4. Előadás

Python kurzus

Tárgyfelelős:
Dr Tejfel Máté

Előadó:
Dr. Király Roland

4. Előadás tematikája

Függvények és hibakezelés

1. Ellenőrző teszt
2. Hibakezelés, try - except blokkok, kivételek
3. Függvények: paraméter átadás, pozicionális és kulcsszó-argumentumok
4. Anonim és lambda függvények
5. Beépített függvények és modulok

1. Ellenőrző teszt – Canvas

- Listaműveletek
- String műveletek
- Ciklusok
- Függvény definíció, paraméterátadás

2. Hibakezelés

Hibák az adatokban és hibák a kódban – a tesztelés kötelező

- **Adat hibák:**
  helyes kód, hibás adatbevitel

```
x = int(input('Kérek egy egész számot: '))
```

Kérek egy egész számot: 12o
ValueError: invalid literal for int(): '12o'

- **Kódolási hibák:** hibakeresés kódelemzéssel, debugger használatával, AI eszközökkel

```
for i in range(5):
    if i % 2 != 0:
        print(i, 'páros')
    else:
        print(i,'páratlan')
```

0 páratlan
1 páros
2 páratlan
3 páros
4 páratlan

```
Sugár = 10
terület =  sugár**2*3.14
```

NameError: name 'sugár' is not defined.

```
x = 2.
y = '3'
print(x + y)
```

TypeError: unsupported operand type(s) for +: 'float' and 'str'

```
import math
x = float(input("Kérek egy számot: "))
print(x, "négyzetgyöke", math.sqrt(x))
```

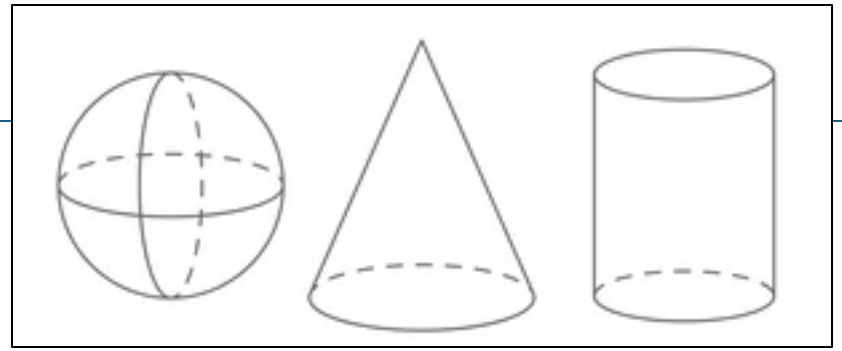
Kérek egy számot: -1
ValueError: math domain error

```
lista = []
a = lista[-1]
```

IndexError: list index out of range

```
from math import pi          # use the value of pi defined in the math library
def main():
    radius = input("Enter the radius: "))
    height = input("Enter the height: "))
    conevol = 1/3*pi*radius**2*height          # cone volume
    cylvol = pi*r**2*height                    # cylinder volume
    spherevol = 4/3*Pi*radius**3               # sphere volume
    print ('The volume of a cone: ',conevol)
    print ("The volume of a cylinder: ",cylvol)
    print ("The volume of a sphere: ",spherevol)
main()
```

Hibaüzenetek!



Kivételkezelés: try – except blokkok

- Hiba esetén: 1. leáll a program 2. egy kivétel generálódik
- Hogyan kezeljük a kivételeket: try – except utasítás

```
try:
    x = int(input("Kérek egy számot: "))
    y = 1 / x
    print(y)
except ZeroDivisionError:
    print("Nullával nem lehet osztani!")
except ValueError:
    print("Egy számot adjon meg!")
except:
    print("Valami más hiba történt!")
print("vége")
```

