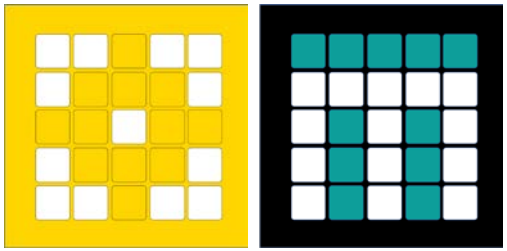
PRIME LESSONS

By the Makers of EV3Lessons



FUNCȚIILE

BY SANJAY AND ARVIND SESHAN

This lesson uses SPIKE 3 software

OBIECTIVELE LECȚIEI

- Învățăm să creeăm și să utilizăm funcțiile
- Învățăm de ce funcțiile sunt utile

FUNCȚII

- Funcțiile Python sunt similare cu funcțiile algebrice
  - $f(x)=3x^2$
  - $f(3)=27 \rightarrow$ f(3) returns 27
- Funcția este definită ca un set de coduri care ia una sau mai multe valori de intrare și returnează una sau mai multe rezultate.
- Funcțiile sunt foarte versatile. Poți să pui atât cod cât vrei, atâtea intrări câte dorești și poți returna atâtea date câte dorești.
- Indentația este necesară pentru a fi siguri ca doar codul pe care-l vrei în funcție rulează atunci când este apelată.

```
def f(x):
    y = 3*x**2 # y=3x^2
    return y
```

```
def g():
    for i in range(7):
        print(i)
    return "Hello"
```

CONSTRUCȚIA UNEI FUNCȚII

- O definire a funcției începe cu:

  def YOUR_NAME_HERE(PARAMETERS):

  - Codul identat mai jos rulează atunci când funcția este apelată
- Poți numi funcția oricum îți dorești, dar numele trebuie să înceapă cu o literă (în general literă mică)
  - O bună convenție de numire a funcțiilor și variabilelor este camelCase (primul cuvânt este cu litere mici și restul începe cu literă mare). Toate cuvintele sunt unite. E.g. myFunction()
- Parametrii sunt listați despărțiți de virgulă în interiorul parantezelor urmăriți de numele funcției
  - IMPORTANT: Acești parametri sunt variabile locale și pot fi folosite doar în interiorul funcției.

```
def g():
    for i in range(7):
        print(i)
    return "Hello"
```

```
def h(a,b):
    for i in range(7):
        print(i, a, b)
    return "Hello"
```

APELAREA/ RULAREA UNEI FUNCȚII

- Pentru a apela o funcție, oriunde în cod, plasează numele funcției, urmat de paranteze cu valorile dorite a parametrilor
  - Linia evidențiată cu galben în dreapta apelează funcția g(a,b)
- Linia care apelează funcția procedează la rularea codului funcției, unde a și b sunt înlocuite temporar cu valorile parametrilor
- După ce această funcție este rulată , codul continuă ca de obicei

```
def g(a, b):
    print("Hello", a, b)

print("Running....")
g(2, 3)
print("Done!")
```

Output:

```
Running....
Hello 2 3
Done!
```

FUNCȚIILE CARE RETURNEAZĂ DATE

- Plasați "return DATA" în interiorul funcției pentru a obține DATA ca un rezultat al funcției
- Funcția g() returnează valoarea 10, care poate fi utilizată mai apoi în program

```
def g(a, b):
    print("Hello", a, b)
    return 10

print("Running....")
print(g(2, 3))
print("Done!")

Output:
Running....
Hello 2 3
10
Done!
```

FUNCȚII PRECONSTRUITE

- Sunt multe funcții preconstruite
- Vezi lista celor mai importante mai jos

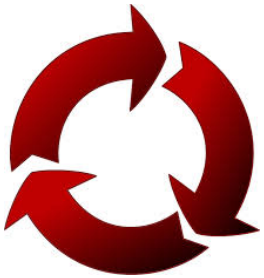
```
print("Type conversion functions:")
print(bool(0))   # convert to boolean (True or False)
print(float(42)) # convert to a floating point number
print(int(2.8))  # convert to an integer (int)

print("Basic math functions:")
print(abs(-5))   # absolute value
print(max(2,3))  # return the max value
print(min(2,3))  # return the min value
print(pow(2,3))  # raise to the given power (pow(x,y) == x**y)
print(round(2.354, 1)) # round with the given number of digits

print("Pause/sleep")
import time
time.sleep(4) # sleep for n seconds
```

CÂND FOLOSIM FUNCȚIILE?

