

CEBD 1100 Introduction to Data Analysis and Python

Conditions, Loops and Debugging





PyCharm

PyCharm Free Version

- Free (General)
 - Intelligent Editor
 - Graphical Debugger
 - Refactorings
 - Code Inspections
 - Version Control Integration
- Platform
 - XML, HTML, YAML, JSON, RelaxNG
 - Git, Mercurial, CVS, Subversion, GitHub
 - IntelliLang
 - Local terminal
 - Task management
- Python Framework and Tools
 - Core Python language support
 - Code Inspections
 - Refactoring
 - Local debugger
 - Test runners
 - reStructuredText support
 - PyQt
 - PyGTK
 - Package management
 - Virtualenv/Buildout
 - Python console
 - IPython Notebook

Extra Features in PyCharm Pro

- Web development with JavaScript, CoffeeScript, TypeScript, HTML/CSS and more
- Frameworks: Django, Flask, Google App Engine, Pyramid, web2py
- Remote development capabilities: Remote run/debug, VM support
- Database & SQL support
- UML & SQLAlchemy Diagrams
- Scientific Tools
- Code Coverage
- Django
- Thread Concurrency Validation
- Platform
 - CSS/HAML/SASS/LESS/Stylus
 - Database/SQL
 - JavaScript and JS Debugger
 - Perforce, TFS
 - FTP/SFTP/FTPS remote host deployment
 - TextMate bundles
 - REST Client
 - Puppet
 - File watchers

Student Edition of PyCharm

- If you provide proof that you are a student, you can get a fully functional Pycharm (and IntelliJ) which is similar to the Pro version.
- See : <https://www.jetbrains.com/education/download/#section=pycharm-edu>

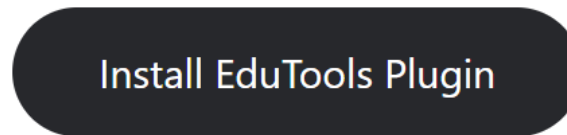
PyCharm Edu is free & open source.

Licensed under Apache License, Version 2.0.



PyCharm Edu

or



If you have already installed PyCharm
Professional or Community



Version: 2019.3
Build: 193.5233.132
3 December 2019

[Release notes](#)

File Edit View Navigate Code Refactor Run Tools VCS Window Help experimenting [D:\Code\Python\experimenting] - ...list_test.py - PyCharm

experimenting > list_test.py

Project Files / Files

These are files needed to run your program (like the interpreter).

