
      |    |    +-- UnboundLocalError
      |    +-- ReferenceError
      |    +-- RuntimeError
      |    |    +-- NotImplementedError
      |    +-- SyntaxError
      |    |    +-- IndentationError
      |    |         +-- TabError
      |    +-- SystemError
      |    +-- TypeError
      |    +-- ValueError
      |         +-- UnicodeError
      |              +-- UnicodeDecodeError
      |              +-- UnicodeEncodeError
      |              +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
```

kivételek felsorolása és hierarchiája

speciális kivétel a warning

Ha valamilyen formában jelezni akarjuk, hogy a programban olyan részek vannak amiket nem szívesen látunk, akkor a warning - val olyan jelzéstkuldhetünk, ami a programot nem terminálja ha nincs lekezelve, de valamilzen formában megjelenik. A megjelenést, kezelést tudjuk módosítani filter definiálásával.

Warning dobása.

```
import warnings as ws
wa.warn("deprecated", DeprecationWarning)
```

Filterezés

```
import warnings

def fxn():
    warnings.warn("deprecated", DeprecationWarning)

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    fxn()
```

részletes leírás

kivételkezelési technikák

bevezetés alapcsomagokba

A python egy széles körben használható nyelv, és ezt annak köszönheti, hogy rengeteg területre szolgáltat, vagy létezik 3rd party kiegészít? csomagjai. Itt most az alap csomagok egy kis részét fogjuk áttekinteni.

továbbiakban

regexp (re)

Célja hogy reguláris kifejezések segítségével szövegeken tudjatok keresni egy adott mintára és azok elemeit felhasználni.

reguláris kifejezés

a reguláris kifejezés egy karaktersorozat amiben speciális karakterek segítségével mintákat adhatunk meg. a 'alma' egy olyan reg. kif. ami pontosan az alma karaktersorozatra fog egyezni.

speciális karakterek

'.' - a pont a bármilyen karaktert jelöli, kivéve a \n

'^' - azt jelzi, hogy az illesztés során ott van a karakter sorozat eleje

'$' - a kar. sor. végét jelöli

számosság

'*' - jelentése 0 vagy több (mohó)

'+' - 1 vagy több (mohó)

'?' - 0 vagy 1

```
*?, +?, ?? – ha az el?z?k mögé a ? tesszük akkor a mohó tulajdonságot vezérelhetjük
```

{m} - az ismétlés pontos számát adhatjuk meg

{m,n} - m és n közötti ismétlést

{m,n}? - hasonó...

escape karakter

'\'

halmaz

[] - ezen belül felsorolt karakter valamelyike lehet az erre illeszked? karakter. Lehet tartományokat megadni mint a kisbet?k: [a-z], stb [a-zA-Z0-9]. A speciális karakterek elvesztik értelmüket a halmazon belül. Karakterosztályok is használhatóak lásd kés?bb. Ha az els? karakter a '^' akkor a jelentése az lesz, hogy minden olyan karakter ami nincs a halmazban (ha más pozícióban szerepel akkor nincs különleges szerepe). Zárójelek: [()[\]{}] ua. []()[{}]

csoportosítás

'|' - A|B jelentése A vagy B, ahol A és B tetsz?leges reguláris kif.

(...) - csoport jelzése. a Csoport újrahasználása a '\' és a csoport számával lehet: \1

(?...) -

Karakter osztályok

\A - a karakterlánc kezdete

\b - üres karakter a karakterláncon kívül

\B - üres karakter a karakterláncon belül

\d - [0-9]

\D - [^0-9]

\s - [\t\n\r\f\v]

\S - [^ \t\n\r\f\v]

\w - [a-zA-Z0-9_]

\W - [^a-zA-Z0-9_]

\Z - a karakterlánc vége

felhasználása

```
prog = re.compile(pattern)
result = prog.match(string)
```

ami megegyezik

```
result = re.match(pattern, string, flag = re.I)
```

```
re.search(pattern, string, flags=0)
```

végigfut és egyezéseket keres

re.match(pattern, string, flags=0) csak a kezdeti egyezéseket adja vissza

az eredmény

```
>>> t=re.match(r'a{0,3}b','abaaab')
>>> t
<_sre.SRE_Match object at 0xb7235d78>
>>> dir(t)
['__copy__', '__deepcopy__', 'end', 'expand', 'group', 'groupdict', 'groups', 'span', 'start']
```

Jelz?k használata:

```
result = re.match(pattern, string, flag = re.I)
```

jelentése, hogy a nagy és kisbet?ket ne különböztesse meg

glob

Az aktuális könyvtárban keresi azokat mikre illik a keresési feltétel.