- Fantastice pentru task-urile repetitive
- Mișcarea pentru o anumită distanță, întoarcerile etc.
- Fantastice pentru organizarea și simplificarea codului



CE FACE O FUNCȚIE SĂ FIE UTILĂ

- Notă: Să faci funcții cu date de intrare și de ieșire este foarte util. Totuși trebuie să fii atent să nu faci funcții prea complicate.
- Întrebare: Priviți cele 3 funcții de mai jos. Care dintre ele crezi că e utilă?
  - Turn90degrees (Întoarce robotul 90 de grade)
  - TurnDegrees cu input unghi și putere
  - TurnDegrees cu input unghi, putere, croazieră/frână etc

- Răspuns:
  - Turn90degrees poate fi utilizată des, dar vei fi forțat să faci un alt Myblock pentru unghiuri. Aceasta nu poate fi rezolvat mai târziu.
  - TurnDegrees cu unghi și putere este probabil cea mai bună alegere.
  - TurnDegrees cu unghi, putere, croazieră/frînă, etc. E poate cea mai customizată opțiune, dar unele input-uri nu vor fi niciodată folosite.

VARIABILE SCOPE ÎN FUNCȚII

- Ce credeți că va face acest cod?

```
y = 7
def f(x):
    print(x)
    print(y)
f(4)
print(y)
print(x)
```

VARIABLE SCOPE ÎN FUNCȚII

- Ce crezi ca va face acest cod?

```
y = 7
def f(x):
    print(x)
    print(y)
f(4)
print(y)
print(x)
```

Output:

```
4
7
7
NameError: name 'x' is not defined
```

Hmmm....there seems to be an error

VARIABLE SCOPE ÎN FUNCȚII CONT.

- Am menționat că parametrii funcțiilor sunt variabile locale... Asta înseamnă că acele "variabile" pot fi accesate din interiorul funcției
- Print(x) de pe ultima linie este înafara funcției și de aceea variabila x nu poate fi citită
- Variabilele definite în afara funcției sunt considerate globale, însemnând că pot fi folosite oriunde.
- Observați că dacă o variabilă locală sau globală împart același nume, variabila locală va fi cea apelată, dacă nu se specifică altceva

```
y = 7
def f(x):
    print(x)
    print(y)
f(4)
print(y)
print(x)
```

Red este o funcție scope.
Yellow este global scope.
Red poate de asemenea accesa variabilele globale

Avansați: utilizați "`global x`" în funcție pentru a forța utilizarea unei variabile globale peste o variabilă locală

EXEMPLU DE DOMENIU DE APLICARE A VARIABILELOR

- Același nume de variabile sunt utilizate în domenii diferite

```
y = 7
x = 2
def f(x):
    print(x)
    print(y)
f(4)
print(y)
print(x)
```

Output:

```
4
7
7
2
```

In this case the x from the global scope is used on the last line, while the local one is used in the function (not overwriting the global one)

