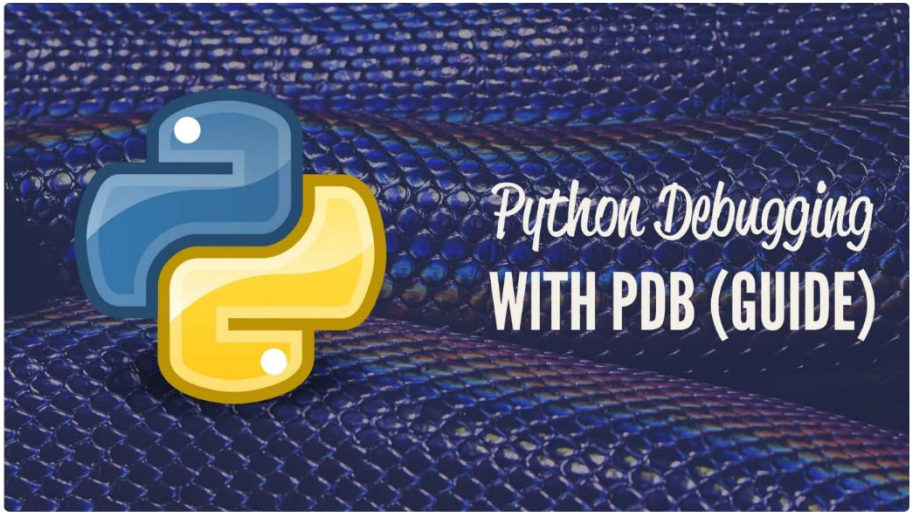
Lecture 14 – Practical Examples

https://cw.fel.cvut.cz/wiki/courses/be5b33prg/start

Tomas Jenicek

Czech Technical University in Prague,
Faculty of Electrical Engineering, Dept. of Cybernetics,
Center for Machine Perception
http://cmp.felk.cvut.cz/~jenicto2/
tomas.jenicek@fel.cvut.cz



Python Debugging With Pdb

by Nathan Jennings · Apr 09, 2018 · 1 Comment · intermediate · python · tools

Table of Contents

- Debugging in Python (**pdb**) vs. debugging in PyCharm (**IDE**)
- Reference for pdb debugging: https://realpython.com/python-debugging-pdb/

source https://realpython.com/python-debugging-pdb/

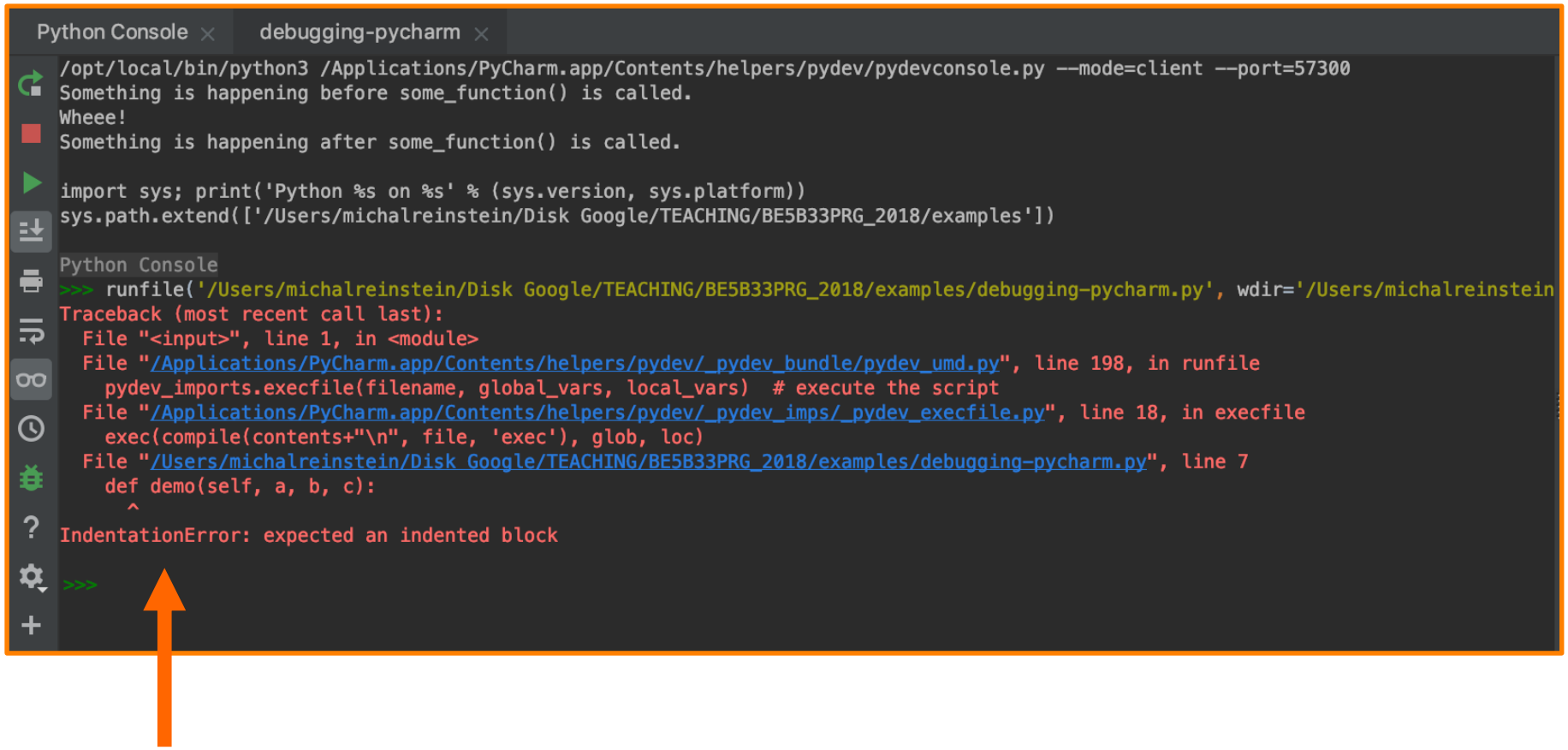
```python
import math


class Solver:


def demo(self, a, b, c):
    d = b ** 2 - 4 * a * c
    if d > 0:
        disc = math.sqrt(d)
        root1 = (-b + disc) / (2 * a)
        root2 = (-b - disc) / (2 * a)
        return root1, root2
    elif d == 0:
        return -b / (2 * a)
    else:
        return "This equation has no roots"
```

```python

if __name__ == '__main__':
    solver = Solver()

while True:
    a = int(input("a: "))
    b = int(input("b: "))
    c = int(input("c: "))
    result = solver.demo(a, b, c)
    print(result)