```
glob.glob(pathname)
```

```
glob.iglob(pathname) # generátorral tér vissza
```

pathname - lehet relatív, és abszolút elérési útvonal, ami tartalmazhat reguláris kifejezésbeli elemeket...

math

Matematikai funkciók tárháza... (Ha komplex számokkal akarunk dolgozni, akkor a cmath modult használjuk) pl.: math.factorial(x), math.fabs(x)

konstansok

```
math.e
math.pi
```

transzformációk

```
math.degrees(x) # radiánt alakít fokká
```

```
math.radians(x) # fordítva
```

szögfugvények

```
math.acos(x)
math.asin(x)
math.atan(x)
math.cos(x)
math.sin(x)
math.tan(x)
```

kerekites

```
math.ceil(x)
math.floor(x)
math.trunc(x)
```

hatványozás, logaritmus

```
math.log(x[, base])
math.exp(x)
math.pow(x, y)
math.sqrt(x)
```

random

Random generálással kapcsolatos elemek tárháza...

elemei

```
random.seed([x])
# a generátor alapját határozhatjuk meg, ha nem adunk meg semmit akkor a aktuális id?t veszi alapu
# meg a fv. a generátor mindig ua. a sorozatot adja...

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

# range(start, stop, step) elemekb?l választ random. (choice(range(start, stop, step)))

random.randint(a, b)
# a <= N <= b, N-el tér vissza

random.choice(seq)
# a listából választ egyet. IndexError ha nincs elem a paraméterként átadott listában.

random.random()
# lebeg?pontos számmal tér vissza a [0.0, 1.0) - ból
random.uniform(a, b)
# lebeg?pontos számmal tér vissza (a <= N <= b ha a <= b és b <= N <= a ha b < a)
```

urllib2

A modul által kínált függvények és osztályok segítségével képesek vagyunk az interneten található anyagokat kezelni. azonosítás, átirányítás, sütik, stb.

```
>>> import urllib2 as ul
>>> f = ul.urlopen('http://www.python.org')
>>> for i in f:
...   print i
...
```

timeit

Segítségével kis kódrészek lefutását mérhetjük meg.

```
def test():
    """Stupid test function"""
    L = []
    for i in range(100):
        L.append(i)

if __name__ == '__main__':
    from timeit import Timer
    t = Timer("test()", "from __main__ import test")
    print t.timeit()
```

init

A Timer osztály létrehozásával tudjuk az adott kódrészletet megadni, és a kezdetbe állítási kódot is.

```
timeit.Timer([stmt='pass'[, setup='pass'[, timer=<timer function>]]])
```

test

Maga a teszt a timeit meghívásával fut le. A setup code egyszer fut le, a stmt kod mérése meg a number által def. szor fog lefutni és a kapott értékeket átlagolja.

```
Timer.timeit([number=1000000])
```

tudjuk megadott számban ismételni a timeit lefutását a repeat-tal

```
Timer.repeat([repeat=3[, number=1000000]])
```

doctest

A doctest modul a docstring-ben leírt m?ködést képes értelmezni és azokat ellen?rizni.

Az alap esetek amire használható:

- docstring aktualitását ellen?rizni hogy a leírt példák úgy m?ködnek ahogy ott vannak.
- regressziós tesztekre
- hogyan dokumentációk készítésére

unittest

nose over unittest

datetime

xml

json

email

smtplib

poplib

threading

zipfile

logging

weakref

gc

array

collections

bisect

heapsq

...