Kérek egy számot: **2**
0.5
vége

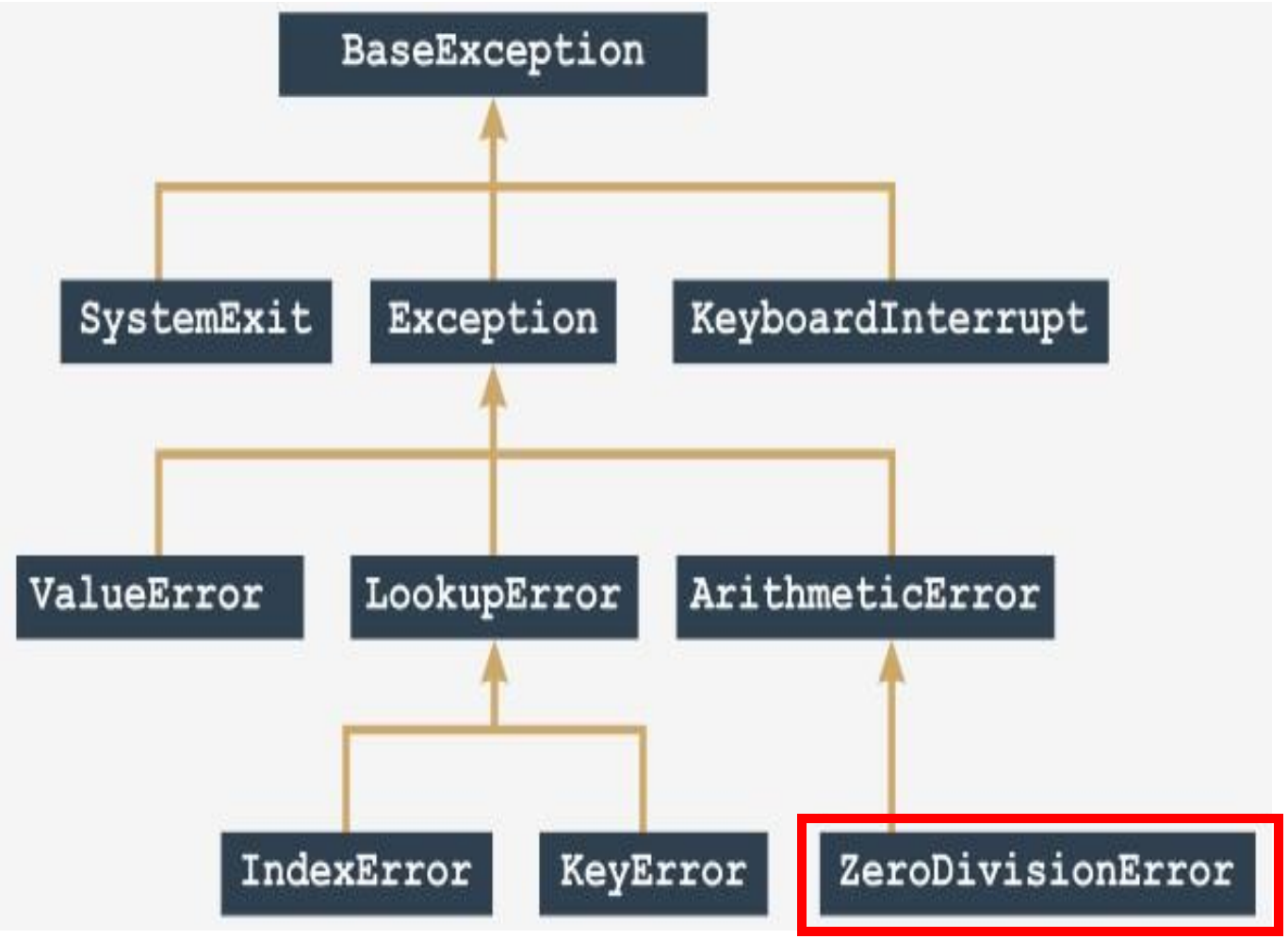
Kérek egy számot: **0**
Nullával nem lehet osztani!
vége