```

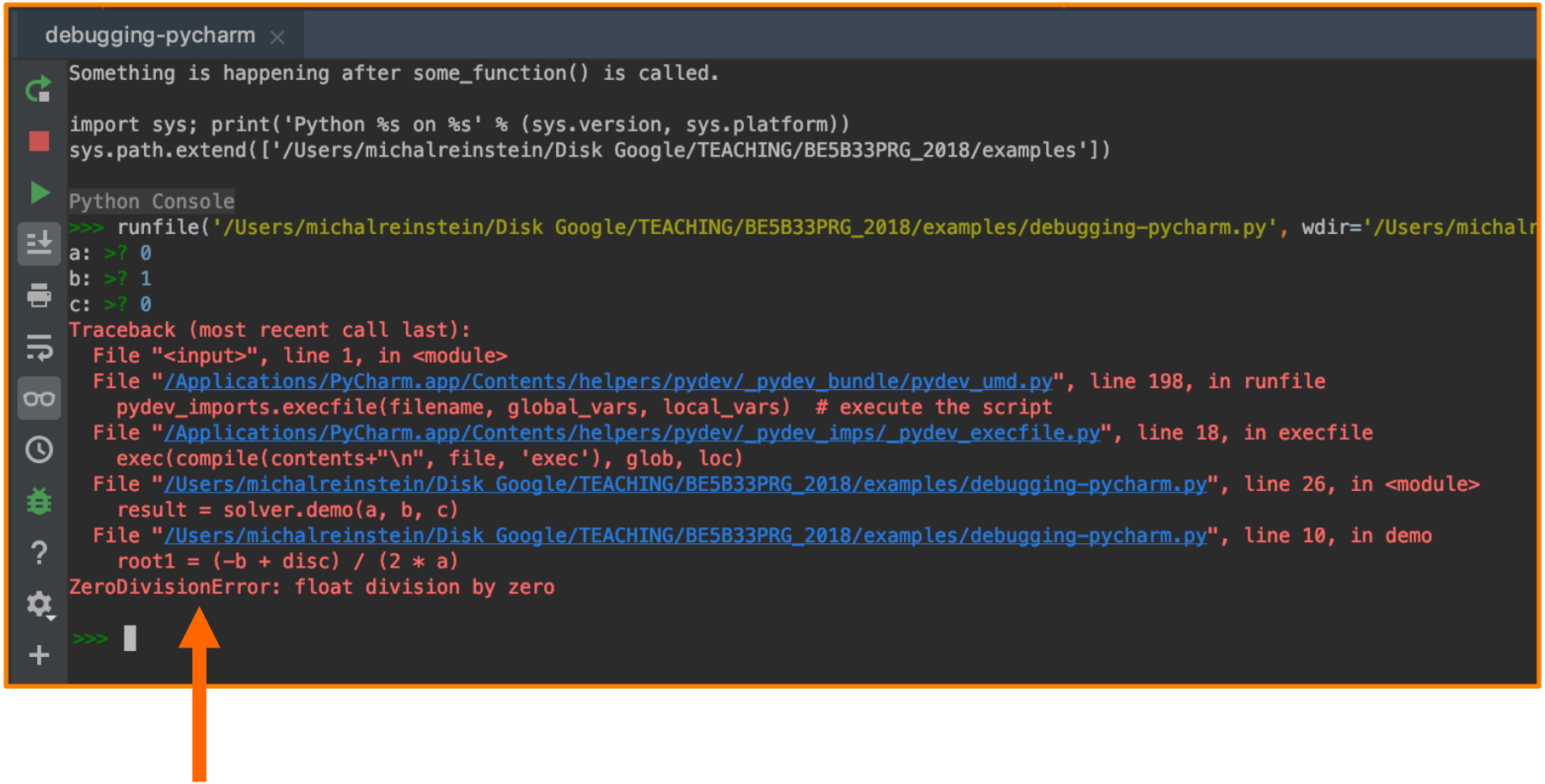
Quadratic formula - known as the A, B, C formula, it's used for solving a simple quadratic equation: **ax2 + bx + c = 0**
https://www.khanacademy.org/math/algebra/quadratics/solving-quadratics-using-the-quadratic-formula/a/quadratic-formula-explained-article

