

CMP4266: Computer Programming UG1

Lab Session 6: Python Errors and Test Case

Objectives

- Describe syntax errors, logic errors and runtime errors which may occur while developing a program.
- Understand and identify different types of errors.
- Test Case Design.

1. Reminder about errors

1.1 Syntax Error

The most common type of error, a syntax error occurs when the interpreter is reading the Python file to determine if it is valid Python code and encounters something it doesn't "understand". The interpreter will check things like indentation, use of parentheses and square brackets, proper forms of if, elif, and else blocks, and so on.

Common Python syntax errors include:

- leaving out a keyword
- putting a keyword in the wrong place
- leaving out a symbol, such as a colon, comma or brackets
- misspelling a keyword
- incorrect indentation
- empty block

1.2 Runtime Error

If a program is syntactically correct – that is, free of syntax errors – it will be run by the Python interpreter. However, the program may exit unexpectedly during execution if it encounters a *runtime error* – a problem which was not detected when the program was parsed, but is only revealed when a particular line is executed. When a program comes to a halt because of a runtime error, we say that it has crashed.

Some examples of Python runtime errors:

- division by zero
- performing an operation on incompatible types
- using an identifier which has not been defined
- accessing a list element, dictionary value or object attribute which doesn't exist
- trying to access a file which doesn't exist

Runtime errors often creep in if you don't consider all possible values that a variable could contain, especially when you are processing user input. You should always try to add checks to your code to make sure that it can deal with bad input and edge cases gracefully.

1.3 Logical errors

Logical errors are the most difficult to fix. They occur when the program runs without crashing, but produces an incorrect result. The error is caused by a mistake in the program's logic. You won't get an error message, because no syntax or runtime error has occurred. You will have to find the problem on your own by reviewing all the relevant parts of your code.

Here are some examples of mistakes which lead to logical errors:

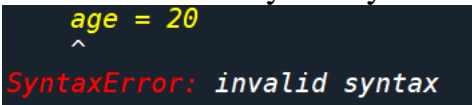
- using the wrong variable name
- indenting a block to the wrong level
- using integer division instead of floating point division
- getting operator precedence wrong
- making a mistake in a boolean expression
- off-by-one, and other numerical errors

1.4 Examples of errors

For example, here's a simple file that has a syntax error. In this case, the programmer has forgotten to add a closing square bracket to the definition of the list `my_list`:

```
my_list = [1, 2, 3
age = 20
```

When this is run by the Python interpreter, we get the following error message:

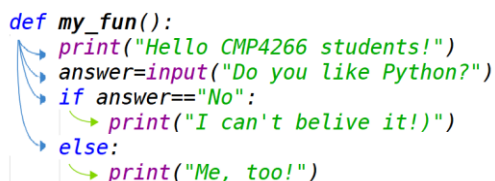


After Python is done with the first line, it expects to keep reading elements of `my_list` (i.e., it expects to see a `,` and then another number). However, it sees that there is a variable assignment happening on the next line, and it stops with a syntax error.

Indentation Error

Indentation refers to the spaces at the beginning of a code line. Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important. Python uses indentation to indicate a block of code. Consequently, if you have an indentation error, the entire program will fail to run, and the interpreter will return an error.

For example, the `def` line used in function definition is considered to be the parent of the lines indented underneath it. Here's an illustrated example, which shows the parent/child relationships of each line:



All of the lines indented under the `def` line are children of the `def` line, with the exception of the lines indented deeper. The `print("I can't believe it!")` call is a child of the `if` line. The `print("Me, too!")` line is a child of the `else` line.

Unindent does not match any outer indentation level

```
1 def my_func(x):
2     if x < 5:
3         a = 10
4     b = 5
```

Since the line containing `b = 5` only has six spaces before it, Python couldn't determine whether that line's parent was the `if` line or the `def` line.

Expected an indented block

```
1 def my_func(num):
2     if num % 2 == 0:
3
4     else :
5         print("odd number!")
```

After Python reads the `if` line, it expects to see at least one child line following it.