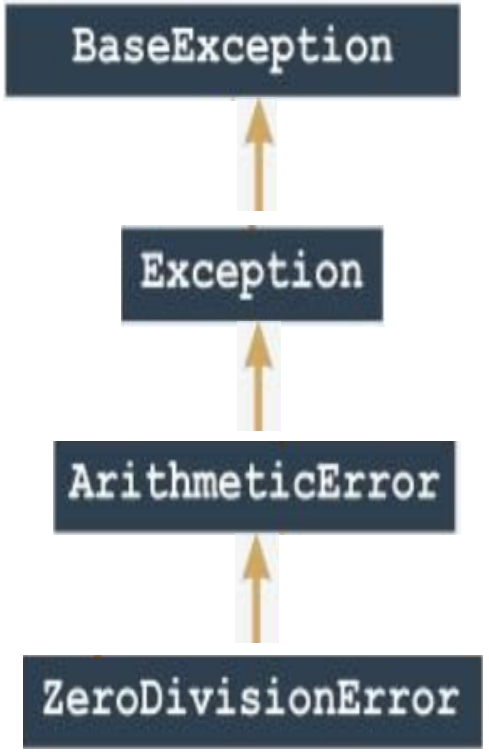
Kérek egy számot: **a**
Egy számot adjon meg!
vége

Kérek egy számot: Valami más hiba történt!
vége

Kivételek - Exceptions

A Python3 63 beépített kivételt definiál, amelyek egy **fa alakú hierarchiát** alkotnak:





Kivételek kiváltása: a raise utasítás

A **raise** utasítás az adott **exception** kivételt úgy fogja kiváltani, mintha az a szokásos (természetes) módon keletkezett volna:

Hiba eset felmerülésekor:

```
def próba(n):
    try:
        return 1 / n
    except ArithmeticError:
        print("Aritmetikai probléma!")
    return None
próba(0)            #a hiba kiváltása
print("vége")
```

```
Aritmetikai probléma!
vége
```

A **raise** alkalmazásával:

```
def próba(n):
    raise ZeroDivisionError
try:
    próba(2)        #nem okoz hibát
except ArithmeticError:
    print("Mi történt? Egy hiba?")
print("vége")
```

```
Mi történt? Egy hiba?
vége
```

A raise utasítás

```
def próba(n):
   try:
      return n / 0
   except:
      print("Megint egy hiba!")
      raise

try:
   próba(2)
except ZeroDivisionError:
   print("Igen, ez egy aritmetikai hiba.")

print("vége")
```

```
Megint egy hiba!
Igen, ez egy aritmetikai hiba.
vége
```

A **raise** nélkül:

```
Megint egy hiba!
vége
```

Beépített kivételek

- **ArithmeticError**
  Aritmetikai műveletek által okozott kivételek, mint a nullával osztás vagy egy argumentum érvénytelen tartománya.
- **AssertionError**
  Az assert utasítás sikertelen.
- **AttributeError**
  Helytelen attribútumra való hivatkozás vagy értékadás.
- **BaseException**
  A legáltalánosabb kivétel, az **except** és **except BaseException** kifejezések egyenértékűek.
- **IndexError**
  Nem található egy index a sorozatban.
- **KeyError**
  Nem létező dictionary kulcsra hivatkozunk.

