Introduction to Programming Language (ITP101)

Debugging

Mulualem Teku

GCIT, Bhutan

Oct 2019

...So Far & Today...

- Computational Thinking
- Core Python objects
  - Functions
  - Lists
  - Tuples
  - Dictionaries
  - Exceptions

Next...

- Bugs
- Debugging

Brainstorm

1. Bug? Etymology?

2. Some popular bugs you have heard of?

3. Debugging vs Exception handling? The how-to of debugging?

Brainstorm

1. Bug? Etymology?

2. Some popular bugs you have heard of?

3. Debugging vs Exception handling? The how-to of debugging?

Brainstorm

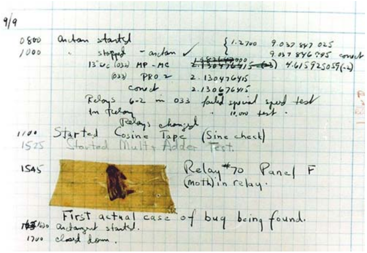
1. Bug? Etymology?

2. Some popular bugs you have heard of?

3. Debugging vs Exception handling? The how-to of debugging?

Debugging: The Whats

- The *process* of finding and reducing the number of bugs in a computer program. *Debugger* is the tool used.
- The process of ascertaining why the program is not working.
- Origin: Grace Hopper's moth bug



```
9/9

0800  Arctan started
1000    "    stopped - arctan ✓         { 1.2700   9.037 847 025
                                                 9.037 846 995 conct
       13"uc (032) MP-MC   ~~1.982147000~~
                           ~~2.130476415~~ (-3))  4.615925059(-2)
             (033) PRO 2   2.130476415
                   conct   2.130676415
       Relays 6-2 in 033 failed special speed test
       In relay          .   11,000 test .
             Relays changed
1100  Started Cosine Tape (Sine check)
1525  Started Mult+ Adder Test.
1545                    Relay #70 Panel F
                        (moth) in relay.
          First actual case of bug being found.
1630  arctangent started.
1700  closed down.
```

- Debugging vs exception handling vs testing

Why Debug?

- Errors are inevitable and cost nations billions of $s.

  "Yeah, but I already know exception handling and that should suffice, right?"

- Exception handling ensures that when your program encounters an issue, it will continue to run and provide informative feedback to the user.
- The principle: any erroneous code must be debugged!
- The fact: debugging time $>>$ development time!
  - Debugging large programs can be difficult and frustrating without the skill (of techniques, tools...).
- Debugging skill pays off big time.

Who Said What?

Edsger W. Dijkstra

> *"If debugging is the process of removing software bugs, then programming must be the process of putting them in."*
>
> *"Testing can only prove the presence of bugs, not their absence."*

Who Said What?

Edsger W. Dijkstra

> *"If debugging is the process of removing software bugs, then programming must be the process of putting them in."*
>
> *"Testing can only prove the presence of bugs, not their absence."*

Brian Kernighan (C language)

> *"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."*

Who Said What?

Edsger W. Dijkstra

> *"If debugging is the process of removing software bugs, then programming must be the process of putting them in."*
>
> *"Testing can only prove the presence of bugs, not their absence."*

Brian Kernighan (C language)

> *"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."*

Donald Knuth

> *"Beware of bugs in the above code; I have only proved it correct, not tried it."*

Debugging Techniques

Some of them:

1. *Rubber Duck Debugging*

2. *Tracing (a.k.a. Print Debugging)*

   Watching `print` statements (e.g. values of variables)

3. *Post-Mortem Debugging*

   After the program has crashed (e.g. memory/core dump analysis)

4. *Remote Debugging*

Debuggers

- Provide extensive view of the program's execution process.
- Enable a programmer to trace program execution step-by-step and "kill" the bug.
- Can be language-dependent. Some debuggers:

  `pdb` (Python), gdb, `Idb`, visual debuggers (e.g. `MS Visual Studio`), etc

Some features:

- Stepping through the code
- Pausing at some point in the code and resuming (breaking and continuing)
- Stack inspection, etc

Python Debugger (pdb)

- `Pdb` is an interactive debugger that is made available as a module.
- Provides a full-fledged debugging environment with support for:
  - Breakpoints
  - Source code listing
  - Single stepping through source code
  - Stack inspection
  - Post-mortem debugging

  and many more ...
- Can be used in two modes:
  1. Embedding debugging routines in source code
  2. Launching debugger as a script w/out embedding it:

```
python -m pdb <program_name>
```

Pdb: Operations & Commands

First off, import the pdb module.

1) **Setting Breakpoints**

   Program execution pauses at the specified points & tracing begins.

   a. Insert `pdb.set_trace()` where you want tracing to begin.

   b. The `break ("b")` command

Breakpoints

```
b(reak) ([program_name:]lineno | function)
```

Example

```
break myprog.py:3          # break @ line 3 of myprog.py
b myprog.func              # break when func() of myprog.py is called
```

Pdb: Operations & Commands (2)

2) **Displaying the source code** → The `list ("l")` command

3) **Stepping through the code** → The `next ("n")` command

4) **Printing variables' values** → The `print ("p")` command

5) **Continuing program execution** → The `continue ("c")` command

Pdb: Operations & Commands (2)

2) **Displaying the source code** → The `list ("l")` command

3) **Stepping through the code** → The `next ("n")` command

- Executes the current line and moves to the next.

4) **Printing variables' values** → The `print ("p")` command

5) **Continuing program execution** → The `continue ("c")` command

Operations & Commands (2)

2) **Displaying the source code** → The `list ("l")` command

3) **Stepping through the code** → The `next ("n")` command

   - Executes the current line and moves to the next.

4) **Printing variables' values** → The `print ("p")` command

5) **Continuing program execution** → The `continue ("c")` command

Operations & Commands (2)

2) **Displaying the source code** → The `list ("l")` command

3) **Stepping through the code** → The `next ("n")` command

   - Executes the current line and moves to the next.

4) **Printing variables' values** → The `print ("p")` command

5) **Continuing program execution** → The `continue ("c")` command

   - Lets the program continue execution and stop only when a breakpoint is encountered.

Pdb: Operations & Commands (3)

6) **Stepping into functions** → The `step ("s")` command

7) **Continuing until return of functions** → The `return ("r")` command

8) **Restarting the debugged Program** → The `run` or `restart` commands

9) **Getting help** → The `help ("h")` command

10) **Quitting altogether** → The `quit ("q")` command

Pdb: Operations & Commands (3)

6) **Stepping into functions** → The `step ("s")` command

7) **Continuing until return of functions** → The `return ("r")` command

8) **Restarting the debugged Program** → The `run` or `restart` commands

9) **Getting help** → The `help ("h")` command

10) **Quitting altogether** → The `quit ("q")` command

Pdb: Operations & Commands (3)

6) **Stepping into functions** → The `step ("s")` command

7) **Continuing until return of functions** → The `return ("r")` command

8) **Restarting the debugged Program** → The `run` or `restart` commands

9) **Getting help** → The `help ("h")` command

10) **Quitting altogether** → The `quit ("q")` command

Pdb: Operations & Commands (3)

6) **Stepping into functions** → The `step ("s")` command

7) **Continuing until return of functions** → The `return ("r")` command

8) **Restarting the debugged Program** → The `run` or `restart` commands

9) **Getting help** → The `help ("h")` command

10) **Quitting altogether** → The `quit ("q")` command

Operations & Commands (3)

6) **Stepping into functions** → The `step ("s")` command

7) **Continuing until return of functions** → The `return ("r")` command

8) **Restarting the debugged Program** → The `run` or `restart` commands

9) **Getting help** → The `help ("h")` command

10) **Quitting altogether** → The `quit ("q")` command