Run the Last Run File

Debug Last Debugged File

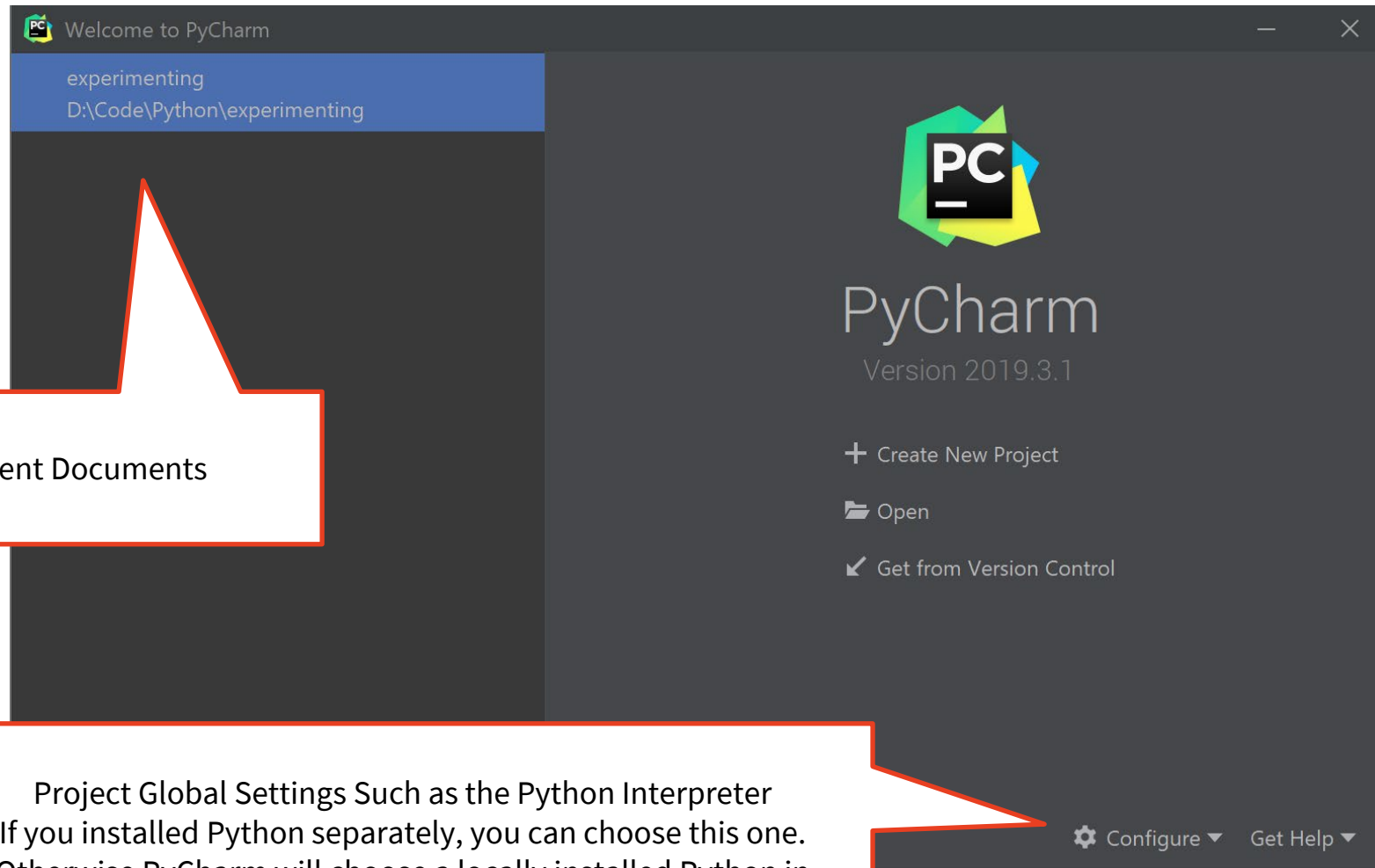
Live Search / Shortcut: SHIFT SHIFT

```
1 import decimal
2
3 from money import Money
4
5 item_original_currency = "USD"
6 item_original_price = 19.99
7 item_name = "Bluray Movie"
8 item_purchase_currency = "CAD"
9
10
11
12
13 exchange_rates = {
14     "AED": 3.6733,
15     "AFN": 76.05,
16     "ALL": 110.1196,
17     "AMD": 486.9206,
18     "ANG": 1.8392,
19     "AOA": 316.854,
20     "ARS": 41.025,
21     "AUD": 1.4099,
22     "AWG": 1.801,
23     "AZN": 1.705,
24     "BAM": 1.7205,
25     "BBD": 1.9953,
26     "BDT": 84.232,
27     "BGN": 1.73,
28     "BHD": 0.3771,
29     "BIF": 1809.95,
30     "BMD": 1.
```

Database
Event Log
Favorites
Project
Python Console
SciView
Structure
Terminal
TODO

6: TODO Terminal Python Console

1245:1 CRLF UTF-8 4 spaces Python 3.7 (experimenting)









Intro to Debugging

Break Points

- A breakpoint is a position in your code where program execution stops.
- When stopped, we can investigate the values of other variables.
- There are different types of breakpoints that react differently (see next slide)

