

Előadás -3

Programozás Pythonban

Oktató :Dr. AALWAHAB DHULFIQAR

Felelős : Dr. Tejfel Mate



Mit fogsz tanulni:

- Döntések meghozatala (if utasítás)
- Ciklusok
- Logikai és bitenkénti műveletek
- Listák Pythonban
- Függvények, tuplik és szótárak



Kérdések és válaszok

Szerencsére a számítógépek csak kétféle választ ismernek:

Igen, ez igaz;
nem, ez hamis.

Hogyan kérdezzünk pythonban

Kérdéseket feltenni: A Python egy csomó nagyon speciális operátort használ.

Összehasonlítás: egyenlőségi operátor

Kérdés: két érték egyenlő?

Ennek a kérdésnek a feltevéséhez az == (egyenlő egyenlő) operátort használjuk (Ez egy bal oldali kötésű bináris operátor). Két argumentumra van szüksége, és ellenőrzi, hogy azok egyenlők-e.

Példa : A==B vagy A!=B

Összehasonlító operátorok: nagyobb, mint

Kérdés: két érték egyenlő?

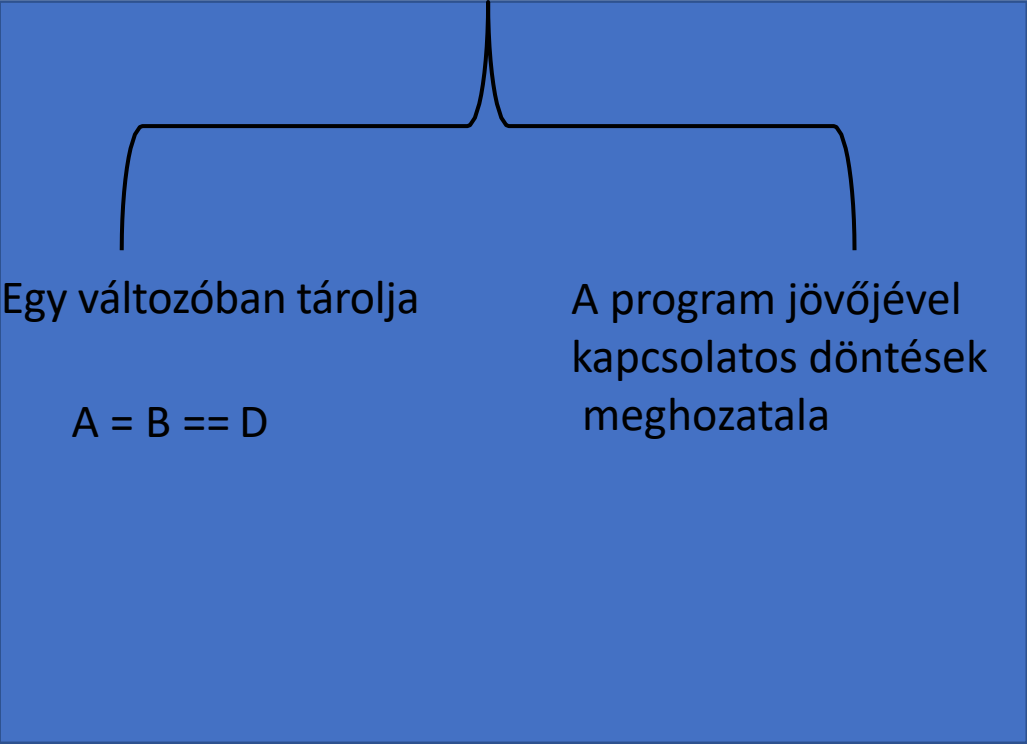
Összehasonlító kérdést a > (nagyobb, mint) operátorral is feltehet.

