python

FUNCTIONS

פונקציות מוגדרות על ידי המשתמש הן בלוק של קוד שמתבצע רק כאשר קוראים להן ניתן
להעביר לפונקציה נתונים הנקראים פרמטרים פונקציה יכולה להחזיר נתונים כתוצאה

Creating a Function

ב- Python פונקציה מוגדרת באמצעות מילת המפתח def

Example :

```
def my_function():
  print("Hello from a function")
```

FUNCTIONS

Calling a Function

כדי לקרוא לפונקציה, יש להשתמש בשם הפונקציה ולאחריו סוגריים.

Example

```
def my_function():
  print("Hello from a function")

my_function()
```

FUNCTIONS

With format and string

ניתן להשתמש בפונקציה יחד עם עיצוב format ומחרוזת string גם כן.

Example :

```
def mystring(text):
    print(f'Hi {text}')
mystring('Mickael')
```

FUNCTIONS

Arguments – “Args” / Parameters

ניתן להעביר מידע לפונקציות כארגומנטים.

הארגומנטים מוגדרים אחרי שם הפונקציה, בתוך הסוגריים. ניתן להוסיף כמה ארגומנטים שרוצים, כל עוד מפרידים ביניהם עם פסיק.

הפונקציה הבאה מכילה ארגומנט אחד fname כאשר הפונקציה נקראת, אנחנו מעבירים שם פרטי, המשמש בתוך הפונקציה להדפיס את השם המלא.

```
def my_function(fname):
  print(fname + " Jones")

my_function(“Lisa")
my_function(“Fred")
my_function(“Liam")
```

FUNCTIONS

Number of Arguments

כברירת מחדל, יש לקרוא לפונקציה עם מספר הארגומנטים הנכון. כלומר, אם הפונקציה שלך מצפה ל-2 ארגומנטים, יש לקרוא לפונקציה עם 2 ארגומנטים, לא יותר ולא פחות.

אם תנסה לקרוא לפונקציה עם 1 או 3 ארגומנטים, תקבל שגיאה.

Example :
This function expects 2 arguments, and gets 2 arguments:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Lisa", "Jones")
```

FUNCTIONS

Arbitrary Arguments, *args

אם אינך יודע כמה ארגומנטים יועברו לפונקציה שלך, הוסף * לפני שם הפרמטר בהגדרת הפונקציה. כך הפונקציה תקבל זוג tuple של ארגומנטים, ותוכל לגשת לפריטים בהתאם.

Example :
If the number of arguments is unknown, add a * before the parameter name:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

FUNCTIONS

Keyword Arguments - kwargs

ניתן גם לשלוח ארגומנטים עם תחביר של מפתח = ערך. בצורה זו הסדר של הארגומנטים לא משנה.

Example :

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

FUNCTIONS

Default Parameter Value

אם נגדיר ערך ברירת מחדל לארגומנט ונקרא לפונקציה מבלי להעביר את הארגומנט, היא תשתמש בערך ברירת המחדל.

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

FUNCTIONS

Passing a List as an Argument

ניתן לשלוח כל סוג נתונים כארגומנט לפונקציה (מחרוזת, מספר, רשימה, מילון וכו'), והוא יטופל באותו סוג נתונים בתוך הפונקציה. זה עדיין יהיה סוג הנתונים המקורי (לדוגמה, רשימה) כשהוא יגיע לפונקציה.

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]
vegetables = ["carrot","eggplant","green beans"]

my_function(fruits)
my_function(vegetables)
```

```
apple
banana
cherry
carrot
eggplant
green beans
```

FUNCTIONS

Return Statement

כדי לאפשר לפונקציה להחזיר ערך, יש להשתמש בהצהרת return. פעולה זו תסיים את הביצוע של קריאת הפונקציה ותשלח את תוצאת הפונקציה בחזרה אל הקורא.

Example :

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

הפונקציה מבצעת חישוב ומחזירה את התוצאה, והפקודות print() מדפיסות את התוצאה הזו.

הסיבה שמשתמשים ב-`return` היא כדי שהפונקציה תוכל להחזיר תוצאה שניתן להשתמש בה מחוץ לפונקציה עצמה.