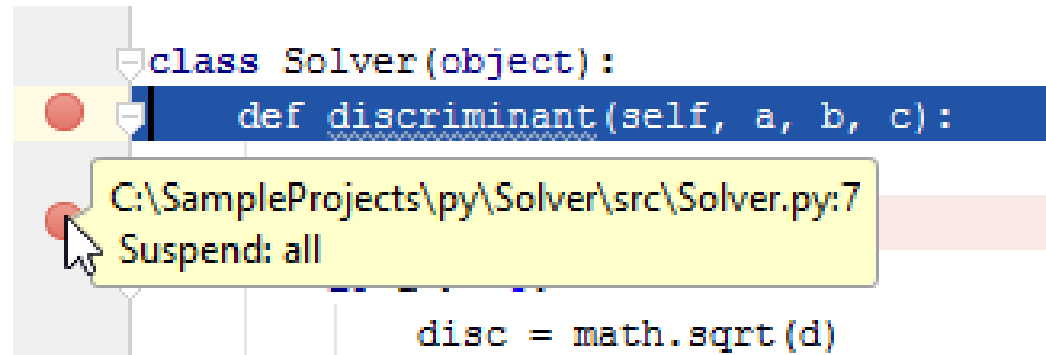
Breakpoint Summary

- We can disable a breakpoint if we know that we might need to use it later on for re-testing the application.

Status	Line Python / Django/ JavaScript	Temporary Line
Enabled		
Disabled		
Conditionally disabled		

Setting a breakpoint

- Click on the border of the code to set.
- ALT-Click to disable!

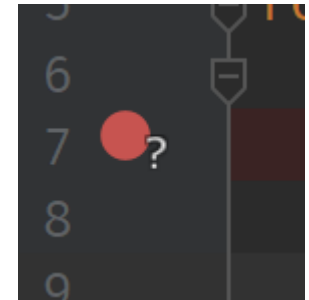


Key Combinations for Breakpoints

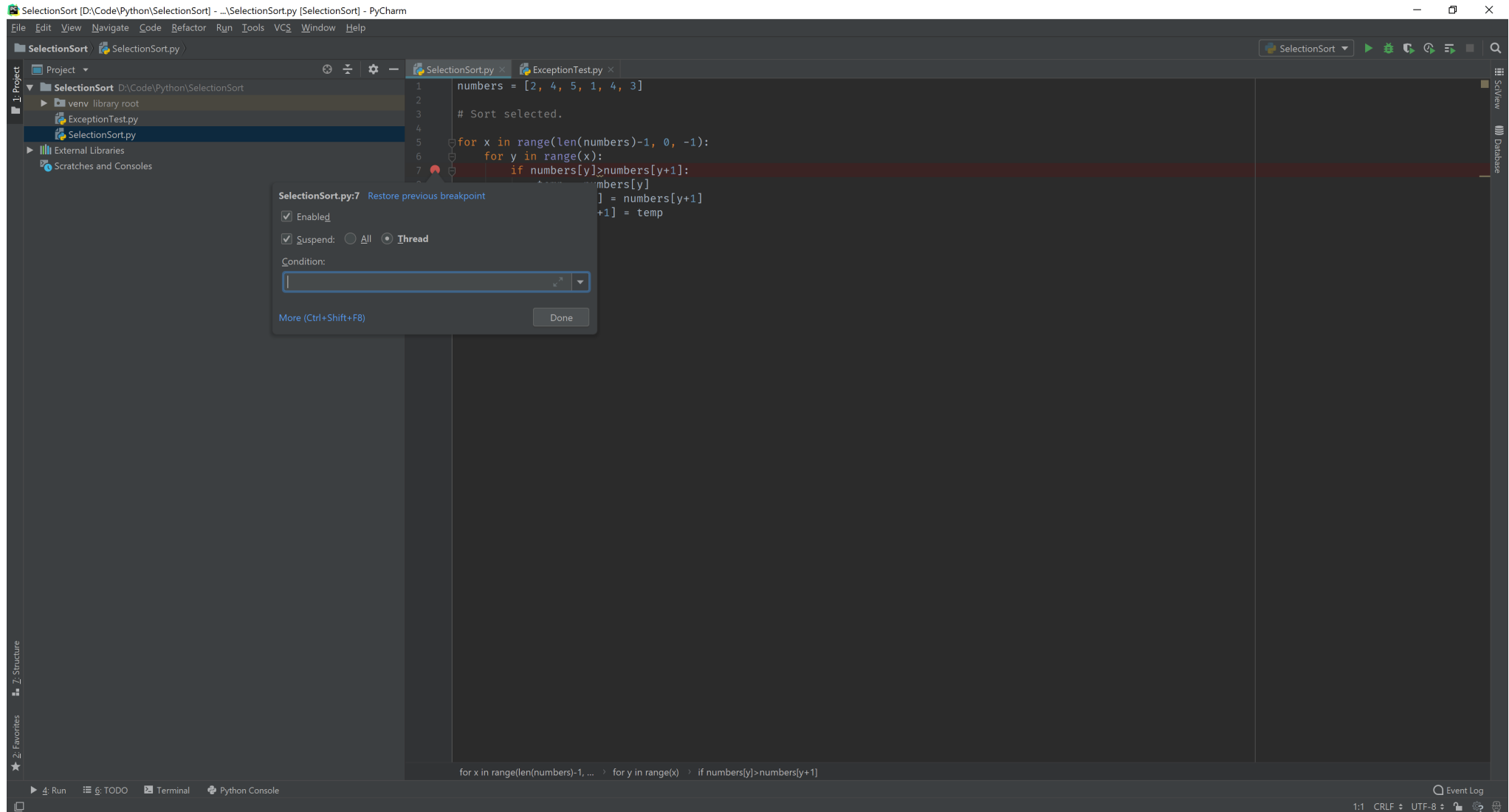
- CTRL-F8
 - Sets a breakpoint
- CTRL-SHIFT-F8
 - Sets condition – Or shows all breakpoints
 - See next slide for dialog box.