OBIECTE ȘI METODE

- Obiectele sunt ca un set de funcții dar sunt inițializate și „salvate” într-o variabilă.
  - În Python, orice este tehnic un obiect (chiar și ints, strings, etc.)
- Obiectele sunt create utilizând o funcție constructor
  - E.g., var = object()
- Metodele sunt tipuri speciale de funcții asociate cu un obiect
- Pentru a apela o metodă, trebuie fie să ai o variabilă fie o valoare de același tip pe care să o apelezi
  - Variabila/valoarea pe care o vei folosi ca input implicit la metodă
- Tipurile de variabile special asociate cu SPIKE Prime/MINDSTORMS expun un interval de diferite metode pentru a controla robotul tău. Vom parcurge aceste tipuri și metodele lor mai apoi în alte lecții.
- De exemplu, șirurile de caractere au o varietate de metode cu scopuri diferite
  - Unele exemple sunt arătate în dreapta
  - Toată lista de metode cu șiruri de caractere sunt listate aici https://www.w3schools.com/python/python_ref_string.asp

```
s = str("Test“)
s.upper() # TEST
s.lower() # test
s.find("T") # 0
```

PROVOCARE

- Creează o funcție cu un parametru n care să adune toate numerele de la 0 la n, unde n este întreg
- Returnează răspunsul
- Indicii:
  - Veți utiliza un loop și o declarație de întors

SOLUȚIE

```
def sumToN(n):
    total = 0
    counter = 1
    while (counter <= n):
        total += counter
        counter += 1
    return total

print(sumToN(5), 1+2+3+4+5)
```

FUNCȚIILE ASYNC

- An **async** function is a special function that can optionally return as soon as it is called, without waiting for all its code to finish. You must **await** them if you want to wait for them to finish before continuing with the next line of code.
- Async functions are very useful when you want to run two independent pieces of code concurrently (i.e. at the same time)
  - Move an attachment while moving the robot at the same time
  - Moving each wheel of the robot independently under certain conditions, e.g., squaring on a line.
- Spike 3 has built-in async functions. If you do not call them with await, they will run concurrently. This can be a common source of bugs, so be aware of it. The following code will run both motors concurrently:

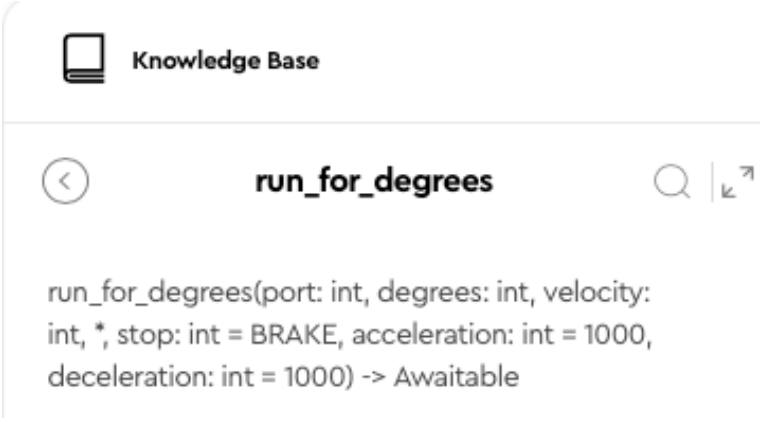
```
motor.run_for_degrees(port.A, 360, 200)
motor.run_for_degrees(port.B, 360, 200)
```

To run them sequentially, add an **await** to the first line.

```
await motor.run_for_degrees(port.A, 360, 200)
motor.run_for_degrees(port.B, 360, 200)
```

FUNCȚIILE ASYNC

- Cum poți spune care funcții predefinite sunt „asynchronous”?
  - Uitați-vă în documentația „Cunoștințe de bază”. Dacă vezi un `->Awaitable` după definirea funcției, aveți nevoie să știți că o vei „citi” cu **await** dacă vă doriți să finalizeze task-ul.



PROVOCARE

- Scrie un program care să facă HUB-ul să scoată un sunet la fiecare 3 secunde și să afișeze simbolul inimii pe matricea LED la fiecare 2 secunde, la infinit.
- Sfaturi:
  - Ai nevoie de o funcție async pentru a scoate un sunet, și alta pentru a face un sunet de bătaie a inimii pe hub.
  - Rulează-le concomitent în funcția principală.