אם לא היה `return`, הפונקציה רק הייתה מבצעת את החישוב בתוך עצמה, אבל לא הייתה מעבירה את התוצאה חזרה למקום שקרא לה. בפועל, המשמעות היא שלא הייתה דרך להשתמש בתוצאה של החישוב.

לדוגמה:

- כאשר יש `return`, התוצאה של החישוב (x * 5) מוחזרת אל הפונקציה שקראה ל-`my_function()`.
לכן, אתה יכול להדפיס את התוצאה בעזרת `print()`, או להשתמש בה בצורה אחרת.

אם לא היה `return`, התוצאה הייתה מחושבת אבל לא ניתן היה להחזיר אותה לשימוש בקוד שקרא לפונקציה.

בקיצור, `return` מאפשר לפונקציה להחזיר תוצאה כדי שתוכל להשתמש בה מחוץ לפונקציה.

אם הפונקציה לא הייתה מחזירה ערך, לא היית יכול להדפיס או להשתמש בתוצאה מחוץ לפונקציה.

FUNCTIONS

Global Variables

משתנים שנוצרים מחוץ לפונקציה (כמו בדוגמאות למעלה) נקראים משתנים גלובליים.

משתנים גלובליים יכולים לשמש את כולם, הן בתוך פונקציות והן מחוץ להן.

Example : Create a variable outside of a function, and use it inside the function

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

FUNCTIONS

Global Variables

אם יוצרים משתנה עם אותו שם בתוך פונקציה, המשתנה הזה יהיה מקומי ויוכל לשמש רק בתוך הפונקציה.

המשתנה הגלובלי עם אותו שם יישאר כפי שהיה, גלובלי ועם הערך המקורי שלו.

Example : Create a variable inside a function, with the same name as the global variable

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

FUNCTIONS

The global Keyword

בדרך כלל, כאשר יוצרים משתנה בתוך פונקציה, המשתנה הוא מקומי וניתן להשתמש בו רק בתוך אותה פונקציה.

כדי ליצור משתנה גלובלי בתוך פונקציה, ניתן להשתמש במילת המפתח global.

Example : If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

FUNCTIONS

The global Keyword

כמו כן, יש להשתמש במילת המפתח global אם ברצונך לשנות משתנה גלובלי בתוך פונקציה.

Example : To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

FUNCTIONS

The pass Statement

הגדרות של פונקציות לא יכולות להיות ריקות, אך אם מסיבה כלשהי יש לך הגדרה של פונקציה ללא תוכן, אפשר לשים את ההצהרה pass כדי להימנע משגיאה.

Example :

```
def myfunction():
  pass
```

FUNCTIONS

MAIN function

הפונקציה הראשית main משמשת לרוב כנקודת הכניסה של תוכנית. היא מהווה את נקודת ההתחלה של הביצוע ובדרך כלל מכילה את הלוגיקה ברמה הגבוהה של התוכנית. הפונקציה הראשית נקראת בדרך כלל בסוף הסקריפט או באמצעות בלוק if __name__ == '__main__', על מנת להבטיח שהיא תבוצע רק כאשר הסקריפט מורץ ישירות ולא כאשר הוא מיובא כמודול.

```
def main():
    # Main function logic goes here
    print("Hello, world!")

# Call the main function if the script is run directly
if __name__ == '__main__':
    main()
```

באמצעות שימוש בפונקציה הראשית main ניתן לארגן את הקוד בצורה מודולרית יותר, להפריד את נקודת הכניסה מהגדרות פונקציות אחרות, ולהפוך את הקוד לקל יותר לקריאה ולתחזוקה.

FUNCTIONS

Proper Code Structure

1) Imports

```
Import random
```

2) Variables and inputs (Global)

```
num = 0
name = input(“Name“)
list = ["John","Ben","Jul"]
```

3) Functions

4) Main

```
if __name__ == '__main__':
    main()
```

FUNCTIONS

Built-in functions

פייתון מציעה מגוון רחב של פונקציות מובנות, שהן מאוד שימושיות עבור משימות תכנות שונות.

print(): Used to display output or values to the console.

input(): Allows the user to enter input from the console.

len(): Returns the length of a string, list, tuple, or any iterable.

type(): Returns the type of an object.

int(), float(), str(), bool(): Convert values to integer, float, string, or boolean types, respectively.