Conditional Breakpoints

- Pressing CTRL-SHIFT-F8 will give you a short dialog for setting the properties of a breakpoint.
- Pressing CTRL-SHIFT-F8 again, shows all the breakpoints.
- The icon for a conditional breakpoint is a red point with a question mark beside it.
- A condition might look like this:
 - `a=8`
 - `t!="blue"`



Conditional Breakpoints



+ - [📁]

- ▼ ☒ ● Python Line Breakpoint
 - ☒ ● test.py:3
 - ☒ ● test.py:4
 - ☒ ● test.py:2
- ▼ ☐ ⚡ JavaScript Exception Breakpoints
 - ☐ ⚡ Any exception
- ▼ ☒ ⚡ Python Exception Breakpoint
 - ☒ ⚡ Any exception
- ▼ ☐ ⚡ Django Exception Breakpoint
 - ☐ ⚡ Template render error
- ▼ ☐ ⚡ Jinja2 Exception Breakpoint
 - ☐ ⚡ Template render error
- ▼ ☐ ⚡ Jupyter Exception Breakpoint
 - ☐ ⚡ Any exception

test.py:2

- ☒ Enabled
- ☒ Suspend: ☐ All ☒ Thread

☐ Condition:

Log: ☐ "Breakpoint hit" message ☐ Stack trace

☐ Evaluate and log:

☐ Remove once hit

Disable until hitting the following breakpoint:

<None>

After hit: ☒ Disable again ☐ Leave enabled

```

1  print(1)
2  ● print("test")
3  ● t=4
4  ● print(t)
5

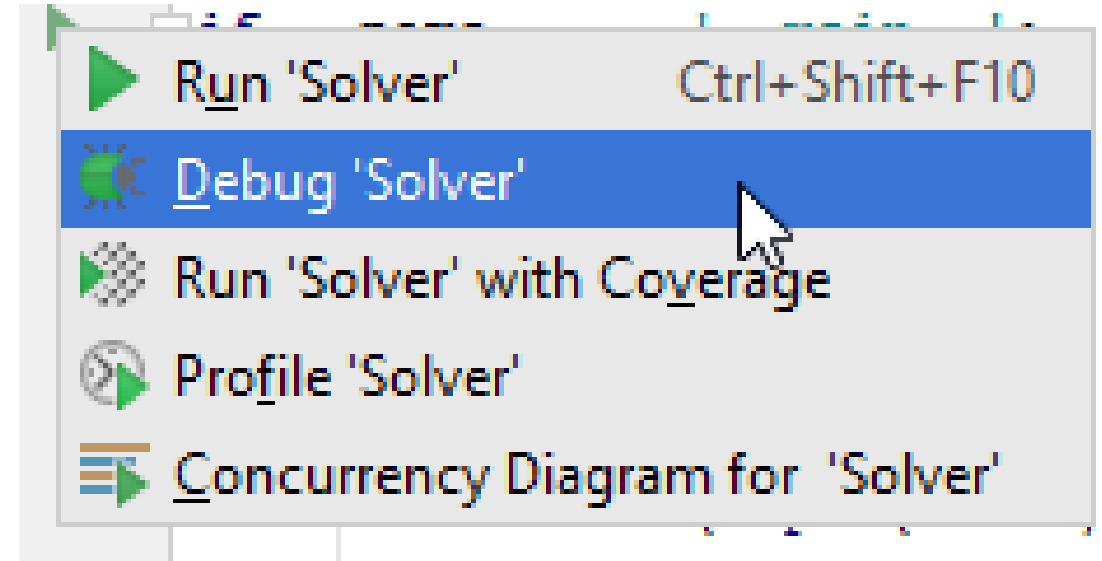
```

Other Options

- Log
 - Logs messages to console once logs are hit.
 - This can be used to analyse your output after a run.
- Suspend
 - You can choose NOT to suspend (only log).
 - This is useful when used with "Log" above.
 - The point becomes YELLOW when it's suspended.

Starting a Debugger Session

- Right-clicking on a specific file and selecting "Debug xxxxxx" will debug that specific file.







Common Debug Environment Icons and Commands

Basic Navigation Functions

- Rerun
 - Stop and debug again.
- Play
 - Once you are stopped (or stepping) you can resume by pressing play. Don't confuse this option with the "Run" command.
- Pause
 - Suspends program execution and enables debugging
- Stop
 - Stops debugging (but continues running the program to completion)



Item	Tooltip and Shortcut	Description
	Show Execution Point Alt+F10	Click this button to highlight the current execution point in the editor and show the corresponding stack frame in the Frames pane.
	Step Over F8	Click this button to execute the program until the next line in the current method or file, skipping the methods referenced at the current execution point (if any). If the current line is the last one in the method, execution steps to the line executed right after this method.
	Step Into F7	Click this button to have the debugger step into the method called at the current execution point.
	Step Into My Code Shift+Alt+F7	Click this button to skip stepping into library sources and keep focused on your own code.



Step Out

Shift+F8

Click this button to have the debugger step out of the current method, to the line executed right after it.



Drop frame

Interrupts execution and returns to the initial point of method execution. In the process, it drops the current method frames from the stack.



Run to Cursor

Alt+F9

Click this button to resume program execution and pause until the execution point reaches the line at the current cursor location in the editor. No breakpoint is required. Actually, there is a temporary breakpoint set for the current line at the caret, which is removed once program execution is paused. Thus, if the caret is positioned at the line which has already been executed, the program will be just resumed for further execution, because there is no way to roll back to previous breakpoints. This action is especially useful when you have stepped deep into the methods sequence and need to step out of several methods at once.

If there are breakpoints set for the lines that should be executed before bringing you to the specified line, the debugger will pause at the first encountered breakpoint.