source https://www.jetbrains.com/help/pycharm/part-1-debugging-python-code.html

DEBUGGING



source https://www.jetbrains.com/help/pycharm/part-1-debugging-python-code.html

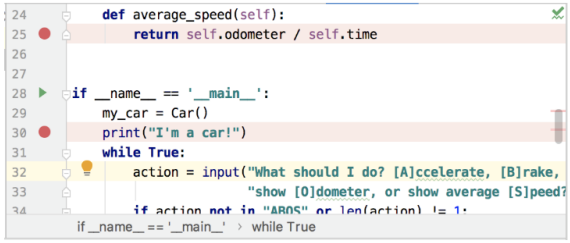


source https://www.jetbrains.com/help/pycharm/part-1-debugging-python-code.html

DEBUGGING – PLACING BREAKPOINTS


```python
import math


class Solver:


    def demo(self, a, b, c):
        d = b ** 2 - 4 * a * c
        if d > 0:
            disc = math.sqrt(d)
            root1 = (-b + disc) / (2 * a)
            root2 = (-b - disc) / (2 * a)
            return root1, root2
        elif d == 0:
            return -b / (2 * a)
        else:
            return "This equation has no roots"


if __name__ == '__main__':
    solver = Solver()

while True:
    a = int(input("a: "))
    b = int(input("b: "))
    c = int(input("c: "))
    result = solver.demo(a, b, c)
    print(result)

```

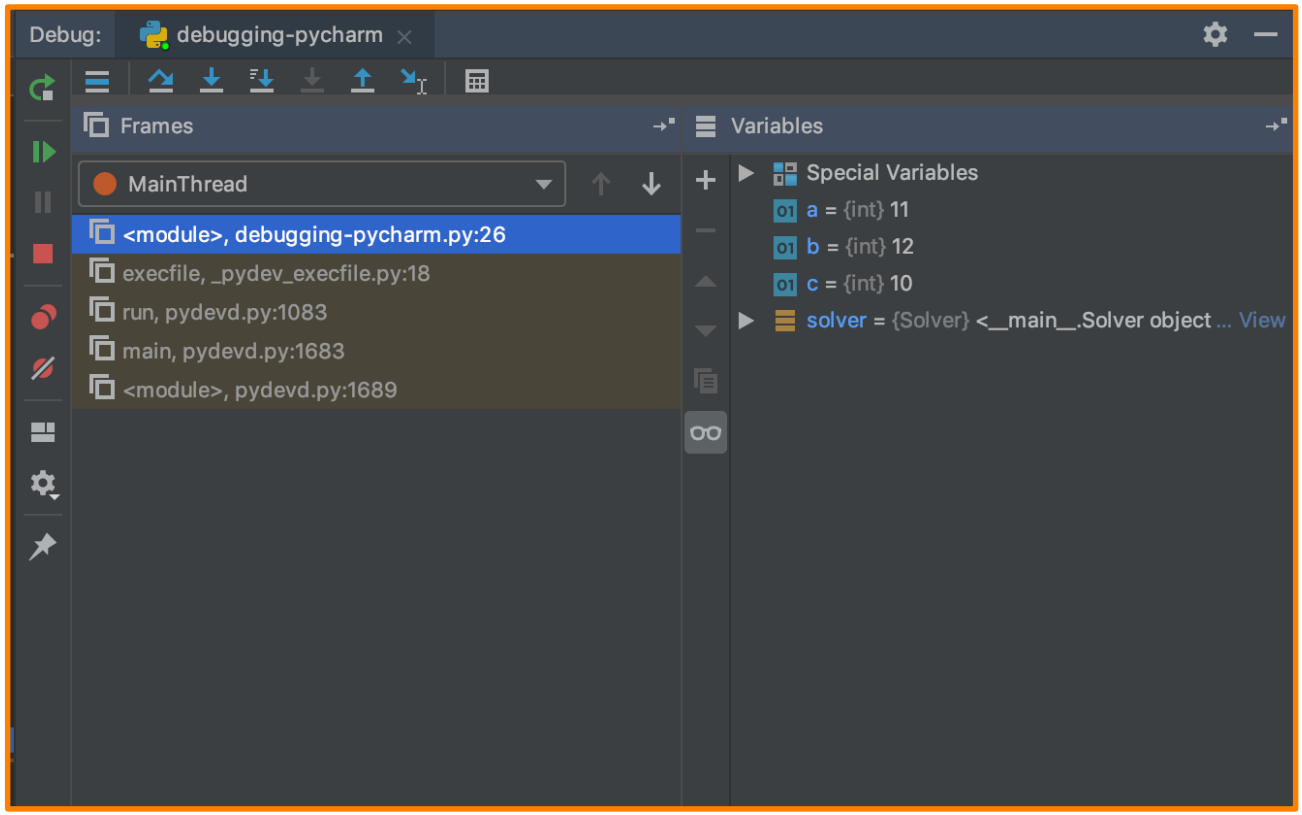
Breakpoints are source code markers that let you suspend program execution at a specific point and examine its behavior.