Példa : black_sheep > white_sheep # Nagyobb mint

Összehasonlító operátorok: nagyobb vagy egyenlő

>= (nagyobb vagy egyenlő).
< (kisebb, mint) operátor .
<= (kisebb vagy egyenlő)

A válaszok kihasználása



Elsőbbség	Operátor	
1	+, -	unary
2	**	hatvány
3	*, /, //, %	Szorzás, osztás
4	+, -	kivonás, összeadás
5	<, <=, >, >=	
6	==, !=	

Feltételek és feltételes végrehajtás

Ha egy feltétel teljesül, akkor tegye meg, és ne tegye meg, ha nem.

```
if true_or_not:  
    do_this_if_true
```

Ex:

```
if the_weather_is_good:  
    go_for_a_walk()  
    have_lunch()
```

```
if sheep_counter >= 120:  
    make_a_bed()  
    take_a_shower()  
    sleep_and_dream()  
    feed_the_sheepdog()
```

Feltételes végrehajtás: az if-else utasítás(kijelentés)

```
if true_or_false_condition:  
    perform_if_condition_true  
else:  
    perform_if_condition_false
```

```
if the_weather_is_good:  
    go_for_a_walk()  
else:  
    go_to_a_theater()  
have_lunch()
```

```
if the_weather_is_good:  
    go_for_a_walk()  
    have_fun()  
else:  
    go_to_a_theater()  
    enjoy_the_movie()  
have_lunch()
```

beágyazott if-utasítás

```
if the_weather_is_good:  
    if nice_restaurant_is_found:  
        have_lunch()  
    else:  
        eat_a_sandwich()  
else:  
    if tickets_are_available:  
        go_to_the_theater()  
    else:  
        go_shopping()
```

Az elif utasítás

```
if the_weather_is_good:  
    go_for_a_walk()  
elif tickets_are_available:  
    go_to_the_theater()  
elif table_is_available:  
    go_for_lunch()  
else:  
    play_chess_at_home()
```

Pszudokód és bevezetés a ciklusokba

Hogyan gondolkodsz, amikor programot akarsz írni?

```
1 largest_number = -999999999
2 number = int(input())
3- if number == -1:
4-     print(largest_number)
5-     exit()
6- if number > largest_number:
7-     largest_number = number
8- # Go to line 02
```

A kód ciklusba kötése a while

while there is something to do
do it

while conditional_expression:
 instruction

```
while
conditional_expression:
    instruction_one
    instruction_two
    instruction_three
    :
    :
    instruction_n
```

Végtelen ciklus

```
while True:
    print("I'm stuck inside a loop.")
```

```
# Az aktuális legnagyobb számot tárolja itt.
largest_number = -999999999

# Adja meg az első értéket.
number = int(input("Enter a number or type -1 to stop: "))

# Ha a szám nem egyenlő -1-el, akkor folytassuk.
. while number != -1:
    # A szám nagyobb, mint a legnagyobb_száma?
    if number > largest_number:
        # Igen, frissítse a legnagyobb_száma.
        largest_number = number
    # Adja meg a következő számot.
    number = int(input("Enter a number or type -1 to stop: "))

# Nyomtassa ki a legnagyobb számot.
print("The largest number is:", largest_number)
```

A kód cikluszása a for segítségével

Kódrészlet

```
i = 0
while i < 100:
    # do_something()
    i += 1
```

```
for i in range(100):
    # do_something()
    pass
```

```
for i in range(10):
    print("The value of i is currently", i)
```

```
for i in range(2, 8): # 8 et nem tartalmazza
    print("The value of i is currently", i)
```

Warning ! Donot do this

```
for i in range(1, 1):
    print("The value of i is currently", i)
```

Or this

```
for i in range(2, 1):
    print("The value of i is currently", i)
```

A break és continue utasítások

break - azonnal kilép a ciklusból, és feltétel nélkül befejezi a ciklus működését; a program a ciklus teste után a legközelebbi utasítás végrehajtását kezdi el;

continue - úgy viselkedik, mintha a program hirtelen elérte volna a ciklustest végét; a következő forduló megkezdődik, és a feltételkifejezés azonnal tesztelésre kerül.

kimenet

```
===== RESTART: C:/
The value of i is currently 0
The value of i is currently 1
The value of i is currently 2
The value of i is currently 3
The value of i is currently 4
The value of i is currently 5
The value of i is currently 6
The value of i is currently 7
The value of i is currently 8
The value of i is currently 9
```

Példa:

```
power = 1
```

```
for expo in range(16):
```

```
    print("2 to the power of", expo, "is", power)
```

```
    power *= 2
```

break - példa

```
print("The break instruction:")
for i in range(1, 6):
    if i == 3:
        break
    print("Inside the loop.", i)
print("Outside the loop.")
```

folytatás - példa

```
print("\nThe continue instruction:")
for i in range(1, 6):
    if i == 3:
        continue
    print("Inside the loop.", i)
print("Outside the loop.")
```