Name Error

This error occurs when a name (a variable *or* function name) is used but there is not an assignment statement for this variable.

```
1 x=2
2 print(x+y)
```

description	meaning
name 'y' is not defined	on this line, the variable name <code>y</code> is used, but Python has not yet seen an assignment statement for this variable

Type Error

This error occurs when a function or operator cannot be applied to the given values, due to the fact that the value's type is inappropriate. This can happen when two incompatible types are used together, for example:

```
1 module="CMP"
2 print(module+4266)
```

```
TypeError: can only concatenate str (not
"int") to str
```

Index Error

This error occurs when an index into a list or string is out of range. For example, attempting `my_list[3]` when the list `my_list` has zero, one, or two or three elements will produce an `IndexError`.

```
list1=["Playstation", "IOS", "Steam"]
list1[10]="Google Play"
```

```
IndexError: list assignment index out of range
```

Note also that, when *slicing* a list or a string, `IndexError` will *not* be produced if any indices in the slice are out of bounds.

```
list1=["Playstation","IOS","Steam"]
print(list1[0:100])
```

Output :

```
['Playstation', 'IOS', 'Steam']
```

Logical Error

```
x=5
y=6

z = x+y/2

print("The average of %f and %f is %f" % (x, y , z))
```

The output is:

The average of 5.000000 and 6.000000 is 8.000000

2. Getting started

- 2.1 Consider the following Python program. Find potential sources of errors in this code (Hint:3 errors).

```
x = float(input("Please enter x: "))
y = float(input("Please enter y: "))
z = x / y
z_rounded = math.round(z,2)
```

- 2.2 Consider the following Python program. Find potential sources of errors in this code.

```
x = 0
for i in range(5):
    x *= 2

sum_squares = 0
for i in range(5):
    i_sq = i**2
    sum_squares += i_sq

total = 0
for i in range(10):
    i += i
```

3. Lab 6 Submission Exercises

Exercise 3.1

Reminder: Testing weighted 10% in your final mark for the third component of the assignment.

Create a test table for the Bank exercise in Week 3.

- Refer to Lab 3 exercise 2 for the requirements.
<https://moodle.bcu.ac.uk/mod/resource/view.php?id=7532565>
- Refer to the lecture notes for the table design.

Exercise 3.2

1. Create a script named **lab6_contacts.py** which contains a simple contact app.

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

2. Your program will start with this code:

3. This contact app displays the following menu:

```
Select an operation:
v view contacts
a add contact
d delete contact
q quit

Enter choice(v/a/d/q- to quit):
```

4. Based on the user choice, the program will do the following:

selected "v"	<pre>Enter choice(v/a/d/q- to quit): v ----- view_contacts ----- 1 Stish 123 2 Rita 321 The contact app should have 2 pre-defined contacts.</pre>	For display format, the number of spaces does not matter.
selected "a"	<pre>Enter choice(v/a/d/q- to quit): a ----- add_contact ----- Enter contact name: Sun Enter contact number: 456 Sun - 456 has been added into the contact.</pre>	<p>After entered a new contact, you can verify the result by selecting "v".</p> <pre>Enter choice(v/a/d/q- to quit): v ----- view_contacts ----- 1 Stish 123 2 Rita 321 3 Sun 456</pre>
selected "d"	<pre>Enter choice(v/a/d/q- to quit): d ----- delete_contact ----- 1 Stish 123 2 Rita 321 Enter the ID of the contact to delete: 1 Deleting: Record 1 Stish 123</pre>	<p>After deleted a contact, you can verify the result by selecting "v".</p> <pre>Enter choice(v/a/d/q- to quit): v ----- view_contacts ----- 1 Rita 321</pre> <p>Note. Since stish was deleted, the ID of Rita becomes 1 as she is the only contact in the app.</p>
Invalid options	<pre>Enter choice(v/a/d/q- to quit): haha ----- invalid_choice ----- Try again...</pre>	
<p>For all the choices above, after the operation finished, the contact app will redisplay the menu and awaits user to select an operation.</p> <pre>Select an operation: v view contacts a add contact d delete contact q quit Enter choice(v/a/d/q- to quit):</pre>		
selected "q"	<pre>Enter choice(v/a/d/q- to quit): q ----- goodbye -----</pre>	The contact app will exit and finish.

