Debugging

UBCO Master of Data Science – DATA 530

Debugging: What's the Problem?

Debugging is the process of determining why a system does not work properly.

We perform debugging all the time in daily life, usually to fix problems with other systems and tools we interact with (cars, lights, appliances, electronics, our own bodies, etc.).

Debugging is different with computers and information technology because *usually* it is not a component failure that is the source of the problem. More commonly, it is our interaction and limited understanding of how the computer works.

Whose Problem Is It?

When we debug an information system, we are always part of the problem!

- We give the commands and the input, so the only other possible cause is a broken system.

People do not *knowingly* make errors, but we frequently do if we do not understand how to use a system properly.

- We must be precise and know what the computer expects. Computers are dumb… so we must be precise.

Debugging is challenging as a computer user because:

- the computer cannot debug itself
- we cannot debug it directly either because the error is internal to the computer

Debugging involves *working with* the computer to try and understand what is happening and why.

Entering Data into Forms

One way data is entered into a computer is using a form.

A programmer can restrict the types and number of symbols that can go into a form field.

- e.g. only allow numbers in a phone number field

Many errors occur when users either enter data that does not follow these restrictions, or they enter incorrect data that is accepted by the computer because it is not properly checked.

Debugging: Solving a Mystery

Debugging is very similar to solving a mystery.

To discover and solve the problem we ask questions like:

- Do I need more clues?
- Are my clues reliable?
- What is a theory to explain the problem?
- How can I test if my theory is correct?

Like solving mysteries, the way to get good at debugging is practice and gaining experience about common problems and solutions.

The Four Key Steps in Debugging

1) Check that the error is reproducible.
   - Computers are deterministic. Make sure you know exactly how to reproduce the error.
2) Do not jump to conclusions.
   - The actual cause of the error may be many steps removed from the visible symptoms.
3) Check all the "obvious" sources of error.
   - You would be surprised how often a cable is not plugged in…
4) Isolate the problem
   - The goal is to make good assumptions and divisions of parts that you know are working and others that need investigation.
   - Be careful! It is often parts (including yourself) that you assume are working that really are not.
     - Make sure assumptions are backed up by tests.

Home

PUBLIC

Stack Overflow

Tags

Users

FIND A JOB

Jobs

Companies

TEAMS What's this?

Free 30 Day Trial

How to create a Minimal, Reproducible Example

When asking a question, people will be better able to provide help if you provide code that they can easily understand and use to reproduce the problem. This is referred to by community members as creating a minimal, reproducible example (**reprex**), a minimal, complete and verifiable example (**mcve**), or a minimal, workable example (**mwe**). Regardless of how it's communicated to you, it boils down to ensuring your code that reproduces the problem follows the following guidelines:

Your code examples should be…

- …Minimal – Use as little code as possible that still produces the same problem
- …Complete – Provide all parts someone else needs to reproduce your problem *in the question itself*
- …Reproducible – Test the code you're about to provide to make sure it reproduces the problem

The rest of this help article provides guidance on these aspects of writing a minimal, reproducible example.

Types of Python Errors

Syntax errors - are identified when the code is run and will cause an error due to incorrect commands

Logic errors - occur when code has correct syntax but does not function as expected

Exceptions - occur when a run-time event causing an error is not handled by the code (try-except)

Steps in Debugging Python Code

1) Reproduce errors – Run the program several times to understand where the error is occurring.

2) Do not jump to conclusions – The line of code where the error is detected is not necessarily where the error is located.