A while ciklus és az else ág

i = 1	
while i < 5:	1
print(i)	2
i += 1	3
else:	4
print("else:", i)	else: 5

for i in range(5):	
print(i)	
else:	0
print("else:", i)	1
	2
i = 111	3
for i in range(2, 1):	4
print(i)	else: 4
else:	else: 111
print("else:", i)	

Számítógépes logika---Computer logic

A) érv	B) érv	A and B
False	False	False
False	True	False
True	False	False
True	True	True

A) érv	B) érv	A or B
False	False	False
False	True	True
True	False	True
True	True	True

	tagadás
érv	not Argument
False	True
True	False

Logikai kifejezések

not (p and q) == (not p) or (not q)
not (p or q) == (not p) and (not q)

Logikai értékek vs. egyes bitek (Bitenként)

i = 1
j = not not i

tagadás

Bitenkénti műveletek (~) negation	
Argument	~ Argument
0	1
1	0

Bitsebességes operátorok----Bitwise operators

Bitwise operations (&, , and ^)				
Argument A	Argument B	A & B	A B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Logikai vs. bit műveletek

i = 15 i: 000000000000000000000000000000001111

j = 22 j: 00000000000000000000000000000000010110

log = i and j logneg = not l

True False

```
bit = i & j
```

```
i      000000000000000000000000000000001111
```

```
j      0000000000000000000000000000000000000000000000000000000
```

```
bit = i & j 000000000000000000000000000000000110
```

$x = x \& y$	$x \&= y$
$x = x y$	$x = y$
$x = x \wedge y$	$x \wedge= y$

rövidített forma

Hogyan kezeljük az egyes biteket?

```
flag_register = 0x1234
```

```
flag_register = 00000000000000000000000000000000x000
```

$$x \& 1 = x$$
$$x \& 0 = 0$$

the mask = 8

$$x \mid 1 = 1$$
$$x \mid 0 = x$$
$$x \wedge 1 = \sim x$$
$$x \wedge 0 = x$$

if flag register & the mask:

```
# My bit is set.
```

```
else:
```

My bit is reset.

Bináris balra eltolás és bináris jobbra eltolás

var = 17

```
var right = var >> 1
```

 osztunk 2-vel

```
var left = var << 2
```

 szorozzuk meg 2-vel

```
print(var, var_left, var_right)
```

Miért van szükségünk listákra?

```
var1 = int(input())
var2 = int(input())
var3 = int(input())
var4 = int(input())
var5 = int(input())
var6 = int(input())
:
```

a lista az az elemek gyűjteménye

```
numbers = [10, 5, 7, 2, 1]
```

A len() függvény

A lista hossza a végrehajtás során változhat. A listához új elemek adhatók hozzá, míg mások eltávolíthatók belőle. Ez azt jelenti, hogy a lista egy nagyon dinamikus entitás.

Ha ellenőrizni akarjuk a lista aktuális hosszát, használhatjuk a len() nevű függvényt (a neve a length szóból származik).

A függvény a lista nevét veszi argumentumként, és visszaadja a listában jelenleg tárolt elemek számát (más szóval a lista hosszát).

Hogyan változtathatja meg a lista egy kiválasztott elemének értékét?

```
numbers = [10, 5, 7, 2, 1]
```

```
numbers[0] = 111 <- indexing
numbers[1] = numbers[4]
print(numbers)
print(numbers[0])
print("\nList length:", len(numbers))
```

Elemek eltávolítása egy listából
del numbers[1]

Nem lehet hozzáférni egy nem létező elemhez.

```
print(numbers[4])
numbers[4] = 1
```

A negatív indexek legálisak



```
numbers = [111, 7, 2, 1]
print(numbers[-1])
print(numbers[-2])
```

1
2

Funkciók vs. módszerek

```
result = function(arg) ← Function  
result = data.method(arg) ← Method  
hogyan tudjuk megkülönböztetni
```

