

python

Debugging and Error Handling

Debugging and Error Handling

ניפוי שגיאות Debugging הוא תהליך של זיהוי ותיקון שגיאות בקוד שלך. זהו חלק חיוני מתהליך פיתוח התוכנה, מכיוון שהוא מאפשר לך לזהות ולתקן בעיות לפני שהן הופכות לבעיות גדולות יותר. נבחן את יסודות ניפוי השגיאות ונכיר טכניקות נפוצות לזיהוי ותיקון שגיאות בקוד שלך. אחת מהנקודות החשובות ביותר שצריך להבין לגבי ניפוי שגיאות היא שזהו תהליך של שלילת אפשרויות. כלומר, תצטרך לעבור בצורה שיטתית על הקוד שלך ולשלול את הגורמים האפשריים לשגיאה.

אחת מהטכניקות הנפוצות ביותר לניפוי שגיאות היא שימוש בפקודות הדפסה print על ידי הוספת פקודות הדפסה בקוד שלך, תוכל לראות אילו ערכים מוקצים למשתנים ואיך הם משמשים. זה יכול לעזור לך לזהות היכן השגיאה מתרחשת ומה עשוי לגרום לה.

טכניקה נוספת לפתרון שגיאות היא שימוש בנפיין שגיאות Debugger נפיין שגיאות הוא כלי שמאפשר לך לעבור על הקוד שלך שלב אחר שלב ולבדוק את הערכים של המשתנים בכל שלב. זה יכול להיות כלי חזק לזיהוי ותיקון שגיאות, מכיוון שהוא מאפשר לך לראות בדיוק מה קורה בקוד שלך בכל שלב.

Common Types of Errors in Python

בפייתון, ישנם מספר סוגים של שגיאות שעלולות להופיע במהלך פיתוח הקוד שלך. הבנה של הסוגים השונים של השגיאות הללו ואופן גרימתן יכולה לעזור לך לזהות ולפתור בעיות בקוד בצורה יעילה יותר.

תוכל לזהות ולפתור בעיות בקוד בצורה יעילה יותר על ידי הבנה של הסוגים הנפוצים של השגיאות ואופן גרימתן.

בנוסף, שימוש נכון בטיפול בשגיאות exception handling בקוד שלך יכול לעזור לך לתפוס ולטפל בשגיאות אלו, מה שימנע מהן לגרום לקריסת התוכנית שלך.

Common Types of Errors in Python

Syntax Errors

שגיאות תחביר: שגיאות אלו מתרחשות כאשר המפרש ה-Python Interpreter נתקל בקוד שלא תואם את כללי התחביר של השפה. לדוגמה, נקודתיים חסרים או סוגריים לא תואמים.

Name Errors:

שגיאות שם: שגיאות אלו מתרחשות כאשר המפרש נתקל במשתנה או בפונקציה שלא הוגדרו. זה עלול לקרות כאשר יש שגיאת כתיב בשם המשתנה או כאשר לא מייבאים פונקציה מהמודול הנכון.

Type Errors

שגיאות סוג: שגיאות אלו מתרחשות כאשר משתמשים במשתנה עם סוג נתונים לא נכון. לדוגמה, ניסיון להוסיף מחרוזת string למספר שלם integer יגרום לשגיאת סוג.

Common Types of Errors in Python

Syntax Errors:

שגיאות אלו מתרחשות כאשר המפרש של פייתון נתקל בקוד שאינו תואם את כללי התחביר של השפה. לדוגמה, נקודתיים חסרים או סוגריים לא תואמים.

```
1 a = 2
2 if(a == 2)
3 print("True")
```

```
File "<ipython-input-4-ffea50a12548>", line 2
```

```
    if(a==2)
```

```
        ^
```

```
SyntaxError: invalid syntax
```

Common Types of Errors in Python

Name Errors:

שגיאות אלו מתרחשות כאשר המפרש נתקל במשתנה או בפונקציה שלא הוגדרו. זה יכול לקרות כאשר יש שגיאת כתיב בשם המשתנה או כאשר פונקציה לא יובאה מהמודול הנכון.

```
>>> a = 10
```

```
>>> name = 'hello'
```

```
>>> print(NAME)
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#2>", line 1, in <module>
```

```
    print(NAME)
```

```
NameError: name 'NAME' is not defined
```

Common Types of Errors in Python

Type Errors:

שגיאות אלו מתרחשות כאשר משתמשים במשתנה עם סוג נתונים שגוי. לדוגמה, ניסיון להוסיף מחרוזת string למספר שלם integer יגרום לשגיאת סוג.

```
Traceback (most recent call last):  
  File "main.py", line 3, in <module>  
    result = list2.join(list1)  
TypeError: sequence item 2: expected str instance, int found
```


Common Types of Errors in Python

Attribute Errors

שגיאות מאפיין: שגיאות אלו מתרחשות כאשר מנסים לגשת למאפיין או שיטה באובייקט שלא קיימים. לדוגמה, ניסיון לגשת למאפיין שאינו קיים במופע של מחלקה.

Index Errors

שגיאות אינדקס: שגיאות אלו מתרחשות כאשר מנסים לגשת לאינדקס ברשימה או בטופל שנמצא מחוץ לגבולות.

Key Errors:

שגיאות מפתח: שגיאות אלו מתרחשות כאשר מנסים לגשת למפתח במילון שאינו קיים.

Zero Division Errors:

שגיאות חלוקה באפס: שגיאות אלו מתרחשות כאשר מנסים לחלק מספר באפס.

Import Errors:

שגיאות ייבוא: שגיאות אלו מתרחשות כאשר המפרש לא מצליח למצוא מודול שיובא.

Common Types of Errors in Python

Index Errors:

שגיאות אינדקס: שגיאות אלו מתרחשות כאשר מנסים לגשת לאינדקס ברשימה או בטופל שנמצא מחוץ לגבולות..

```
main.py
1 # index      0      1      2      3      4
2 color = ['red', 'green', 'blue', 'black', 'white']
3
4 print(color[5])
```

Traceback (most recent call last):
File "main.py", line 4, in <module>
print(color[5])
IndexError: list index out of range

Common Types of Errors in Python

Key Errors:

שגיאות מפתח: שגיאות אלו מתרחשות כאשר מנסים לגשת למפתח במילון שאינו קיים.

```
[5] dictionary = {"name": "John", "age": 25}
print(dictionary["profession"])
```

KeyError

Traceback (most recent call last)

<ipython-input-5-d658100ca2f9> in <cell line: 2>()

1 dictionary = {"name": "John", "age": 25}

----> 2 print(dictionary["profession"])

KeyError: 'profession'



Common Types of Errors in Python

Zero Division Errors:

שגיאות חלוקה באפס: שגיאות אלו מתרחשות כאשר מנסים לחלק מספר באפס.

```
1 first_number = 56.4
2 second_number = 0
3 output = first_number / second_number
4 print(output)
5
```

Shell ×

```
>>> %Run -c $EDITOR_CONTENT
```

```
Traceback (most recent call last):
```

```
File "<string>", line 3, in <module>
```