5. You can implement your contact app in your preferred way as long as it fulfils the following requirements:

- You program should start with the function main():

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

1. the main function should have a parameter with a default value as below:

2.

```
def main(contacts=[("Stish", "123"), ("Rita", "321")]):
```

- It behaves as step 2-4 specifie.
- It contains the following functions:

All functions below take a parameter contacts , which is of type list. The list is structured as below: [(name1, num1), (name2, num2), ..., (nameN, numN)] For the 2 pre-defined contacts, the list looks like this: [("Stish", "123"), ("Rita", "321")]	
view_contacts(contacts) for option "v"	<ul style="list-style-type: none"> • It displays (prints out) the list of contacts. • The contacts parameter remains unchanged.
add_contact(contacts) for option "a"	<ul style="list-style-type: none"> • It asks the user to enter the name and number of the new contact and then adds this contact into contacts. • The contacts parameters will be changed, and the new contact is added to the end of list.
delete_contact(contacts) for option "d"	<ul style="list-style-type: none"> • It removes the contact at the user's choice. • The contacts parameters will be changed. The chosen contact will be removed from the list.

4 Moodle submission

At the end of this lab session, zip the python file that you created in 3.2 and the word/pdf document with test case in 3.1 and then upload the zip file in the Moodle. **Note:** It is important to complete the weekly labs in particular labs 1, 2, 3, 4, 5 and 6 because it contains questions that are part of the coursework. Failure to complete and submit the answers to these labs on the Moodle may mean giving you zero on for this assessment component.

The zip file should be named using the following format StudentName_studentID.zip For example: OgertaElezaj_123456789.zip

5 Additional exercises

5.1 Write a program that reads integers from the user and stores them in a list. Use 0 as a sentinel value to mark the end of the input. Once all of the values have been read your program should display them in reverse order.

5.2 Create in Python the program which must:

- a. show the contents of the math module
- b. show the help of the functions used to perform the following calculations:
 - round up
 - exponentiation
 - logarithm
 - square root
 - absolute value

c. ask the user for an integer value (positive, zero or negative)

d. display the following calculation table:

Entered value: xxx

Calculations

- Absolute value: xxx

- Squared: xxx

- Cube: xxx

- Square root: xxx

- Natural logarithm: xxx

- Base 10 logarithm: xxx

e. prevent any error if the calculations are not feasible, displaying the message “Cannot be calculated”

5.3 Create in Python the program containing a function, called report, which creates a short report of the quantities of spare parts sold in three different stores (located in **London, Birmingham and Nottingham**). The function:

- **receives** three mandatory parameters (q1, q2 and q3) containing the quantities sold, respectively, in **London, Birmingham and Nottingham**.
- **receives** three optional parameters containing the price of the spare parts: € 60 for spare parts sold in **London**, € 57 for those sold in **Birmingham** and € 59 for those sold in **Nottingham**.
- **calculates** the revenue of each store, the average quantity sold and the average revenue.
- create a report similar to the example shown below, showing two decimal digits for revenues and one decimal digit for the average quantity sold.

The main program must ask the user for the quantities sold in each store and call the report function.

E.g.: entering quantities of 580 for London, 406 for Birmingham and 495 for Nottingham, the report should look like this:

Store	Quantity	Revenue [€]
London	580	34800.00
Birmingham	406	23142.00
Nottingham	495	29205.00
MEAN	493.7	29049.00

5.4 Create in Python the program Exercise13.8.py containing the tuple: tuplex = (4, 9, ['a','b'], 123.45, 0)

At the end of the program the tuple must be changed in the following way, applying the changes in the exact order given below (remember: tuples cannot be modified, they can only be reassigned, thus a list must be used to perform the transition from the initial tuple to the final one):

1. add value 7 at the end of the sequence
2. add tuple (10, 100, 1000) in the fourth position
3. add the string "bob" stored in the position with index 2
4. add number 3.5 in first position
5. add False in position -1
6. delete value 9
7. delete the element stored in the position with index -4

Display each change on the screen (printing each time the intermediate list) and, at last, print the final tuple tuplex.

5.5. Create in Python a program which receives a positive integer number from the user and displays the list of prime numbers smaller than the given value. The program must follow these guidelines:

- a. it must manage the entry of invalid inputs
- b. even numbers should be excluded from the check, since they cannot be prime numbers

E.g., if the user enters number 10, the result must be: “Prime list: 1, 2, 3, 5, 7”.