- **KeyboardInterrupt**
  A felhasználó megszakítja a program végrehajtását
- **NameError**
  Nem meghatározott lokális vagy globális név keresésekor.
- **SyntaxError**
  Az elemző szintaktikai hibát talált.
- **TypeError**
  Beépített operátornak vagy függvénynek helytelen típust adunk át.
- **ValueError**
  Argumentum hiba esetén, melyet a TypeError nem tartalmaz, ha argumentumokként érvénytelen adatokat adtunk meg.
- **ZeroDivisionError**
  Osztásnál vagy maradék (modulo) műveletnél 0 második argumentum esetében.

```
def próba(n):
    try:
        return n / 0
    except Exception as error:
        print("Megint egy hiba!")
        print(error)
```

Megint egy hiba!
division by zero

```
import math
try:
    x = math.sqrt(-1)
    print(x)
except Exception as error:
    print(error)
```

math domain error

```
lista = []
try:
    a = lista[-1]
except Exception as error:
    print(error)
```

list index out of range

```
try:
    print("1" + 1)
except Exception as error:
    print(error)
```

can only concatenate str (not "int") to str

```
def próba(a, b=2, c):
    print(a, b, c)
```

SyntaxError: non-default argument follows default argument

Az assert kifejezés

```
import math
x = float(input("Kérek egy számot: "))
assert x >= 0.0
x = math.sqrt(x)
print(x)
```

```
Kérek egy számot: -1
Traceback (most recent call last):
  File "4ea_proba.py", line 3, in <module>
    assert x >= 0.0
AssertionError
```

```
import math
while True:
    x = float(input("Kérek egy számot: "))
    try:
        assert x >= 0.0
        break
    except AssertionError:
        print("A szám negatív!")
x = math.sqrt(x)
print(x)
```

```
Kérek egy számot: 4
2.0
```

```
Kérek egy számot: -1
A szám negatív!
Kérek egy számot: -2
A szám negatív!
Kérek egy számot: 0
0.0
```

A **yield** speciális kulcsszó: **generátor függvények**ben használjuk

- A normál függvényeknél a **return** egyszeri értéket ad vissza és megszakítja a függvény futását
- A **yield** visszaad egy értéket és nem fejezi be a függvény végrehajtását

```
def my_generator():
    print("Első lépés")
    yield 1
    print("Második lépés")
    yield 2
    print("Harmadik lépés")
    yield 3

gen = my_generator()
print(next(gen))
print(next(gen))
print(next(gen))
```

```
Első lépés
1
Második lépés
2
Harmadik lépés
3
```

```
#2 hatványok kiírása
def kettő_hatvány(n):
    hatv = 1
    for i in range(n):
        yield hatv
        hatv *= 2
for elem in kettő_hatvány(8):
    print(elem)
```

```
1
2
4
8
16
32
64
128
```

3. Függvények: paraméter átadás

Függvény meghívásakor az aktuális paramétert, ha az változó, értékadásban felelteti meg a formális paraméternek, mint lokális változónak.

```
def szorzás(érték, hányszor):
    érték *= hányszor
    print(érték, hányszor)
    print(szám, szorzó)

szám, szorzó = 5, 2
print(szorzás(szám, szorzó))
print(érték, hányszor)
```

```
10 2
5 2
None
NameError: name 'érték' is not defined
```

```
def add_citrom(list):
    list.append("Citrom")
    print("Belső lista:",list)
gyümölcs = ['Alma', 'Körte', 'Narancs']
add_citrom(gyümölcs)
print("Eredeti lista:", gyümölcs)
```

```
Belső lista: ['Alma', 'Körte', 'Narancs', 'Citrom’]
Eredeti lista: ['Alma', 'Körte', 'Narancs', 'Citrom']
```

```
def add_citrom(list):
    list = ["Citrom"]
    print("Belső lista :",list)
gyümölcs = ['Alma', 'Körte', 'Narancs']
add_citrom(gyümölcs)
print("Eredeti lista:", gyümölcs)
```

```
Belső lista : ['Citrom']
Eredeti lista: ['Alma', 'Körte', 'Narancs']
```

- ***args – pozicionális argumentumok**
  - A * jelöli azt, hogy a függvény **tetszőleges számú** pozicionális argumentumot fogadhat el.
  - Az argumentumok **tuple**-ként kerülnek tárolásra a függvényen belül.