FUNCTIONS

Built-in functions

range(): Generates a sequence of numbers within a specified range.

sum(): Returns the sum of all elements in an iterable.

max(), min(): Returns the maximum or minimum value from an iterable.

sorted(): Returns a sorted list of elements from an iterable.

abs(): Returns the absolute value of a number.

round(): Rounds a number to a specified precision.

zip(): Combines multiple iterables into a single iterable of tuples.

FUNCTIONS

Built-in functions

enumerate(): Returns an iterable of tuples containing indices and values from an iterable.

map(): Applies a function to each item in an iterable.

filter(): Filters elements from an iterable based on a specified condition.

all(), any(): Returns True if all or any elements in an iterable are true, respectively.

str.format(): Formats a string by replacing placeholders with values.

list(), tuple(), set(), dict(): Converts an iterable to a list, tuple, set, or dictionary, respectively.

open(): Opens a file and returns a file object.

FUNCTIONS

Lambda Functions

בפייתון, פונקציה מסוג Lambda היא פונקציה אנונימית (ללא שם) שמוגדרת בקוד. הפונקציות האלו לרוב קצרות ומשמשות בעיקר במקום שבו לא נדרש להגדיר פונקציה שלמה.

התחביר הבסיסי של פונקציית Lambda הוא:

Example : Add 10 to argument a, and return the result:

lambda *arguments* : *expression*

```
x = lambda a : a + 10
print(x(5))
```

הארגומנטים הם הקלטים לפונקציה, והביטוי מתבצע ומחזיר את התוצאה.

FUNCTIONS

Lambda Functions

פונקציות Lambda יכולות לקבל כל מספר של ארגומנטים, והביטוי הפנימי מבוצע בהתאם לארגומנטים שנמסרו לה.

Example : Multiply argument a with argument b and return the result:

```
x = lambda a, b : a * b
print(x(5, 6))
```

Example : Summarize argument a, b, and c and return the result:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

FUNCTIONS

Why Use Lambda Functions?

הכוח של פונקציות Lambda בולט כאשר משתמשים בהן בתור פונקציה אנונימית בתוך פונקציה אחרת. בפונקציה האנונימית, אפשר להגדיר פעולה שתתבצע על ארגומנט אחד, כמו הכפלה של מספר ב- Lambda בתוך פונקציה אחרת כדי להחזיר את התוצאה.

פונקציות Lambda בנויות לביצוע ביטויים פשוטים וקצרים ולא מתאימות למצבים שבהם יש צורך בלוגיקה מורכבת, כמו לולאות או זרימות שליטה. במקרים כאלו, עדיף להשתמש בפונקציה רגילה שמוגדרת עם def.

פונקציות Lambda שימושיות בעיקר כשצריך פונקציה זמנית לצורך מסוים או בעת שימוש בתכנות פונקציונלי שבו פונקציות מועברות כארגומנטים לפונקציות אחרות.

```
def myfunc(n):
  return lambda a : a * n
mydoubler = myfunc(2)
mytripler = myfunc(3)
print(mydoubler(11))
print(mytripler(11))
```

Exercises

Exercises

Exercise 1 :
Write a function to add “ish” at the end of the user input. The user would have to tape “exit” to stop the program, if not, it will continue to ask him.

Exercise 2 :
Write a program with functions that asks the user to choose 2 numbers.
Each number has to be multiplied by 2.
Add the results together

Exercise 3 :
Write a Login Database : it will ask the user for his username and his password, and check if it’s the same credentials stored in your database.

Exercises

Exercise 4 :
Create a keylogger, using a counter. It will count the number of characters that the user tape on his keyboard.

Exercise 5 :
Create the game “Hot & Cold”. Ask the user to guess an random number.
If it’s too low or too high, tell him.

Exercise 6 :
Create a calculator.


Q&A