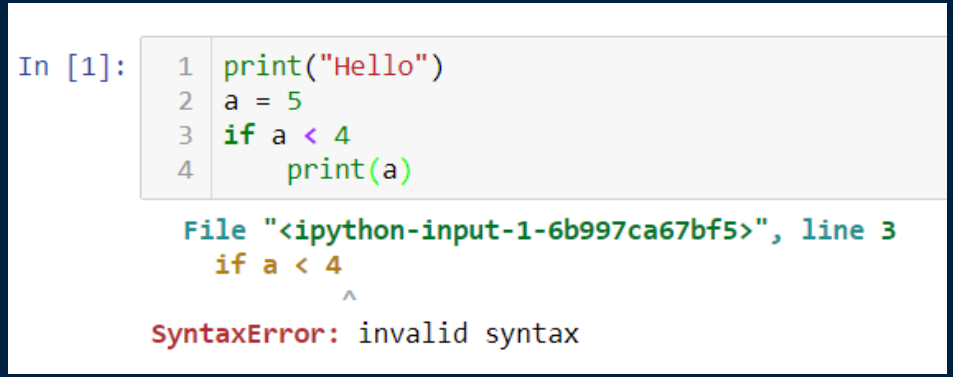
3) Obvious sources of errors – Fix syntax errors such as missing colons, improper indentation, etc. As you gain experience, more errors become obvious.

4) Isolate the problem – Use print statements and comment out code to isolate where the error occurs.

Q17

Python Debugging Question

Question: What type of error is this?



A) Exception **B)** Logic error **C)** Syntax error

Q18
Python Debugging Question

Question: This code works many times but not always. Why?

```
In [2]:  1  n = int(input("Enter a number: "))
         2  print(n)

Enter a number: 5
5
```

A) Exception **B)** Logic error **C)** Syntax error

Aside: The Cost of Debugging

When developing a computer system or application, the process of testing and debugging is extremely costly.

Most software requires 40% of the total time, cost, and effort to debug and fix problems in the system.

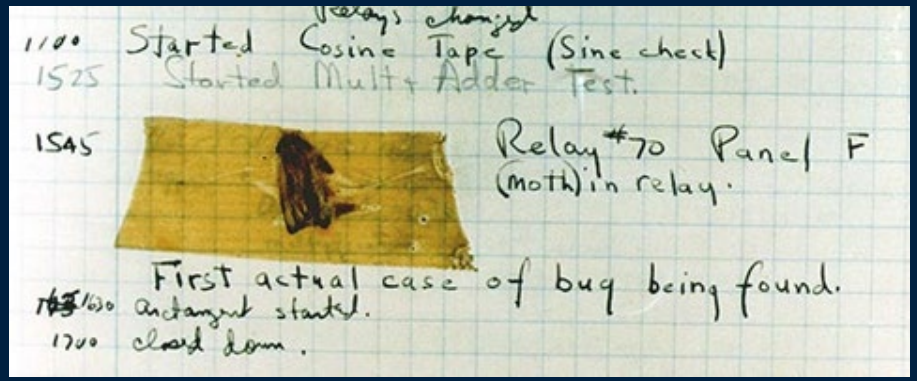
- Even so, many errors go unnoticed until the system is used.

To make software development more efficient and less costly, ***software engineering*** principles and techniques are followed.

- Although building software is harder than building a bridge due to its complexity, software engineers continually strive to make software development better.

Aside: The First Bug

The first "bug" in a computer system was actually a moth found in the Harvard Mark II computer system in 1947 by Rear Admiral Grace Hopper.



Source: U.S. Naval Historical Center Online Library Photograph NH 96566-KN

Debugging Follows the Scientific Method

Determining what a program does and finding any errors follows the scientific method.

1) ***Model*** – create a hypothesis on what the program does

2) ***Predict*** - for inputs not yet tried / simulated

3) ***Experiment*** – run the program to check your prediction

4) ***Refine*** – modify your hypothesis based on experimental results and repeat.

Understanding software is in many ways similar to understanding how complex real-world processes work.

Try it: Debug Question

Fix all errors in this code intended to print only even numbers from 1 to N where N is the number entered by the user.

```
1  n = int(input("Enter a number: "))
2  for n in range(1,n)
3      if n % 2 = 0
4      print(n)
```

pdb — The Python Debugger

Source code: Lib/pdb.py

The module `pdb` defines an interactive source code debugger for Python programs. It supports setting (conditional) breakpoints and single stepping at the source line level, inspection of stack frames, source code listing, and evaluation of arbitrary Python code in the context of any stack frame. It also supports post-mortem debugging and can be called under program control.