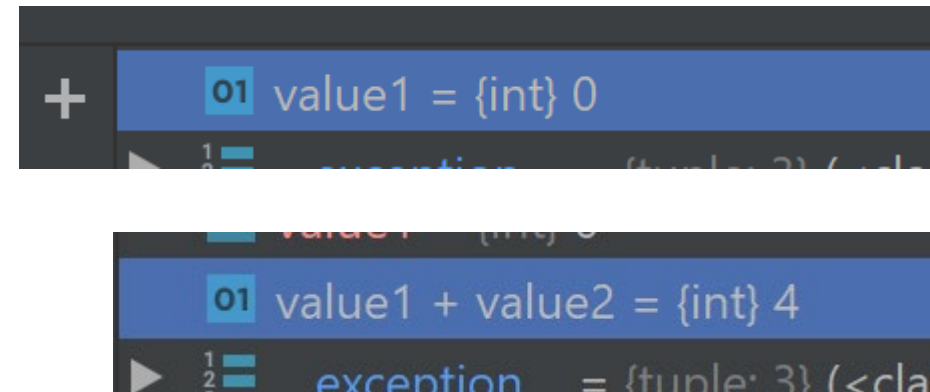
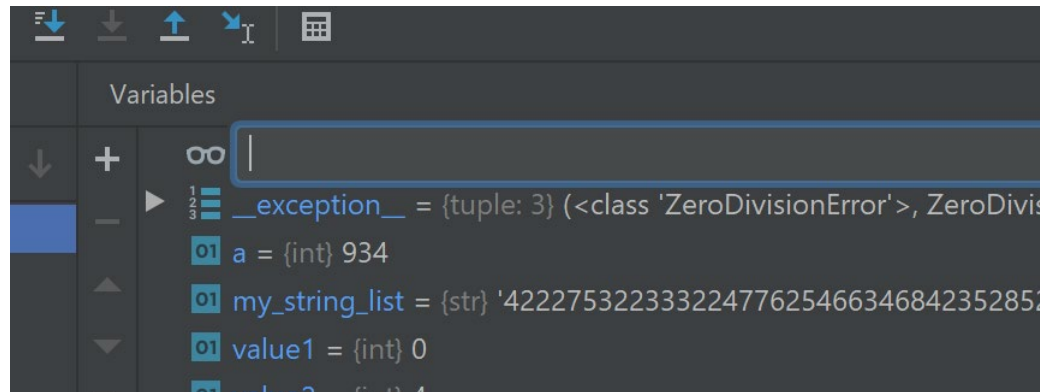
Evaluate
Expression

Alt+F8

Click this button to open the [Evaluate Expression](#) dialog.

Watches

- To add a new watch, start debugging, then press the (+) button to add a new watch.
- Type the name of a variable and press enter.
- OR, you can type a function like $a + b$



Debugging Exercise

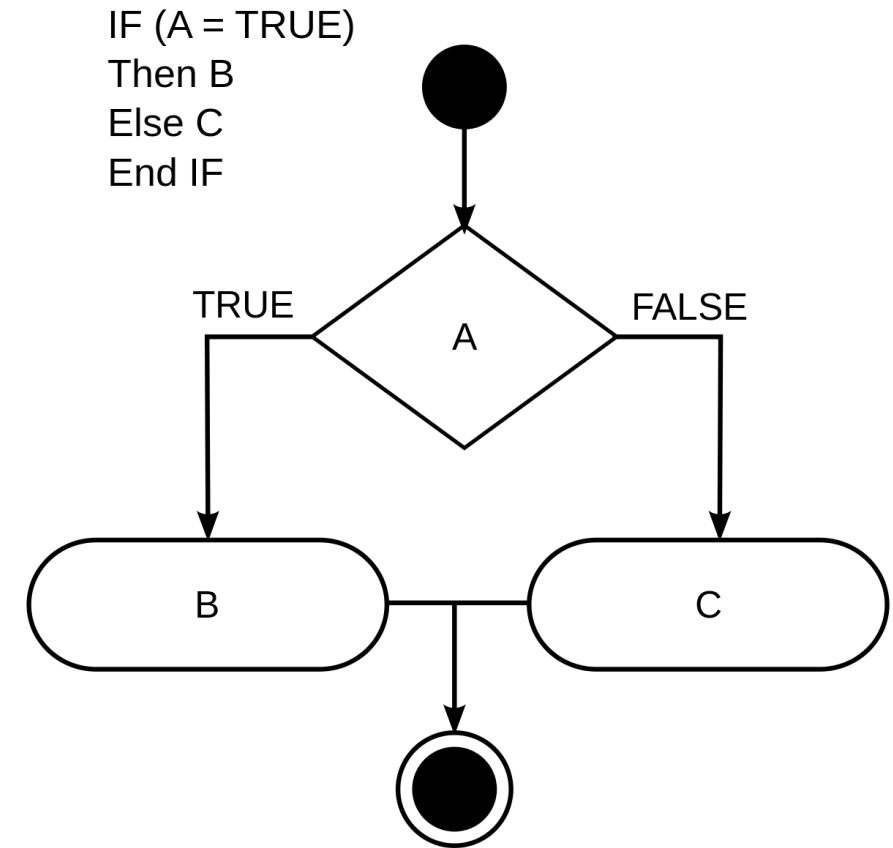
1. Copy the code from the sample python code.
2. Set a watch for value1, value2 and value3.
3. Place a debug in the code, on line 1, and step through the code to see how it works.
4. Observe the division by 0 error.
5. Set a watch on a, value1, value2 and value3.
6. Set a conditional break when one of the variables above is 0, make note of the position in the input string.
7. Make a copy of the input data lines, comment the original data
8. Delete the problem data.
9. Rerun