- ****kwargs - kulcsszó-argumentumok (keyword arguments)**
  - **Változó számú** kulcsszó-argumentumot adhatunk át egy függvénynek.
  - A kulcsszó-argumentumok név-érték párokként kerülnek átadásra.
  - A **kwargs a kulcsszó-argumentumokat **szótárként (dict)** tárolja, ahol a kulcsok az argumentumok nevei, az értékek pedig az ezekhez tartozó értékek.

Pozicionális (*args) és kulcsszó-argumentumok (**kwargs)

```
def összeg(*args):
    return sum(args)

print(összeg(1, 2, 3))
print(összeg(5, 10, 15, 20))
```

```
6
50
```

```
def profil(**kwargs):
    for kulcs, érték in kwargs.items():
        print(f"{kulcs.capitalize()}: {érték}")

profil(név="Anna", kor=20, város="Sopron")
```

```
Név: Anna
Kor: 30
Város: Sopron
```

```
def mindent_kezel(név, *args, **kwargs):
    print(f"Név: {név}")
    print("Pozicionális argumentumok:", args)
    print("Kulcsszó-argumentumok:", kwargs)
mindent_kezel("Anna", 10, 20, 30, város="Sopron", kor=20)
```

```
Név: Anna
Pozicionális argumentumok: (10, 20, 30)
Kulcsszó-argumentumok: {'város': 'Sopron', 'kor': 20}
```

4. Anonim és lambda függvények

lambda argument1, argument2, ... : kifejezés

- A **lambda** kulcsszóval egy **anonim** (névtelen) függvényt definiálunk
- Az argumentumok a függvény bemeneti paraméterei
- A kettőspont utáni kifejezés a függvény visszatérési értéke lesz
- A lambda függvényekben nincs return, hanem automatikusan visszatérnek a kifejezés kiértékelt eredményével

```
összeadás = lambda a, b: a + b
print(összeadás(2, 5))
```

7

```
hatvány = lambda x, k: x **k
print(hatvány(2, 5))
```

32

```
print((lambda a, b: a + b)(2,5))
```

7

Megj.: A () szükséges a lambda fv-hez

lambda argument1, argument2, ... : kifejezés

```
print((lambda a: a + 6)(2))          #eltolás +6-tal
print((lambda a, b: a - b)(1,5))     #kivonás
print((lambda a, b: a * b)(3,4))     #szorzás
print((lambda a, b: a ** b)(5,3))    #hatványozás
```

```
8
-4
12
125
```

```
print((lambda a,b: (a**2 + b**2)**0.5)(5,12))  #Pitagorasz-tétel
```

```
13.0
```

```
def fv_érték(args, polinom):
   for x in args:
      print('g(', x,')=', polinom(x), sep='')

fv_érték([x for x in range(-2,4)], lambda x: 3 * x**2 - 6 * x + 3)
```

```
g(-2)=27
g(-1)=12
g(0)=3
g(1)=0
g(2)=3
g(3)=12
```

A lambda alkalmazása magasabb rendű függvényekben

A **map** függvény: egy függvényt alkalmaz minden elemre egy iterálható objektumban. Itt lambda függvényt használunk a számok négyzetre emeléséhez:

```
számok = [1, 2, 3, 4]
négyzetek = list(map(lambda x: x ** 2, számok))
print(négyzetek)
```

```
[1, 4, 9, 16]
```

A **filter** függvény: lambda függvényt használ arra, hogy kiválassza azokat az elemeket, amelyek egy adott feltételnek megfelelnek. Itt a páros számokat szűrjük ki:

```
számok = [1, 2, 3, 4, 5, 6]

párosok = list(filter(lambda x: x % 2 == 0, számok))

print(párosok)
```

```
[2, 4, 6]
```