Once set, a breakpoint remains in your project until you remove it explicitly (except for temporary line breakpoints). If a file with breakpoints was modified externally, for example, updated through a VCS or changed in an external editor, and the line numbers have changed, breakpoints will be moved accordingly. Note that PyCharm must be running when such changes are made, otherwise they will pass unnoticed.

https://www.jetbrains.com/help/pycharm/using-breakpoints.html

source https://www.jetbrains.com/help/pycharm/part-1-debugging-python-code.html



Item	Tooltip and Shortcut
	Show Execution Point ⌥F10
	Step Over F8
	Step Into F7
	Force Step Into ⌥⇧F7
	Step Into My Code ⌥⇧F7
	Step Out ⇧F8
	Run to Cursor ⌥F9

Stepping toolbar

https://www.jetbrains.com/help/pycharm/debug-tool-window.html#steptoolbar

source https://www.jetbrains.com/help/pycharm/part-1-debugging-python-code.html

1. Syntax errors are produced by Python when it is translating the source code into byte code. They usually indicate that there is something wrong with the syntax of the program. Example: Omitting the colon at the end of a `def` statement yields the somewhat redundant message `SyntaxError: invalid syntax`.
2. Runtime errors are produced by the runtime system if something goes wrong while the program is running. Most runtime error messages include information about where the error occurred and what functions were executing. Example: An infinite recursion eventually causes a runtime error of maximum recursion depth exceeded.
3. Semantic errors are problems with a program that compiles and runs but doesn't do the right thing. Example: An expression may not be evaluated in the order you expect, yielding an unexpected result.

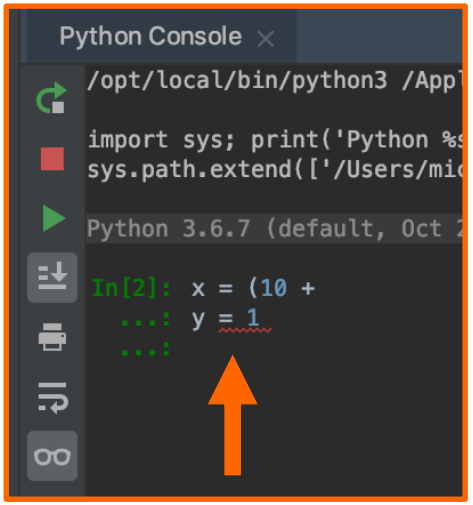
The first step in debugging is to figure out which kind of error you are dealing with:

1. **Syntax** errors
2. **Runtime** errors
3. **Semantic** errors

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

SYNTAX ERRORS


```
michalreinstein ~ python
Python 3.6.7 (default, Oct 21 2018, 08:56:20)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = (10 +
... y = 1
  File "<stdin>", line 2
    y = 1
      ^
SyntaxError: invalid syntax
>>>
```



- In case of syntax errors the error messages may not be often that helpful. The most common messages are:

SyntaxError: invalid syntax
SyntaxError: invalid token

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

- **Unterminated string** (especially multiline) may cause an invalid token error at the end of your program, or it may treat the following part of the program as a string until it comes to the next string. *In the second case, it might not produce an error message at all*!

- **Unclosed bracket** — (, {, or [— Python continues with the next line as part of the current statement. Generally, an error occurs almost immediately in the next line.

- **Comparison vs. assignment**, i.e. = instead of == inside a conditional.

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

SYNTAX ERRORS

- **Keywords** – double check **not** to use a Python keyword as variable name
- **Colon presence** – make sure to have a colon at the end of the header of every compound statement, including for, while, if, and def statements
- **Indentation consistency** – indentation must be consistent throughout the whole code; indent with either spaces or tabs but but do not mix both approaches; each level should be nested the same amount
- **String consistency** – strings in the code should have matching quotation marks, do not mix different styles

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

PROGRAM „HANGS“

1. If there is a particular loop that you suspect is the problem, add a `print` statement immediately before the loop that says entering the loop and another immediately after that says exiting the loop.
2. Run the program. If you get the first message and not the second, you've got an infinite loop. Go to the Infinite Loop section below.
3. Most of the time, an infinite recursion will cause the program to run for a while and then produce a RuntimeError: Maximum recursion depth exceeded error. If that happens, go to the Infinite Recursion section below.
4. If you are not getting this error but you suspect there is a problem with a recursive method or function, you can still use the techniques in the Infinite Recursion section.
5. If neither of those steps works, start testing other loops and other recursive functions and methods.
6. If that doesn't work, then it is possible that you don't understand the flow of execution in your program. Go to the Flow of Execution section below.

- Infinite loop
- Infinite recursion
- Flow of execution

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

GETTING EXCEPTIONS


```python
import math


class Solver:

    def demo(self, a, b, c):
        d = b ** 2 - 4 * a * c
        if d > 0:
            disc = math.sqrt(d)
            root1 = (-b + disc) / (2 * a)
            root2 = (-b - disc) / (2 * a)
            return root1, root2
        elif d == 0:
            return -b / (2 * a)
        else:
            return "This equation has no roots"


if __name__ == '__main__':
    solver = Solver()

    while True:
        a = int(input("a: "))
        b = int(input("b: "))
        c = int(input("c: "))
        result = solver.demo(a, b, c)
        print(result)

```

Error during runtime: Python prints a message that includes the **name of the exception**, the **line of the program** where the problem occurred, and a **traceback**

```
Python Console
>>> runfile('/Users/michalreinstein/Disk Google/TEACHING/BE5B33PRG_2018/examples/debugging-pycharm.py', wdir='/Users/mic
a: >? 10
b: >? 12
c: >? hi
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "/Applications/PyCharm.app/Contents/helpers/pydev/_pydev_bundle/pydev_umd.py", line 198, in runfile
    pydev_imports.execfile(filename, global_vars, local_vars)  # execute the script
  File "/Applications/PyCharm.app/Contents/helpers/pydev/_pydev_imps/_pydev_execfile.py", line 18, in execfile
    exec(compile(contents+"\n", file, 'exec'), glob, loc)
  File "/Users/michalreinstein/Disk Google/TEACHING/BE5B33PRG_2018/examples/debugging-pycharm.py", line 25, in <module>
    c = int(input("c: "))
ValueError: invalid literal for int() with base 10: 'hi'
```