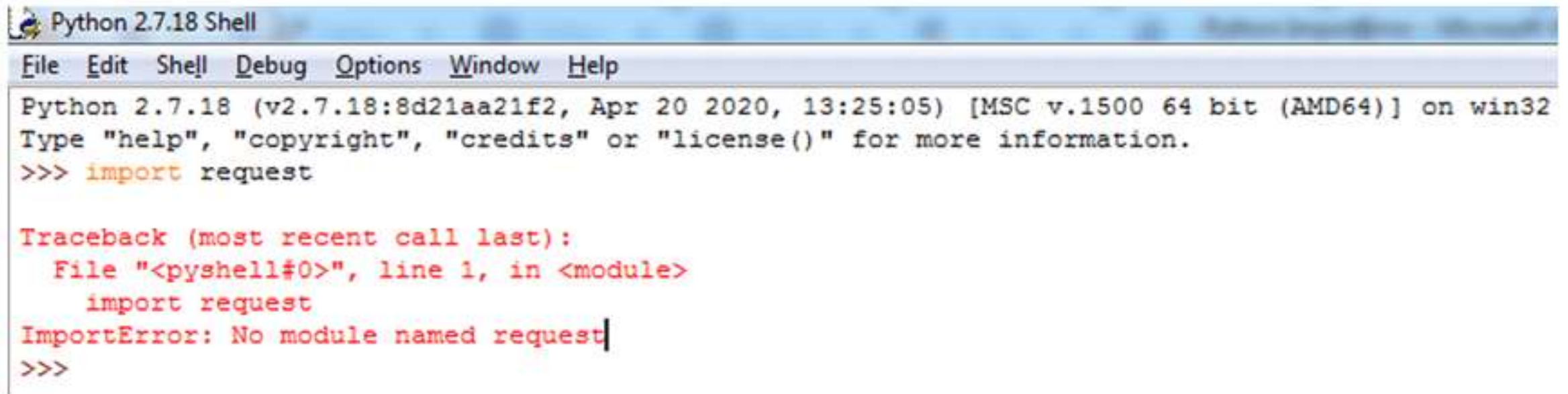
```
ZeroDivisionError: float division by zero
```

```
>>>
```

Common Types of Errors in Python

Import Errors:

שגיאות ייבוא: שגיאות אלו מתרחשות כאשר המפרש לא מצליח למצוא מודול שייבא.

A screenshot of a Python 2.7.18 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the Python version and build information: 'Python 2.7.18 (v2.7.18:8d21aa21f2, Apr 20 2020, 13:25:05) [MSC v.1500 64 bit (AMD64)] on win32'. It then prompts the user to type 'help', 'copyright', 'credits', or 'license()' for more information. The user has entered the command '>>> import request'. The shell responds with a 'Traceback (most recent call last):' message, showing the file '<pyshell#0>', line 1, in <module>, where the 'import request' command was entered. The error message is 'ImportError: No module named request'. The prompt '>>>' is shown again at the bottom.

```
Python 2.7.18 Shell
File Edit Shell Debug Options Window Help
Python 2.7.18 (v2.7.18:8d21aa21f2, Apr 20 2020, 13:25:05) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import request

Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import request
ImportError: No module named request
>>>
```

Note : The request module does exist, but here it isn't written correctly.

Handling Exceptions in Python

פייתון, יוצאים מן הכלל exceptions משמשים לטיפול במצבים בלתי צפויים או שגיאות שמתרחשות במהלך הרצת תוכנית. כאשר נזרק יוצא מן הכלל, הזרימה הרגילה של הרצת התוכנית מופרעת והתוכנית קופצת לחלק מיוחד בקוד הנקרא "מטפל בשגיאות" exception handler הרעיון המרכזי בטיפול בשגיאות הוא להפריד בין הקוד שיכול לזרוק שגיאה לבין הקוד שמטפל בה.

בפייתון, ניתן לטפל בשגיאות באמצעות בלוק try-except. try-בלוק ה-try מכיל את הקוד שעלול לזרוק שגיאה, ובלוק ה-except מכיל את הקוד שמטפל בשגיאה.

Example :

```
try:
    x = 1/0
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Example :

The try block will generate an exception, because x is not defined:

```
try:
    print(x)
except:
    print("An exception occurred")
```

Handling Exceptions in Python

Many Exceptions

אתה יכול להגדיר כמה בלוקים של טיפול בשגיאות exception blocks שתרכזה, אם אתה רוצה להריץ קוד מיוחד עבור סוג מסוים של שגיאה.

Example : Print one message if the try block raises a NameError and another for other errors:

```
try: print(x)
except NameError:
    print("Variable x is not defined")
except:
    print("Something else went wrong")
```

Handling Exceptions in Python

Else

אתה יכול להשתמש במילת המפתח else כדי להגדיר בלוק קוד שירוצץ אם לא התרחשו שגיאות.

Example : the try block does not generate any error:

```
try:  
    print("Hello")  
except:  
    print("Something went wrong")  
else:  
    print("Nothing went wrong")
```


Handling Exceptions in Python

Finally

בלוק ה- finally, אם הוא מוגדר, ירוץ ללא קשר אם בלוק ה- try זרק שגיאה או לא.