5. Beépített függvények

- **A beépített függvények** a Python alapértelmezett részei. Segítségükkel különféle műveleteket hajthatunk végre extra könyvtárak importálása nélkül.
- Csoportosítva a legfontosabb beépített függvények:
  - **Matematikai: abs, round, min, max, sum**
  - **Típusokkal kapcsolatos: type, int, str, bool, float, list, dict**
  - **Iterációs: range, map, filter, zip, sorted, reversed, enumerate**
  - **Fájlkezelési: open, input, print**
  - **Funkcionális programozással kapcsolatos: lambda, eval, exec**
  - **Kivételkezelési: raise, assert**
  - **Objektumkezelési: dir, vars, id, classmethod, staticmethod**
- Pythonban a beépített függvények listáját a **dir(__builtins__)** paranccsal lehet megtekinteni. (pl.: print(__builtins__))

Beépített modulok

- A beépített modulokat a Python-telepítés tartalmazza, használatukhoz nincs szükség külön telepítésre, csak importálni kell őket a programba.
- **Importálás:**
  - **import modul_név**: A teljes modult importálja.
  - **import modul_név as alias**: Alias nevet ad a modulnak.
  - **from modul_név import elem_név**: Csak egyes elemeket importál a modulból.
  - **from modul_név import ***: Minden elemet importál a modulból.

Néhány fontos beépített modul kategóriák szerint:

- **Matematikai és statisztikai: math, cmath, random, statistics**
- **Dátum és időkezelés: datetime, time, calendar**
- **Szövegfeldolgozás: re, string, textwrap, unicodedata**
- **Fájl- és mappakezelés: os, os.path, pathlib, shutil, tempfile**
- **Fájlformátumok kezelése: json, csv, pickle, xml**
- **Rendszerszintű : sys, platform, subprocess, logging**
- **Hálózatkezelési: socket, http.client, urllib, ssl, ftplib**
- **Adatstruktúrák és algoritmusok: collections, array**
- **Titkosítás és hash-elés: hashlib, secrets**
- **Többszálú és párhuzamos programozás: threading, multiprocessing**
- **Tesztelési és hibakezelési: unittest, doctest**
- **Adatbáziskezelés: sqlite3, dbm, csv**
- **GUI és felhasználói felületek: tkinter**

Package (csomag) – modul – függvény hierarchia

- **Függvény**: Egyedi, kisebb kódegység, amely egy műveletet hajt végre. Például egy matematikai művelet vagy egy szöveges művelet.
- **Modul**: Egy fájl, amely Python kódot tartalmaz, és több függvényt, változót vagy osztályt foglalhat magába. Például a math modul tartalmazhat matematikai függvényeket.
- **Package**: Egy könyvtár, amely több modult (és alpackage-eket) tartalmaz, lehetővé téve a kód logikai szervezését és újra felhasználhatóságát. Ez a modulok strukturált gyűjteménye.

A Python package-ek installálása

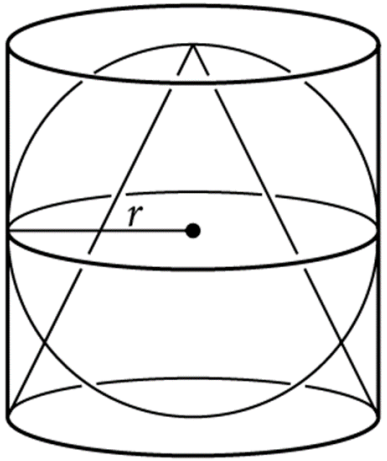
- A Python package-eket leggyakrabban a **Python Package Index**-ből **(PyPI)** lehet installálni. A PyPI a legnagyobb nyilvános Python csomagtár, ahol több tízezer nyílt forráskódú csomag található, amelyeket fejlesztők hoztak létre különféle feladatokhoz, például webfejlesztéshez, adatfeldolgozáshoz, gépi tanuláshoz és sok más területhez.

- **Hogyan lehet installálni Python package-eket?**

  A Pythonban a legelterjedtebb eszköz a package-ek telepítéséhez a **pip** nevű csomagkezelő. A **pip** a PyPI-ről tölti le és telepíti a csomagokat a Python környezetbe.