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

GETTING EXCEPTIONS

- **Using breakpoints** – Put a breakpoint on the line causing the exception and explore the state of flow and variables
- **Explore the traceback** – The traceback identifies the function that is currently running, and then the function that invoked it, and then the function that invoked that function (*tracing the complete list of invocations up to the critical point*) including the line number in your file where each of these calls occurs.
- **Explore different levels on the stack** – The first step is to examine the place in the program where the error occurred and then trace the origin

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

GETTING EXCEPTIONS

NameError

You are trying to use a variable that doesn't exist in the current environment. Remember that local variables are local. You cannot refer to them from outside the function where they are defined.

TypeError

There are several possible causes:

1. You are trying to use a value improperly. Example: indexing a string, list, or tuple with something other than an integer.
2. There is a mismatch between the items in a format string and the items passed for conversion. This can happen if either the number of items does not match or an invalid conversion is called for.
3. You are passing the wrong number of arguments to a function or method. For methods, look at the method definition and check that the first parameter is `self`. Then look at the method invocation; make sure you are invoking the method on an object with the right type and providing the other arguments correctly.

- NameError
- TypeError

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

GETTING EXCEPTIONS

KeyError

You are trying to access an element of a dictionary using a key value that the dictionary does not contain.

AttributeError

You are trying to access an attribute or method that does not exist.

IndexError

The index you are using to access a list, string, or tuple is greater than its length minus one. Immediately before the site of the error, add a `print` statement to display the value of the index and the length of the sequence. Is the sequence the right size? Is the index the right value?

- KeyError
- AttributeError
- IndexError

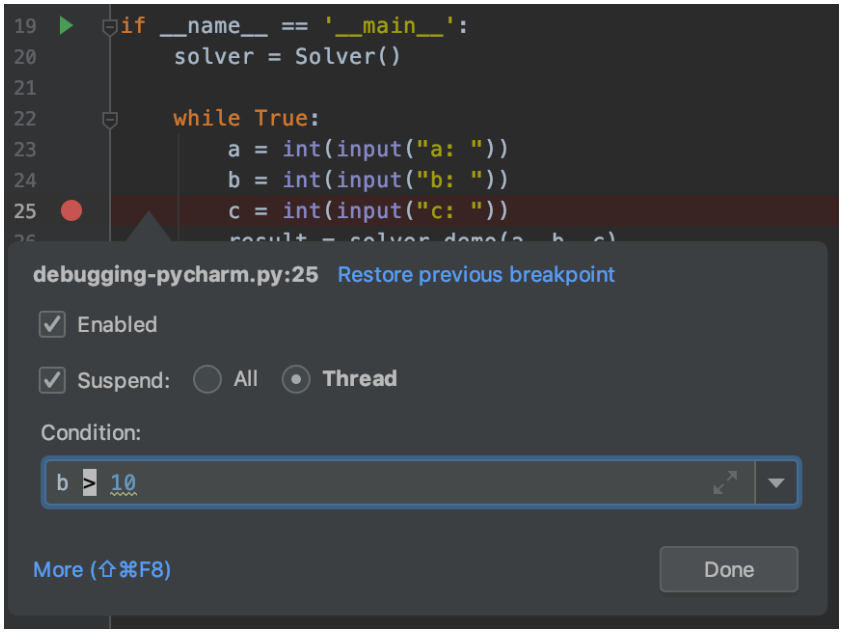
source http://openbookproject.net/thinkcs/python/english3e/app_a.html

SEMANTIC ERRORS

- **Semantic errors are the hardest** – The compiler and the runtime system provide no information about what is wrong.
- **Formulate hypothesis & validate** – The first step is to make a connection between the program text and the behavior; formulate hypothesis about what the program is actually doing.

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

SEMANTIC ERRORS



- Placing print statements and break points (**conditional break points**)
- Use **trivial data** (back to basics) or **dummy inputs** to simulate desired behavior as well as failures
- Walking the program **step-by-step**
- Writing **unit tests** and **integration tests** to avoid future breaking changes (*testing is our save game button!*)

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

DEBUGGING – SIMPLE vs. COMPLEX

If you have a `return` statement with a complex expression, you don't have a chance to print the `return` value before returning. Again, you can use a temporary variable. For example, instead of:

```
return self.hands[i].remove_matches()
```

you could write:

```
count = self.hands[i].remove_matches()
return count
```

1. Start with **simple**, better **readable** and more **verbose** code
2. Verify the **functionality**
   (solve *syntax*, *runtime* and *semantic* errors)
3. Write **tests** to ensure the code does not break with changes
4. Perform **refactoring** and optimization

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

```
1  self.hands[i].add_card(self.hands[self.find_neighbor(i)].pop_card())
```

This can be rewritten as:

```
1  neighbor = self.find_neighbor (i)
2  picked_card = self.hands[neighbor].pop_card()
3  self.hands[i].add_card(picked_card)
```

- **Complex expressions** – Writing complex expressions is fine as long as they are readable; try to break a complex expression into a series of assignments to temporary variables

- **Explicit vs. implicit** – The explicit version is easier to read because the variable names provide additional documentation; easier to debug because the types of the intermediate variables can be inspected for correct values

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- **Incremental development** technique – avoid long debugging sessions by adding and testing only a small amount of code at a time.