```
try:  
    print(x)  
except:  
    print("Something went wrong")  
finally:  
    print("This code will always run.")
```

Handling Exceptions in Python

Finally

זה יכול להיות שימושי לסגור אובייקטים ולנקות משאבים.

Example : Try to open and write to a file that is not writable:

```
try:
    f = open("demofile.txt")
    try:
        f.write("Lorum Ipsum")
    except:
        print("Something went wrong when writing to the file")
    finally:
        f.close()
except:
    print("Something went wrong when opening the file")
```

התוכנית יכולה להמשיך, מבלי להשאיר את אובייקט הקובץ פתוח.

Handling Exceptions in Python

בלוק ה- **try** מאפשר לך לבדוק קטע קוד עבור שגיאות

בלוק ה- **except** מאפשר לך לטפל בשגיאה

בלוק ה- **else** מאפשר לך להריץ קוד כאשר אין שגיאה

בלוק ה- **finally** מאפשר לך להריץ קוד, ללא קשר לתוצאה של בלוקי ה- **try** ו-**except**

Handling Exceptions in Python

Raise an exception

כהמפתח בפייתון, אתה יכול לבחור לזרוק יוצא מן הכלל exception אם מתקיים תנאי מסוים כדי לזרוק (או להעלות) יוצא מן הכלל, השתמש במילת המפתח raise

Example : Raise an error and stop the program if x is lower than 0:

```
x = -1
```

```
if x < 0:
```

```
    raise Exception("Sorry, no numbers below zero")
```

Handling Exceptions in Python

Raise an exception

מילת המפתח `raise` משמשת כדי להעלות יוצא מן הכלל `exception`

אתה יכול להגדיר איזה סוג של שגיאה להעלות ואת הטקסט שיודפס למשתמש

Example : Raise a `TypeError` if `x` is not an integer:

```
x = "hello"
```

```
if not type(x) is int:  
    raise TypeError("Only integers are allowed")
```

Libraries in PYTHON

What is a Module ?

מודול הוא קובץ המכיל קבוצה של פונקציות שתרצה לכלול באפליקציה שלך. כדי ליצור מודול, פשוט שמור את הקוד שברצונך לכלול בקובץ עם סיומת .py אתה יכול לקרוא לקובץ המודול איך שתרצה, אך הוא חייב לכלול את סיומת הקובץ .py

Save this code in a file named mymodule.py

```
def greeting(name):  
    print("Hello, " + name)
```

כעת, נוכל להשתמש במודול שיצרנו על ידי שימוש בפקודת import בא את המודול שנקרא mymodule וקרא לפונקציה greeting:

```
import mymodule  
mymodule.greeting("Jonathan")
```

Note: When using a function from a module, use the syntax: module_name.function_name.

Variables in Module

המודול יכול להכיל פונקציות, כפי שתואר כבר, אך גם משתנים מכל הסוגים (מערכים, מילונים, אובייקטים וכו'):

שמור קוד זה בקובץ שנקרא mymodule.py

```
person1 = {  
    "name": "John",  
    "age": 36,  
    "country": "Norway"  
}
```

ייבא את המודול שנקרא mymodule ולגשת למילון person1.