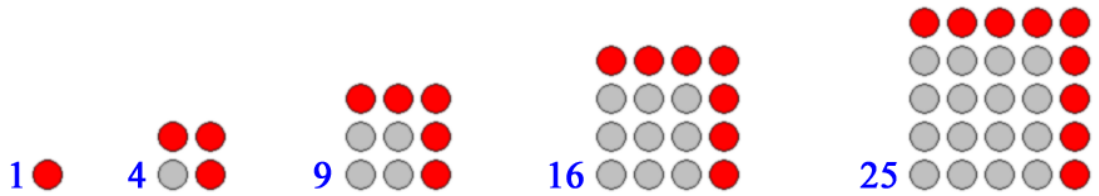
Példa 3.

```
import math     #Testek térfogatai lambda függvényekkel az [1..5] értékekre
kocka_térfogat = lambda oldal: oldal ** 3
gömb_térfogat = lambda átmérő: (4 / 3) * math.pi * (átmérő / 2) ** 3
henger_térfogat = lambda átmérő, mag: math.pi * (átmérő / 2) ** 2 * mag
kúp_térfogat = lambda átmérő, mag: (1 / 3) * math.pi * (átmérő / 2) ** 2 * mag
elemek = range(1, 6)            # Intervallum [1..5]
kocka_térfogatok = [kocka_térfogat(x) for x in elemek]
gömb_térfogatok = [gömb_térfogat(x) for x in elemek]
henger_térfogatok = [henger_térfogat(x, x) for x in elemek]
kúp_térfogatok = [kúp_térfogat(x, x) for x in elemek]
print("Kocka térfogatok:", kocka_térfogatok)
print("Gömb térfogatok:", gömb_térfogatok)
print("Henger térfogatok:", henger_térfogatok)
print("Kúp térfogatok:", kúp_térfogatok)
```



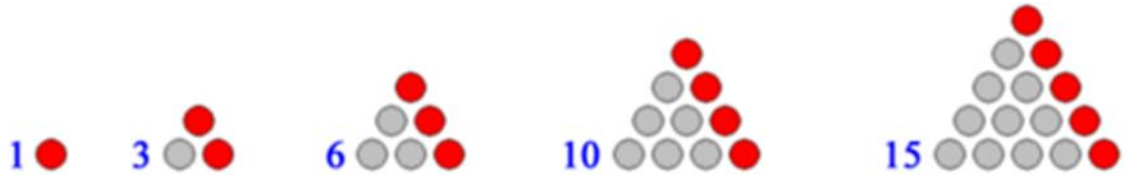
```
import random

def függvény(args, szabály):
    for x in args:
        print(szabály(x), end=' ')
    print()

lista = [random.randint(1, 100) for i in range(10)]
print(lista)

függvény(lista, lambda x: x % 2)
függvény(lista, lambda x: x % 5)
függvény(lista, lambda x: x - 1)
```

```
[5, 64, 70, 63, 99, 75, 8, 82, 80, 48]
1 0 0 1 1 1 0 0 0 0
0 4 0 3 4 0 3 2 0 3
4 63 69 62 98 74 7 81 79 47
```




```
def négyzetszámok(n):

    for i in range(n):
        négyzet = (i+1)**2
        yield négyzet

for elem in négyzetszámok(10):
    print(elem)
```

```
1
4
9
16
25
36
49
64
81
100
```




```
def háromszögszámok(n):
    s = 0
    for i in range(1,n+1):
        s += i
        yield s
for elem in háromszögszámok(10):
    print(elem, end=' ')
print()
```

```
1 3 6 10 15 21 28 36 45 55
```

```
számok = [1, 2, 3, 4, 5]
köbszámok = list(map(lambda x: x ** 3, számok))
print(köbszámok)
```

```
[1, 8, 27, 64, 125]
```

```
import random
lista = [random.randint(1, 100) for i in range(10)]
print(lista)
print(list(filter(lambda x: x % 2 == 0, lista)))
print(list(filter(lambda x: x % 2 != 0, lista)))
```

```
[26, 99, 74, 53, 82, 53, 66, 93, 95, 79]
[26, 74, 82, 66]
[99, 53, 53, 93, 95, 79]
```

```
data = [random.randint(-10,10) for x in range(10)]

filtered = list(filter(lambda x: x >= 0 and x % 2 == 1, data))

print(data)
print(filtered)
```

```
[7, 10, -8, 3, -4, -2, 0, -1, 5, 6]
[7, 3, 5]
```