Elemek hozzáadása egy listához: append() és insert()

a végére csatolni

list.append(value)
list.insert(location, value)

egy pontosított helyre

```
numbers = [111, 7, 2, 1]  
print(len(numbers))  
print(numbers)  
###  
numbers.append(4)  
print(len(numbers))  
print(numbers)  
###  
numbers.insert(0, 222)  
print(len(numbers))  
print(numbers)
```

```
my_list = [] # Creating an empty list.
```

```
for i in range(5):  
    my_list.append(i + 1)
```

```
print(my_list)
```

```
my_list = [10, 1, 8, 3, 5]  
total = 0
```

```
for i in range(len(my_list)):  
    total += my_list[i]
```

```
print(total)
```

```
my_list = [10, 1, 8, 3, 5]  
length = len(my_list)  
for i in range(length // 2):  
    my_list[i], my_list[length - i - 1] = my_list[length - i - 1],  
    my_list[i]  
print(my_list)
```

Lista rendezése

```
my_list = []
swapped = True
num = int(input("How many elements do you want to sort: "))

for i in range(num):
    val = float(input("Enter a list element: "))
    my_list.append(val)

while swapped:
    swapped = False
    for i in range(len(my_list) - 1):
        if my_list[i] > my_list[i + 1]:
            swapped = True
            my_list[i], my_list[i + 1] = my_list[i + 1], my_list[i]

print("\nSorted:")
print(my_list)
```

```
my_list = [8, 10, 6, 2, 4]
my_list.sort()
print(my_list)
```

A listák belső élete - The inner life of lists

```
list_1 = [1]
list_2 = list_1
list_1[0] = 2
print(list_2)
```

Output = 2?

szeletek -- slices

my_list[start:end]

```
# Copying the entire list.
list_1 = [1]
list_2 = list_1[:]
list_1[0] = 2
print(list_2)

# Copying some part of the list.
my_list = [10, 8, 6, 4, 2]
new_list = my_list[1:3]
print(new_list)

my_list = [10, 8, 6, 4, 2]
new_list = my_list[1:-1]
print(new_list)
```

```
my_list = [10, 8, 6, 4, 2]
new_list = my_list[-1:1]
print(new_list)
```

[]

Az in és not in operátorok

```
elem in my_list  
elem not in my_list  
my_list = [0, 3, 12, 8, 2]
```

```
print(5 in my_list)  
print(5 not in my_list)  
print(12 in my_list)
```

Listák listákban

```
row = []  
for i in range(8):  
    row.append('_____')  
print(row)
```

```
row = ["-----" for i in range(8)]
```

Átfogó lista-----List comprehension

```
squares = [x ** 2 for x in range(10)]
```

```
board = [[EMPTY for i in range(8)] for j in range(8)]
```

```
EMPTY = "-"  
ROOK = "ROOK"  
board = []  
for i in range(8):  
    row = [EMPTY for i in range(8)]  
    board.append(row)  
board[0][0] = ROOK  
board[0][7] = ROOK  
board[7][0] = ROOK  
board[7][7] = ROOK  
print(board)
```

```
for i in board:  
    print(i)
```

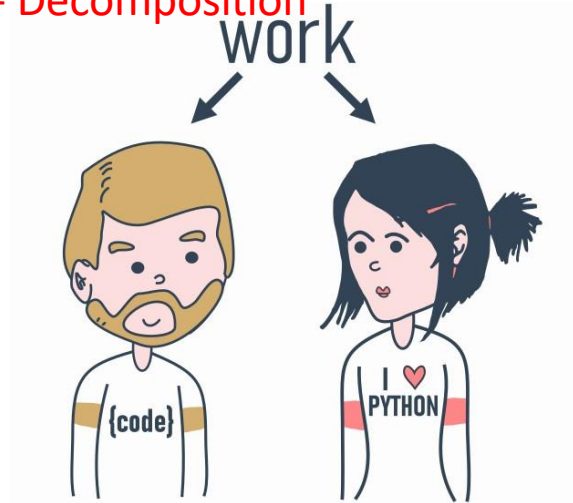
	A	B	C	D	E	F	G	H	
8	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]	[0][6]	[0][7]	8
7	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]	[1][6]	[1][7]	7
6	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]	[2][5]	[2][6]	[2][7]	6
5	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]	[3][5]	[3][6]	[3][7]	5
4	[4][0]	[4][1]	[4][2]	[4][3]	[4][4]	[4][5]	[4][6]	[4][7]	4
3	[5][0]	[5][1]	[5][2]	[5][3]	[5][4]	[5][5]	[5][6]	[5][7]	3
2	[6][0]	[6][1]	[6][2]	[6][3]	[6][4]	[6][5]	[6][6]	[6][7]	2
1	[7][0]	[7][1]	[7][2]	[7][3]	[7][4]	[7][5]	[7][6]	[7][7]	1
	A	B	C	D	E	F	G	H	

Függvények: Miért van szükségünk függvényekre?