The debugger is extensible – it is actually defined as the class `Pdb`. This is currently undocumented but easily understood by reading the source. The extension interface uses the modules `bdb` and `cmd`.

The debugger's prompt is `(Pdb)`. Typical usage to run a program under control of the debugger is:

Conclusion

Debugging is a systematic approach to discover and fix errors in a system.

- Debugging a computer system requires working with the computer to diagnose the problem with the realization that we are often the cause of the problem.

The four key steps of debugging are:

- 1) Check that the error is reproducible.
- 2) Make sure you know what the problem is.
- 3) Check all the "obvious" sources of error.
- 4) Isolate the problem

Programming errors can be classified as syntax errors, logic errors, and exceptions.

Objectives

- Define: debugging
- List and explain the 4 key steps of debugging.
- List and identify three types of program errors: syntax errors, logic errors, and exceptions.
- Be able to debug Python and R code.

Getting Help

Getting help is important as remembering all details about a language and its packages is difficult.

The first source for help is the language's official documentation.

- Python: https://docs.python.org/3/
- R: https://cran.r-project.org/manuals.html

This documentation is often easily available in a development environment.

Online help is available on sites like Stack Overflow.

Python Documentation

The Python documentation provides reference material on the language and modules.

- Make sure to be using Python 3 documentation.

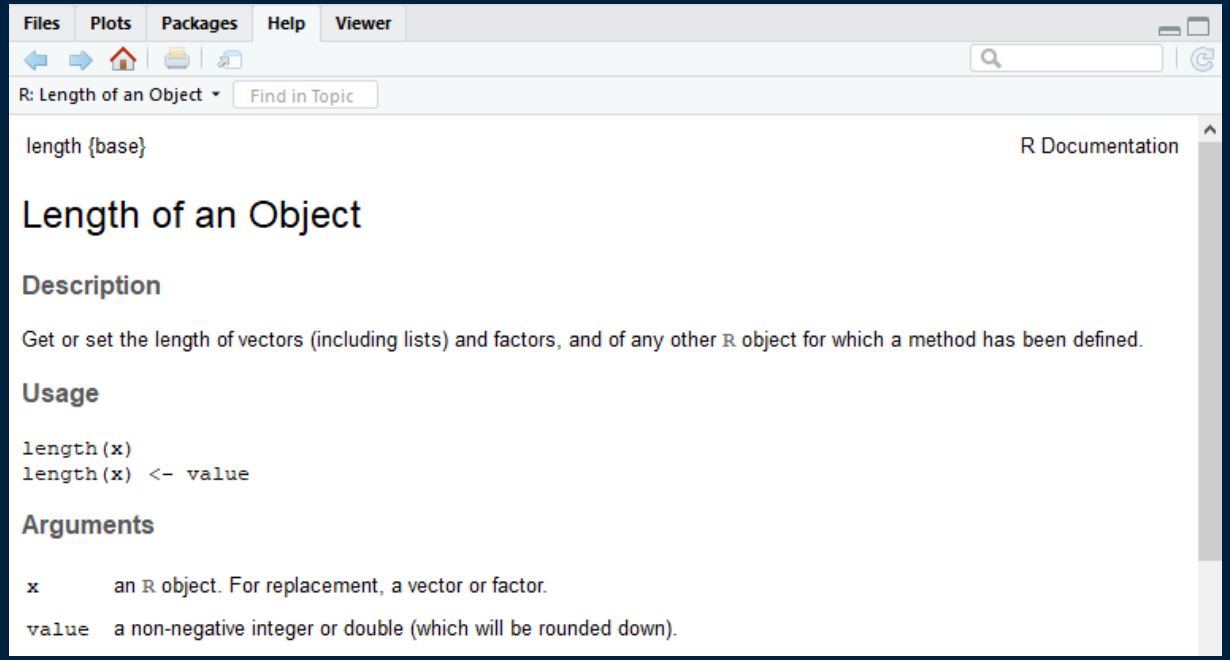
To access information about a function use: **`help(func)`**

Jupyter Notebook shortcut: **`Shift+Tab`**

R Documentation

The R documentation is available in RStudio.