- EXAMPLE: *We want to find the distance between two points, given by the coordinates (x1, y1) and (x2, y2).*
  *(**Pythagorean theorem**)*

  What are the inputs (parameters)?
  What is the output (return value)?

source http://openbookproject.net/thinkcs/python/english3e/fruitful_functions.html

RECAP: PROGRAM DEVELOPMENT

Define interface

```
1  def distance(x1, y1, x2, y2):
2      return 0.0
```

```
>>> distance(1, 2, 4, 6)
0.0
```

Process parameters

```
1  def distance(x1, y1, x2, y2):
2      dx = x2 - x1
3      dy = y2 - y1
4      return 0.0
```

```
>>> distance(1, 2, 4, 6)
0.0
```

Temporary variables

```
1  def distance(x1, y1, x2, y2):
2      dx = x2 - x1
3      dy = y2 - y1
4      dsquared = dx*dx + dy*dy
5      return 0.0
```

```
>>> distance(1, 2, 4, 6)
0.0
```

Return result

```
1  def distance(x1, y1, x2, y2):
2      dx = x2 - x1
3      dy = y2 - y1
4      dsquared = dx*dx + dy*dy
5      result = dsquared**0.5
6      return result
```

```
>>> distance(1, 2, 4, 6)
5.0
```

source http://openbookproject.net/thinkcs/python/english3e/fruitful_functions.html

Another problem that can occur with big expressions is that the order of evaluation may not be what you expect. For example, if you are translating the expression `x/2pi` into Python, you might write:

```
y = x / 2 * math.pi
```

That is not correct because multiplication and division have the same precedence and are evaluated from left to right. So this expression computes `(x/2)pi`.

A good way to debug expressions is to add parentheses to make the order of evaluation explicit:

```
y = x / (2 * math.pi)
```

- **Use parentheses** – Whenever the order of evaluation is not clear, use parentheses. This way mistakes will be avoided and the code will be more readable especially *for those who did not memorize the rules of precedence …*

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

DEBUGGING – CALL FOR HELP!

- Asking for **help** is essential, https://stackoverflow.com/ is the best friend, you can learn a lot from the mistakes of others!
- Make sure you **ask the right question**! (not easy)
- When you bring someone in to help, be sure to give them the **right & complete information** they need:
  - If there is an error message, what is it and what part of the program does it indicate?
  - What was the last thing you did before errors occurred?
  - What version of OS and packages do you use?
  - What were the last lines of code that you wrote, or what is the new test case that fails?
  - What have you tried so far, and what have you learned?

source http://openbookproject.net/thinkcs/python/english3e/app_a.html

Tip: `None` is not a string

Values like `None`, `True` and `False` are not strings: they are special values in Python, and are in the list of keywords we gave in chapter 2 (Variables, expressions, and statements). Keywords are special in the language: they are part of the syntax. So we cannot create our own variable or function with a name `True` — we'll get a syntax error. (Built-in functions are not privileged like keywords: we can define our own variable or function called `len`, but we'd be silly to do so!)

There are two kinds of functions:

- **fruitful**, or value-returning functions, which calculate and return a value that we want
- **void** (non-fruitful) functions that perform actions that we want done

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

TIPS & HINTS: FUNCTIONS

Tip: Understand what the function needs to return

Perhaps nothing — some functions exists purely to perform actions rather than to calculate and return a result. But if the function should return a value, make sure all execution paths do return the value.

Tip: Use parameters to generalize functions

Understand which parts of the function will be hard-coded and unchangeable, and which parts should become parameters so that they can be customized by the caller of the function.

Tip: Try to relate Python functions to ideas we already know

In math, we're familiar with functions like `f(x) = 3x + 5`. We already understand that when we call the function `f(3)` we make some association between the parameter x and the argument 3. Try to draw parallels to argument passing in Python.

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

```
1  def any_odd(xs):  # Buggy version
2      """ Return True if there is an odd number in xs, a list of integers. """
3      for v in xs:
4          if v % 2 == 1:
5              return True
6          else:
7              return False
```

- Solve the problem: "**Does the list have any odd numbers?**"
- The logic "**If I find an odd number I can return True**" is fine.
- There are two issues (bugs) – which ones?

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

```
1  def any_odd(xs):  # Buggy version
2      """ Return True if there is an odd number in xs, a list of integers. """
3      for v in xs:
4          if v % 2 == 1:
5              return True
6          else:
7              return False
```

- **PROBLEM 1**: Cannot return False after only looking at one item — False can be returned only after all the items were explored, and none of them were odd
  (line 6 should not be there, line 7 has to be outside the loop)

- **PROBLEM 2**: ??