Gyakran előfordul, hogy egy adott kódrészlet sokszor ismétlődik a programban.

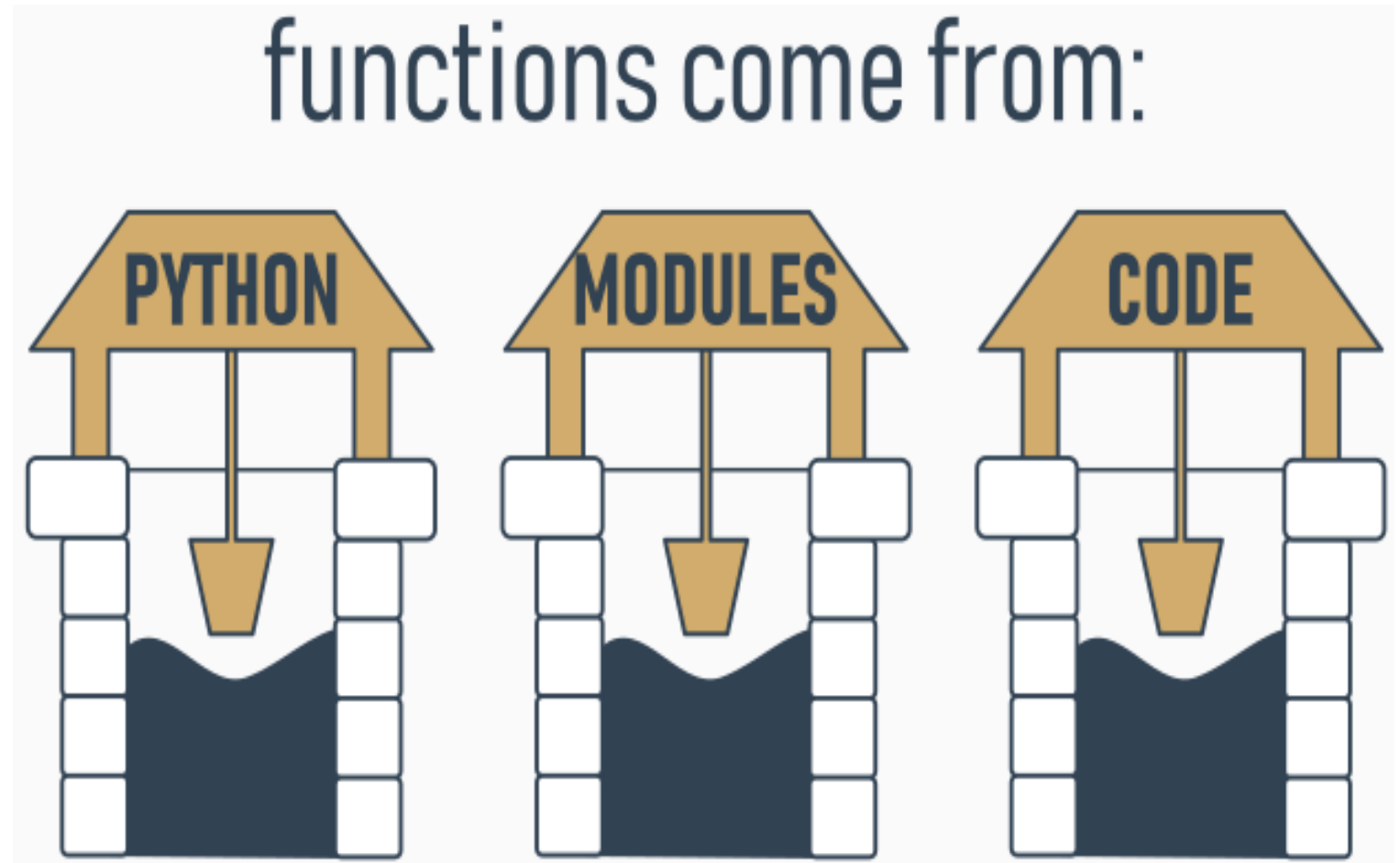
ha egy adott kódrészlet egynél több helyen kezd megjelenni, fontolja meg annak lehetőségét, hogy azt egy függvény formájában elkülönítse