To access information about a function use: **`help(func)`**



Python Libraries

Libraries are called ***modules*** in Python.

- A module is Python code stored in a file (with .py) and loaded when imported.

Modules are imported using the import command:

```
import modulename
```

A Python ***package*** is a collection of modules. Packages may have subpackages. Modules in a package are referenced by `package.module`.

Import module from package:

```
import package.subpackage.modulename
```

Python Import Syntax Examples

Import module X:

```
import X
```

Import module X in subpackage Y of package Z:

```
import Z.Y.X
```

Import module X and rename as N:

```
import X as N
```

Import function names F1, F2 from module Z.Y.X:

```
from Z.Y.X import F1, F2
```

Python Package Clicker Question

Question: Package `random` has a function called `random`. Given the following import, how would you call this function.

```
from random import random as rand
```

A) `random.random()`

B) `random.rand()`

C) `rand.random()`

D) `rand()`

E) `rand.random()`

Python Import Package Clicker Question

Question: Package `random` has a function called `random`. The following code successfully calls the function, what was the import?

```
rand.random()
```

A) `import random`

B) `from random import rand`

C) `from random import random as rand`

D) `import rand`

E) `import random as rand`

R Libraries

Libraries are called ***packages*** in R. There are thousands of packages.

- https://cran.r-project.org/web/packages/available_packages_by_name.html

Two steps:

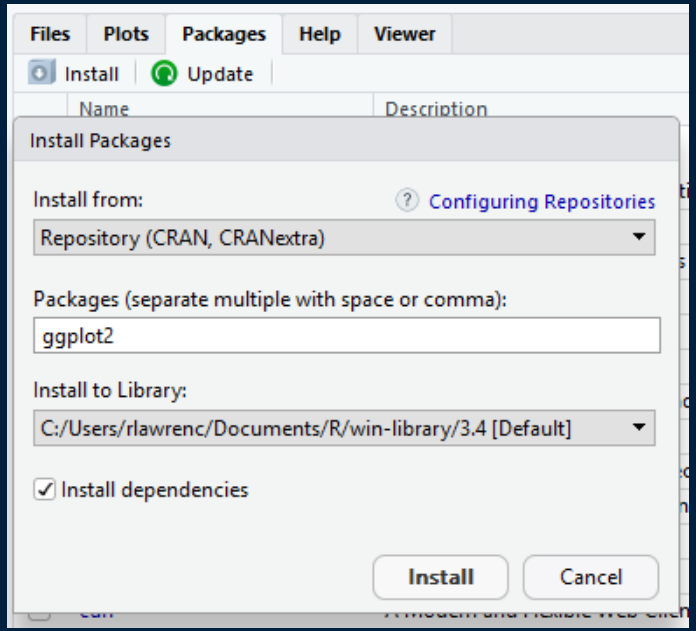
- 1) Install library into your environment using `install.packages()`.
- 2) Add library into current R session using the `library()` command.

Example:

```
install.packages("ggplot2")
library(ggplot2)
```

Adding Packages with RStudio

In RStudio, packages can be added by selecting `Packages` tab in Help section of editor and clicking on `Install`. Search by package name.



R Package Clicker Question

Question: How many of the following are **TRUE**?

1) An installed package is always available in a R session.

2) The `library()` command is used to install a package.

3) The `install.packages()` command makes a package available in the current R session.

4) There is no error if you use `library()` on the same package more than once in the current R session.

5) `install.packages()` handles package dependencies.

A) 1 **B)** 2 **C)** 3 **D)** 4 **E)** 5

Try it: Libraries in R and Python

In either R or Python (or both):

- Find a library for reading XML data.
- Read the persondata.xml file using the library.
- Output the number of people in the data set.
- Output the average (mean) person height.
- Output the max person weight.

Objectives

- Understand different resources and techniques for getting help in R and Python.
- Read and apply standard documentation for R and Python.
- Understand the importance of using libraries to improve code development efficiency and reliability.
- Practice finding a library, reading its documentation, and applying its methods to solve a problem.


THE UNIVERSITY OF BRITISH COLUMBIA