Conditions in Python

Python Control Flow

- To control blocks of code, we use a tab.
- This is a unique function of Python.

```
if val == 42:  
    print("The value was 42")  
else  
    print("The value was not 42")
```



Python Conditions

- Python supports the usual logical conditions from mathematics:
 - Equals: $a == b$
 - Not Equals: $a != b$
 - Less than: $a < b$
 - Less than or equal to: $a \leq b$
 - Greater than: $a > b$
 - Greater than or equal to: $a \geq b$

If statement

```
a = 33  
b = 200  
if b > a:  
    print("b is greater than a")
```

Indentation

- Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.
- If statement, without indentation (will raise an error):

```
a = 33
```

```
b = 200
```

```
if b > a:
```

```
print("b is greater than a") # you will get an error
```

Elif

- The elif keyword is python's way of saying "if the previous conditions were not true, then try this condition".

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

Else

- The else keyword catches anything which isn't caught by the preceding conditions.

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

Else

- The else keyword catches anything which isn't caught by the preceding conditions.

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

Compound Equality and Logic

- `age_0 >= 21 and age_1 >= 21`
- `age_0 >= 21 or age_1 >= 21`
- Use of AND and OR.
- Use of brackets is also permitted for enforcing order of operations or for code clarity.
 - `(age_0 >= 21) or (age_1 >= 21)`

And

- The and keyword is a logical operator, and is used to combine conditional statements:
 - Test if a is greater than b, AND if c is greater than a:

```
a = 200
```

```
b = 33
```

```
c = 500
```

```
if a > b and c > a:
```

```
    print("Both conditions are True")
```

Or

- The or keyword is a logical operator, and is used to combine conditional statements:
 - Test if a is greater than b, OR if a is greater than c:

```
a = 200
```

```
b = 33
```

```
c = 500
```

```
if a > b or a > c:
```

```
    print("At least one of the conditions is True")
```

Nested If

- You can have if statements inside if statements, this is called nested if statements

```
x = 41
```

```
if x > 10:  
    print("Above ten,")  
    if x > 20:  
        print("and also above 20!")  
    else:  
        print("but not above 20.")
```

Exercise - 4

- Restaurant Seating: Write a program that asks the user how many people are in their dinner group. If the answer is more than eight, print a message saying they'll have to wait for a table. Otherwise, report that their table is ready.

Exercise - 5

- Multiples of Ten: Ask the user for a number, and then report whether the number is a multiple of 10 or not.