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

```
1  def any_odd(xs):  # Buggy version
2      """ Return True if there is an odd number in xs, a list of integers. """
3      for v in xs:
4          if v % 2 == 1:
5              return True
6          else:
7              return False
```

- **PROBLEM 1**: Cannot return False after only looking at one item — False can be returned only after all the items were explored, and none of them were odd
  (line 6 should not be there, line 7 has to be outside the loop)
- **PROBLEM 2**: Consider what happens if this function is called with an **argument that is an empty list:** any_odd([]), the for cycle ends immediately

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

```
1  def any_odd(xs):
2      """ Return True if there is an odd number in xs, a list of integers. """
3      for v in xs:
4          if v % 2 == 1:
5              return True
6      return False
```

- **PROBLEM 1**: Cannot return False after only looking at one item — False can be returned only after all the items were explored, and none of them were odd
  (line 6 should not be there, line 7 has to be outside the loop)
- **PROBLEM 2**: Consider what happens if this function is called with an **argument that is an empty list:** any_odd([]), the for cycle ends immediately

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

```
1  def any_odd(xs):
2      """ Return True if there is an odd number in xs, a list of integers. """
3      count = 0
4      for v in xs:
5          if v % 2 == 1:
6              count += 1     # Count the odd numbers
7      if count > 0:
8          return True
9      else:
10         return False
```

- Different solution …
- **PROBLEM 3: What is the disadvantage of this code?**

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

```
1  def any_odd(xs):
2      """ Return True if there is an odd number in xs, a list of integers. """
3      count = 0
4      for v in xs:
5          if v % 2 == 1:
6              count += 1     # Count the odd numbers
7      if count > 0:
8          return True
9      else:
10         return False
```

- Different solution …
- **PROBLEM 3:** The **performance disadvantage** of this one is that it traverses the whole list, even if it knows the results already

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

Tip: Think about the return conditions of the function

Do I need to look at all elements in all cases? Can I shortcut and take an early exit? Under what conditions? When will I have to examine all the items in the list?

The code in lines 7–10 can also be tightened up. The expression `count > 0` evaluates to a Boolean value, either `True` or `False`. The value can be used directly in the `return` statement. So we could cut out that code and simply have the following:

```
1  def any_odd(xs):
2      """ Return True if there is an odd number in xs, a list of integers. """
3      count = 0
4      for v in xs:
5          if v % 2 == 1:
6              count += 1    # Count the odd numbers
7      return count > 0      # Aha! a programmer who understands that Boolean
8                            #   expressions are not just used in if statements!
```

This **code is tighter** but it **is not as nice** as the one that did the short-circuit return as soon as the first odd number was found

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

TIPS & HINTS: BOOLEAN VALUES

```
Python Console
/opt/local/bin/python3.6 /Applications/PyCharm.app/Contents/helpers/pydev/pydevconsole.py 52255 52
Python 3.6.3 (default, Oct  5 2017, 23:34:28)
In[2]: n = 4  # divisible by 2
   ...: print(n % 2 == 0 or n % 3 == 0)
   ...:
True
In[3]: n = 6  # divisible by 2 and 3
   ...: print(n % 2 == 0 or n % 3 == 0)
   ...:
True
In[4]: n = 5  # not divisible by 2 or 3
   ...: print(n % 2 == 0 or n % 3 == 0)
   ...:
False
```

Tip: Generalize your use of Booleans

Mature programmers won't write `if is_prime(n) == True:` when they could say instead `if is_prime(n):` Think more generally about Boolean values, not just in the context of `if` or `while` statements. Like arithmetic expressions, they have their own set of operators (`and`, `or`, `not`) and values (`True`, `False`) and can be assigned to variables, put into lists, etc. A good resource for improving your use of Booleans is http://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_3/Boolean_Expressions

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

So if we need to know if “snake” occurs as a substring within `s`, we could write

```
1  if s.find("snake") >= 0:  ...
2  if "snake" in s: ...           # Also works, nice-to-know sugar coating!
```

It would be wrong to split the string into words unless we were asked whether the *word* “snake” occurred in the string.

Four really important operations on strings:

- **len(str)** finds the length of a string
- **str[i]** the subscript operation extracts the *i’th* character of the string, as a new string
- **str[i : j]** the slice operation extracts a substring out of a string
- **str.find(target)** returns the index where target occurs within the string, or -1 if it is not found

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

```
1  s = "..."                          # Get the next line from somewhere
2  def_pos = s.find("def ")           # Look for "def " in the line
3  if def_pos == 0:                   # If it occurs at the left margin
4    op_index = s.find("(")           # Find the index of the open parenthesis
5    fnname = s[4:op_index]           # Slice out the function name
6    print(fnname)                    # ... and work with it.
```

- Task is to read lines of some data (e.g. *python code file*), find *function definitions* and *print their names*
- The name of the function has to be isolated properly:

  def some_function_name(x, y):

- **There are a few issues with this solution …**

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

TIPS & HINTS: STRINGS

```
1  s = "..."                          # Get the next line from somewhere
2  def_pos = s.find("def ")           # Look for "def " in the line
3  if def_pos == 0:                   # If it occurs at the left margin
4    op_index = s.find("(")           # Find the index of the open parenthesis
5    fnname = s[4:op_index]           # Slice out the function name
6    print(fnname)                    # ... and work with it.
```

- **PROBLEM**: **What if the function def is indented and does not start at column 0**?

- The code needs adjustment to **detection of spaces** – make sure the characters in front of the *def_pos* position are spaces
- Handle **special cases** like comments:

 # I def initely like Python!

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

```
1  s = "..."                          # Get the next line from somewhere
2  def_pos = s.find("def ")           # Look for "def " in the line
3  if def_pos == 0:                   # If it occurs at the left margin
4    op_index = s.find("(")           # Find the index of the open parenthesis
5    fnname = s[4:op_index]           # Slice out the function name
6    print(fnname)                    # ... and work with it.
```

Verification of assumptions is necessary!