Bontás -- Decomposition

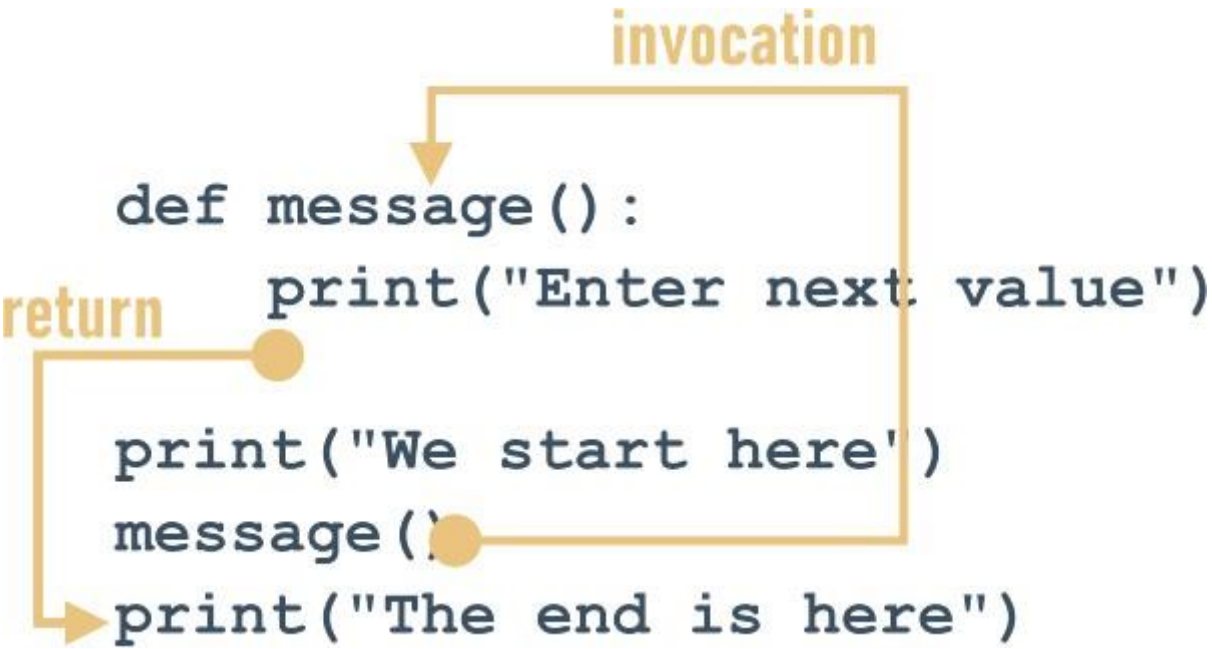


Az első függvényed

```
def function_name():  
    function_body  
  
def message():  
    print("Enter a value: ")
```



Hogyan működnek a függvények



Ne tegye ezeket:

```
print("We start here.")  
message()  
print("We end here.")
```

```
def message():  
    print("Enter a value: ")
```

```
def message():  
    print("Enter a value: ")
```

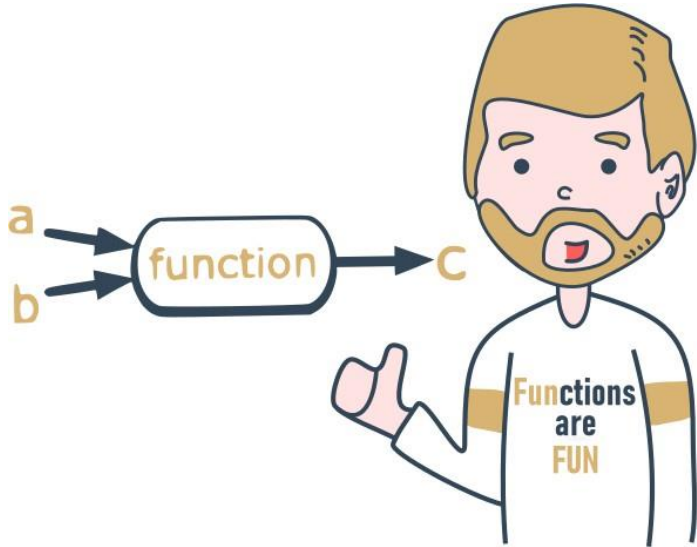
```
message = 1
```

Paraméterezett függvények

```
def function(parameter):  
    ###  
  
def message(number):  
    print("Enter a number:", number)
```

```
def message(number):  
    print("Enter a number:", number)  
  
message()
```