```
import mymodule  
a = mymodule.person1["age"]  
print(a)
```


Re-naming a Module

אתה יכול ליצור כינוי **alias** בעת ייבוא מודול, על ידי שימוש במילת המפתח **as**

Example

Create an alias for mymodule called mx:

```
import mymodule as mx
```

```
a = mx.person1["age"]  
print(a)
```

Built-in Modules

ישנם מספר מודולים מובנים בפייתון, אותם תוכל לייבא מתי שתרצה.

Example

Import and use the random module:

```
import random
```

```
x = random.randrange(3,9)
```

```
print(x)
```

Using the dir() Function

יש פונקציה מובנית להצגת כל שמות הפונקציות (או שמות המשתנים) במודול. הפונקציה dir().

Example

List all the defined names belonging to the random module:

```
import random
```

```
x = dir(random)  
print(x)
```

ניתן להשתמש בפונקציה dir() על כל המודולים, כולל אלו שאתה יוצר בעצמך.

Import From Module

אתה יכול לבחור לייבא רק חלקים ממודול, על ידי שימוש במילת המפתח `from`. המודול שנקרא `mymodule` מכיל פונקציה אחת ומילון אחד.

```
def greeting(name):  
    print("Hello, " + name)
```

```
person1 = {  
    "name": "John",  
    "age": 36,  
    "country": "Norway"  
}
```

Import only the `person1` dictionary from the module:

```
from mymodule import person1  
print (person1["age"])
```

כאשר מייבאים באמצעות מילת המפתח `from`, אין להשתמש בשם המודול כאשר מתייחסים לאלמנטים בתוך המודול. דוגמה:
`person1["age"]` ולא `mymodule.person1["age"]`.

The Standard Library

ספריית הסטנדרט של פייתון מכילה את התחביר המדויק, הסמנטיקה והטוקנים של פייתון.

היא מכילה מודולים מובנים שמספקים גישה לפונקציונליות בסיסית של המערכת.

רוב הספריות של פייתון נכתבות בשפת התכנות C.

ספריית הסטנדרט של פייתון מורכבת מיותר מ-200 מודולים ליבה.

כל אלו עובדים יחד כדי להפוך את פייתון לשפת תכנות ברמה גבוהה.

לספריית הסטנדרט של פייתון יש תפקיד חשוב מאוד.

בלעדיה, למתכנתים אין גישה לפונקציונליות של פייתון. ישנן מספר ספריות אחרות בפייתון שמקלות על חיי המתכנתים.

10 commonly used libraries

- 1. TensorFlow:** This library was developed by Google in collaboration with the Brain Team. It is an open-source library used for high-level computations. It is also used in machine learning and deep learning algorithms. It contains a large number of tensor operations. Researchers also use this Python library to solve complex computations in Mathematics and Physics.
- 2. Matplotlib:** This library is responsible for plotting numerical data. And that's why it is used in data analysis. It is also an open-source library and plots high-defined figures like pie charts, histograms, scatterplots, graphs, etc.
- 3. Pandas:** Pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualizations, Aggregations, etc.

10 commonly used libraries

4. **Numpy:** The name “Numpy” stands for “Numerical Python”. It is the commonly used library. It is a popular machine learning library that supports large matrices and multi-dimensional data. It consists of in-built mathematical functions for easy computations. Even libraries like TensorFlow use Numpy internally to perform several operations on tensors. Array Interface is one of the key features of this library.
5. **SciPy:** The name “SciPy” stands for “Scientific Python”. It is an open-source library used for high-level scientific computations. This library is built over an extension of Numpy. It works with Numpy to handle complex computations. While Numpy allows sorting and indexing of array data, the numerical data code is stored in SciPy. It is also widely used by application developers and engineers.
6. **Scrapy:** It is an open-source library that is used for extracting data from websites. It provides very fast web crawling and high-level screen scraping. It can also be used for data mining and automated testing of data.

10 commonly used libraries

- 7. Scikit-learn:** It is a famous Python library to work with complex data. Scikit-learn is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with Numpy and SciPy.
- 8. PyGame:** This library provides an easy interface to the Standard Directmedia Library (SDL) platform-independent graphics, audio, and input libraries. It is used for developing video games using computer graphics and audio libraries along with Python programming language.
- 9. PyTorch:** PyTorch is the largest machine learning library that optimizes tensor computations. It has rich APIs to perform tensor computations with strong GPU acceleration. It also helps to solve application issues related to neural networks.
- 10. PyBrain:** The name "PyBrain" stands for Python Based Reinforcement Learning, Artificial Intelligence, and Neural Networks library. It is an open-source library built for beginners in the field of Machine Learning. It provides fast and easy-to-use algorithms for machine learning tasks. It is so flexible and easily understandable and that's why is really helpful for developers that are new in research fields.

Use of Libraries in Python Program

כאשר אנו כותבים תוכניות גדולות בפייתון, אנו רוצים לשמור על מודולריות של הקוד. לשם תחזוקה קלה של הקוד, אנו מחלקים אותו לחלקים שונים, וניתן להשתמש בחלקים אלו שוב בעת הצורך.

בפייתון, מודולים ממלאים את התפקיד הזה. במקום להשתמש באותו קוד בתוכניות שונות ולהפוך את הקוד למורכב, אנו מגדירים פונקציות נפוצות במודולים, ואנו יכולים פשוט לייבא אותן לתוכנית בכל פעם שיש צורך בכך.

אין צורך לכתוב את הקוד מחדש, אך עדיין ניתן להשתמש בפונקציונליות שלו על ידי ייבוא המודול. מספר מודולים קשורים מאוחסנים בספרייה. כאשר אנו צריכים להשתמש במודול, אנו מייבאים אותו מהספרייה שלו.

בפייתון, זהו תהליך פשוט מאוד בזכות התחביר הקל שלה. כל מה שצריך הוא להשתמש בפקודת ה-`import`.

Importing library

Importing math library