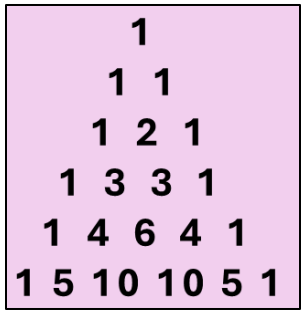
Példa 1.

```
def pizza_készítés(méret, *feltétek, **extra_opciók):
    print(f"Készítünk egy {méret} pizzát a következő feltétekkel:")
    for feltét in feltétek:
        print(f"- {feltét}")

    if extra_opciók.get("extra_sajt"):
        print("Extra sajtot teszünk rá!")

    if extra_opciók.get("glutén_mentes"):
        print("Gluténmentes tésztát használunk.")

pizza_készítés("nagy", "szalámi", "gomba", extra_sajt=True,
glutén_mentes=False)
```

Példa 2.

```
def rendelés_összegzés(**kwargs):
    print("Rendelés részletei:")
    for kulcs, érték in kwargs.items():
        print(f"{kulcs.capitalize()}: {érték}")

    if "fizetési_mód" not in kwargs:
        print("Figyelem! Nincs fizetési mód megadva.")
    if kwargs.get("ingyen_szállítás", False):   #Ha ingyen_szállítás nem létezik,
                                                #False értéket ad vissza.
        print("Az ingyenes szállítás elérhető.")
    else:
        print("Nincs ingyenes szállítás.")
rendelés_összegzés(termék="Laptop", ár=350000, darab=1,
fizetési_mód="bankkártya", ingyen_szállítás=True)
```

Példa 4.




```
def pascal_háromszög(n):       #Generálja a Pascal-háromszög első n sorát.
    háromszög = []
    for sor in range(n):               # Minden sor egy új lista, amelyet 1-gyel kezdünk, és
        új_sor = [1] * (sor + 1)        # a sor belső elemeit az előző sor elemeiből számoljuk
        for i in range(1, sor):
            új_sor[i] = háromszög[sor - 1][i - 1] + háromszög[sor - 1][i]
        háromszög.append(új_sor)
    return háromszög

def háromszög_megjelenít(háromszög):     #Megjeleníti a P.-háromszöget formázottan
    max_szélesség = len(" ".join(map(str, háromszög[-1])))    # A legutolsó sor szélessége
    for sor in háromszög:
        sor_str = " ".join(map(str, sor))
        print(sor_str.center(max_szélesség))

n = int(input("Add meg, hány sor Pascal-háromszöget szeretnél generálni: "))
háromszög = pascal_háromszög(n)              # A Pascal-háromszög generálása,
háromszög_megjelenít(háromszög)              # és megjelenítése
```

Összegzés

- Bemutattuk a hibakezelést, a try - except blokkok használatát és a kivételeket
- Elemeztük a függvény paraméter átadást, és használtunk pozicionális és kulcsszó függvényargumentumokat
- Készítettünk lambda függvényeket
- Áttekintettük a beépített függvények és modulok tárházát
- Bevezettük a package fogalmát
- Bemutattunk néhány példa programot

Konzultáció

Minden héten csütörtökön 18:00 – 20:00

Köszönöm a figyelmet!