SOLUȚIA

```
from hub import light_matrix, sound
import runloop

# Bipăie la fiecare 3 secunde pentru totdeauna
async def beepEveryThreeSeconds():
    while True:
        await runloop.sleep_ms(3000) # așteaptă pentru 3 secunde
        sound.beep() # sună un beep

# inima se aprinde și se stinge pe matricea LED la fiecare 2 secunde pentru totdeauna
async def heartBeatEveryTwoSeconds():
    while True:
        await runloop.sleep_ms(1000) # așteaptă pentru o secundă
        light_matrix.show_image(light_matrix.IMAGE_HEART) # afișează inima
        await runloop.sleep_ms(1000) # așteaptă pentru o secundă
        light_matrix.clear() # ascunde inima

async def main():
    beepFunc = beepEveryThreeSeconds() # creează o co-rutină pentru beep
    heartFunc = heartBeatEveryTwoSeconds() # creează o co-rutină pentru o bătaie a inimii
    runloop.run(*[beepFunc,heartFunc]) # rulează ambele co-rutine concomitent
runloop.run(main())
```

PROVOCARE

- Modificați programul pentru:
  - Să scoată un sunet la fiecare 3 secunde, de 5 ori.
  - Inima LED apare la fiecare 2 secunde, de 10 ori.
  - Afișează "Done" când **ambele** task-uri, și oprește programul.

- Sfaturi:
  - Ai nevoie de o funcție „async"pentru a scote sunetul beep, și altă funcție pentru a face ca inima LED de pe HUB să fie afișată.
  - Fiecare funcție se setează ca un Boolean la „adevărat" când este gata
  - Rularea LOOP așteaptă până când task-urile sunt finalizate, apoi afișează "Done"

SOLUȚIA (PAGINA 1 DIN 2)

```
from hub import light_matrix, sound
import runloop, sys

# variabile globale
beepDone = False
beatDone = False

# Funcția pentru a opri un program via system exit
def stopAndExitProgram():
    sys.exit("Stopping")

# Funcția utilizată de runloop pentru a aștepta
def all_done():
    return beatDone and beepDone # returnează valoarea de adevăr pentru ca ambele variabile sunt ,,adevărat”

# Bipăie la fiecare 3 secunde de n ori
async def beepEveryThreeSeconds(n):
    global beepDone # utilizează variabile globale
    beepDone = False # inițializează ,,false’’
    for i in range(n):
        await runloop.sleep_ms(3000) # așteaptă 3 secunde
        sound.beep() # scoate un singur beep
    beepDone = True # setează variabila pe valoarea ,,true’’
```

<Continued on next page>

SOLUȚIA (PAGINA 2 DIN 2)

```
# Inima LED se aprinde și se stinge la fiecare 2 secunde de n ori
async def heartBeatEveryTwoSeconds(n):
    global beatDone # utilizează valori globale
    beatDone = False # inițializează la ,,false''
    for i in range (n):
        await runloop.sleep_ms(1000) # așteaptă o secundă
        light_matrix.show_image(light_matrix.IMAGE_HEART) # afișează inima LED
        await runloop.sleep_ms(1000) # așteaptă o secundă
        light_matrix.clear() # stinge inima LED
    beatDone = True # setează variabila la valoarea de adevăr ,,true''

async def main():
    beepFunc = beepEveryThreeSeconds(5)
    heartFunc = heartBeatEveryTwoSeconds(10)
    runloop.run(*[beepFunc,heartFunc])
    await runloop.until(all_done)
    print("All done")
    stopAndExitProgram()

runloop.run(main())
```

CREDITS

- Această lecție a fost creată de Sanjay Seshan și Arvind Seshan for SPIKE Prime Lessons
- La această lecție au contribuit membrii comunității FLL Share & Learn.
- Mai multe lecții sunt disponibile pe www.primelessons.org
- Această lecție a fost tradusă în limba romană de echipa de robotică FTC – ROSOPHIA #21455 RO20