- **ASSUMPTION 1**: we assume on line 4 that we will find an **open parenthesis** – this should be checked that it was done!
- **ASSUMPTION 2**: we assume that there is **exactly one space** between the keyword def and the start of the function name; this will not work for multiple spaces: *def f(x)*

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

```python
import random
joe = random.Random()

def sum1():
    """ Build a list of random numbers, then sum them """
    xs = []
    for i in range(10000000):
        num = joe.randrange(1000)   # Generate one random number
        xs.append(num)              # Save it in our list

    tot = sum(xs)
    return tot
```

```python
def sum2():
    """ Sum the random numbers as we generate them """
    tot = 0
    for i in range(10000000):
        num = joe.randrange(1000)
        tot += num
    return tot

print(sum1())
print(sum2())
```

- Loops are a key feature for most of the programs: to *repeat computation*, *accurately* and *fast*
- **EXAMPLE**: Two functions *sum1* and *sum2* both **generate ten million random numbers** and **return their sum**; both work!
- **PROBLEM**: What is the key **performance difference**?

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

TIPS & HINTS: LOOPING EXAMPLE

```
1  import random
2  joe = random.Random()
3
4  def sum1():
5      """ Build a list of random numbers, then sum them """
6      xs = []
7      for i in range(10000000):
8          num = joe.randrange(1000)   # Generate one random number
9          xs.append(num)              # Save it in our list
10
11     tot = sum(xs)
12     return tot
13
14 def sum2():
15     """ Sum the random numbers as we generate them """
16     tot = 0
17     for i in range(10000000):
18         num = joe.randrange(1000)
19         tot += num
20     return tot
21
22 print(sum1())
23 print(sum2())
```

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

- Simple **performance monitoring**: Open a tool like the *Performance Monitor* (e.g. **htop** on Linux) on your computer, and watch the memory usage.

How big can you make the list before you get a fatal memory error in sum1?

- Similar when **working with files**: option to read the whole file contents into a single string, or read one line at a time and process each line as it is read.
  Line-at-a-time is the more traditional and safer way to do things — work comfortably no matter how large the file is.

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

TIPS & HINTS: LOOPING EXAMPLE


```python
import random
import time

rnd = random.Random()
max_range = 10000000


def sum_range():
    tot = 0
    for i in range(max_range):
        tot += i
    return tot


def sum1_random():
    """ Build a list of random numbers, then sum them """
    xs = []
    for i in range(max_range):
        num = rnd.randrange(1000)  # Generate one random number
        xs.append(num)  # Save it in our list

    tot = sum(xs)
    return tot


def sum2_random():
    """ Sum the random numbers as we generate them """
    tot = 0
    for i in range(max_range):
        num = rnd.randrange(1000)
        tot += num
    return tot


if __name__ == "__main__":
    t0 = time.clock()
    result = sum_range()
    t1 = time.clock()
    print(f'Sum of {max_range} numbers was computed in {t1-t0}s'
          f' and the results is {result}')
    print('-----------------------------------------------------------------------')
    t0 = time.clock()
    result = sum(range(max_range))
    t1 = time.clock()
    print(f'Sum of {max_range} numbers was computed in {t1-t0}s'
          f' and the results is {result}')
    print('-----------------------------------------------------------------------')
    # timing sum computed using built-in function
    t0 = time.clock()
    result = sum1_random()
    t1 = time.clock()
    print(f'Sum of {max_range} random numbers was computed in {t1-t0}s'
          f' and the results is {result}')
    print('-----------------------------------------------------------------------')
    # timing sum computed from definition
    t0 = time.clock()
    result = sum2_random()
    t1 = time.clock()
    print(f'Sum of {max_range} random numbers was computed in {t1-t0}s'
          f' and the results is {result}')
    print('-----------------------------------------------------------------------')
```

```
Terminal:  Local  +

1  [|||||||||||||||||||||||||||||||||                    39.9%]   Tasks: 462, 1078 thr; 1 running
2  [|||||||||||||||||                                    19.0%]   Load average: 2.46 2.60 2.41
3  [|||||||||||||||||||||||||||||||||                    33.6%]   Uptime: 2 days, 20:21:37
4  [|||||||||||||||                                      16.3%]
Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||8.49G/16.0G]
Swp[|||||||||||||||||||||||||||||||||||||||||                                      1.36G/3.00G]
```

source http://openbookproject.net/thinkcs/python/english3e/app_e.html

- https://www.jetbrains.com/help/pycharm/part-1-debugging-python-code.html
- https://realpython.com/python-debugging-pdb/
- https://hackernoon.com/prime-numbers-using-python-824ff4b3ea19
- http://openbookproject.net/thinkcs/python/english3e/app_a.html
- http://openbookproject.net/thinkcs/python/english3e/app_e.html
- http://openbookproject.net/thinkcs/python/english3e/app_b.html

This lecture re-uses selected parts of the OPEN BOOK PROJECT

Learning with Python 3 (RLE)

http://openbookproject.net/thinkcs/python/english3e/index.html

available under GNU Free Documentation License Version 1.3)

- Version date: October 2012
- by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers (based on 2nd edition by Jeffrey Elkner, Allen B. Downey, and Chris Meyers)
- Source repository is at https://code.launchpad.net/~thinkcspy-rle-team/thinkcspy/thinkcspy3-rle
- For offline use, download a zip file of the html or a pdf version from http://www.ict.ru.ac.za/Resources/cspw/thinkcspy3/