Paraméterezett függvények:



```
def message(what, number):  
    print("Enter", what, "number", number)
```

```
message("telephone", 11)  
message("price", 5)  
message("number", "number")
```

Pozicionális paraméterátadás

```
def my_function(a, b, c):  
    print(a, b, c)
```

```
my_function(1, 2, 3)
```

```
def introduction(first_name, last_name):  
    print("Hello, my name is", first_name, last_name)
```

```
introduction("Luke", "Skywalker")  
introduction("Jesse", "Quick")  
introduction("Clark", "Kent")
```

Kulcsszó argumentum átadása

```
def introduction(first_name, last_name):  
    print("Hello, my name is", first_name, last_name)
```

```
introduction(first_name = "James", last_name = "Bond")  
introduction(last_name = "Skywalker", first_name = "Luke")
```


Pozicionális és kulcsszavas argumentumok keverése

pozicionális argumentu

```
def adding(a, b, c):  
    print(a, "+", b, "+", c, "=", a + b + c)  
adding(1, 2, 3)
```

1 + 2 + 3 = 6

kulcsszavas argumentumok

```
adding(c = 1, a = 2, b = 3)
```

2 + 3 + 1 = 6

```
adding(3, c = 1, b = 2)
```

3 + 2 + 1 = 6

```
adding(3, a = 1, b = 2)
```

TypeError : adding() got multiple values for argument 'a'

```
adding(4, 3, c = 2)
```

4 + 3 + 2 = 9

```
def introduction(first_name, last_name="Smith"):  
    print("Hello, my name is", first_name, last_name)
```

```
introduction("James", "Doe")
```

```
introduction("Henry")
```

```
introduction(first_name="William")
```

Hatások és eredmények: a return parancs

viisszatérés kifejezés nélkül

```
def happy_new_year(wishes = True):  
    print("Three...")  
    print("Two...")  
    print("One...")  
    if not wishes:  
        return  
  
    print("Happy New Year!")
```

```
happy_new_year()  
Three...  
Two...  
One...  
Happy New Year!
```

```
happy_new_year(False)  
Three...  
Two...  
One...
```

invocation

```
def boring_function():  
    return 13
```

return

```
x = boring_function()
```

```
def boring_function():  
    print("'Boredom Mode' ON.")  
    return 123  
  
print("This lesson is interesting!")  
boring_function()  
print("This lesson is boring...")
```

None

A None egy kulcsszó

```
value = None
if value is None:
    print("Sorry, you don't carry any value")
```

```
print(None + 2)
```

TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'

```
def strange_function(n):
    if(n % 2 == 0):
        return True
print(strange_function(2))
print(strange_function(1))
```

True

None

```
def my_function():
    print("Do I know that variable?", var)
var = 1
my_function()
print(var)
```

listák és függvények

```
def strange_list_fun(n):
    strange_list = []

    for i in range(0, n):
        strange_list.insert(0, i)

    return strange_list

print(strange_list_fun(5))
```

[4, 3, 2, 1, 0]

Függvények és területek

```
def scope_test():
    x = 123
    scope_test()
    print(x)
```

NameError: name 'x' is not defined

A globális kulcsszó

```
global name  
global name1, name2, ...
```

```
def my_function():  
    global var  
    var = 2  
    print("Do I know that variable?", var)
```

```
var = 1  
my_function()  
print(var)
```

Egyszerű funkciók: Fibonacci-számok

```
def fib(n):  
    if n < 1:  
        return None  
    if n < 3:  
        return 1
```

```
    elem_1 = elem_2 = 1  
    the_sum = 0  
    for i in range(3, n + 1):  
        the_sum = elem_1 + elem_2  
        elem_1, elem_2 = elem_2, the_sum  
    return the_sum
```

```
for n in range(1, 10): # testing  
    print(n, "->", fib(n))
```

Találkozunk a laborban 😊