```
import math
```

```
A = 16
```

```
print(math.sqrt(A))
```

Output

```
4.0
```

בקטע הקוד שמעל, ייבאנו את ספריית `math` והשתמשנו באחת מהשיטות שלה, כלומר `sqrt` (שורש ריבועי), מבלי לכתוב את הקוד הממשי לחישוב השורש הריבועי של מספר. כך ספריה הופכת את העבודה של המתכנתים לפשוטה יותר. אבל במקרה הזה, היינו צריכים רק את שיטת ה- `sqrt` מתוך ספריית `math`, אך ייבאנו את כל הספרייה. במקום זאת, אנו יכולים גם לייבא פריטים ספציפיים ממודול בספרייה.

Importing specific items from a library module

במקום לייבא ספרייה שלמה כדי להשתמש רק באחת מהשיטות שלה, אפשר פשוט לייבא פריטים ספציפיים מתוך הספרייה.

```
from math import sqrt, sin
```

```
A = 16
```

```
B = 3.14
```

```
print(sqrt(A))
```

```
print(sin(B))
```

Output

```
4.0
```

```
0.0015926529164868282
```

בקוד שמעל, אפשר לראות שייבאנו רק את השיטות sqrt חישוב (שורש ריבועי) ו-sin (חישוב סינוס) מתוך ספריית math.

Python Datetime Module

תאריך בפייתון אינו סוג נתונים בפני עצמו, אלא הוא מיוצג כאובייקט תאריך.

Example : Import the datetime module and display the current date:

```
import datetime
```

```
x = datetime.datetime.now()
```

```
print(x)
```

➔ 2023-07-24 07:14:17.660624

The date contains year, month, day, hour, minute, second, and microsecond.

Python Datetime Module

The strftime() Method

המודול datetime מכיל הרבה שיטות שמחזירות מידע על אובייקט התאריך.

Example

Return the year and name of weekday:

```
import datetime
```

```
x = datetime.datetime.now()
```

```
print(x.year)
```

```
print(x.strftime("%A"))
```

➔ 2023

Monday

Python Datetime Module

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST

Python Datetime Module

%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%C	Century	20
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%
%G	ISO 8601 year	2018
%u	ISO 8601 weekday (1-7)	1
%V	ISO 8601 weeknumber (01-53)	01

Exercises

Exercises

Exercise 1:

Write a function `division()` that accepts two arguments. The function should be able to catch an `ZeroDivisionError` exception, giving the alert you want, not the default one.

Exercise 2:

Work on the "Student_grades" file, and add error handling (we don't want to see strings...)

Exercise 3:

Write a function , subject on your choice , and put in it error handling

Q&A