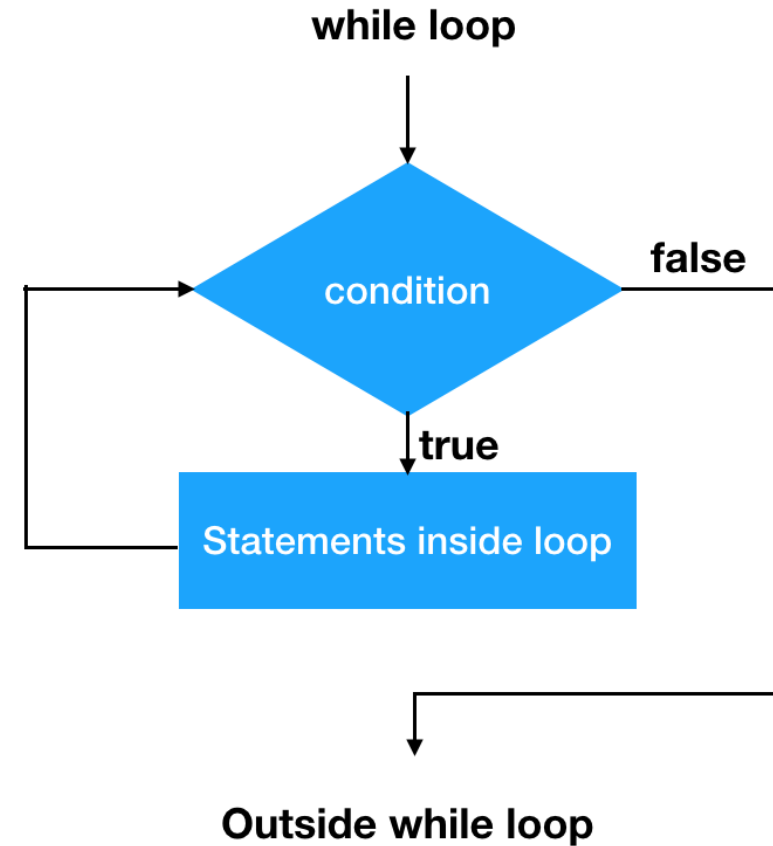


Looping Structures

While loop

```
a=1
while a < 10:
    print(a)
    a += 1
```

- A while loop tests the validity of the Boolean value before the statement(s) are executed.



The while Loop

- With the while loop we can execute a set of statements as long as a condition is true.
 - Print i as long as i is less than 6:

```
i = 1  
while i < 6:  
    print(i)  
    i += 1
```

- **Note:** remember to increment i, or else the loop will continue forever.

While loop asking user when to quit.

```
message = ""  
while message != 'quit':  
    message = input(prompt)  
    print(message)
```

The break Statement

- With the break statement we can stop the loop even if the while condition is true:
 - Exit the loop whe

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

The continue Statement

- With the continue statement we can stop the current iteration, and continue with the next:
 - Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

Exercise - 1

Multiples of Ten: Ask the user for a number, and then report whether the number is a multiple of 10 or not.

- Modify this assignment in order to do it in a loop. The loop should end once the user presses “q” or “Q”.

For Loops

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

for a in range(10):

- Prints the range 0-9.

for a in range(2,10):

- Prints the range 2-9

for a in range (0,a):

- Gets the range from 0 to (a-1).

Looping Through a String

- Strings contain a sequence of characters:
 - Loop through the letters in the word "banana":

```
for x in "banana":  
    print(x)
```

The range() Function

- To loop through a set of code a specified number of times, we can use the range() function,
- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
for x in range(6):  
    print(x)
```

- **Note** that range(6) is not the values of 0 to 6, but the values 0 to 5.

The range() Function

- The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

```
for x in range(2, 6):  
    print(x)
```


Exercise - 2

- Using for and continue or break, print a list of even numbers from 1 to 100.
- Reminder: To see if a number is even, if the modulus of 2 is 0 then it is even ($x \% 2 == 0$).

Exercise - 3

- Modify exercise 2 to prompt the user to provide a number, then determine that the number is even or odd.
- The program needs to ask the user if he want to continue or not, if yes, then the program asks for another number, if No, then the program ends.

Exercise - 4: Guessing Numbers

- Write a program that choose an integer between 0 and 100, inclusive. The program prompts the user to enter a number continuously until the number matches the chosen number. For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently.
- After the user guess correctly, the program will print: “Bravo you guess correctly after ... times”.

Homework Challenge: FizzBuzz

- Print a list of numbers from 0 to 25, including 25.
 - If the number is divisible by 3, print the number and "Fizz".
 - If the number is divisible by 5, print the number and "Buzz".
 - If the number is divisible by both, print the number and "FizzBuzz"
 - If the number is divisible by neither, print just the number.
 - Separate the number and "FizzBuzz" by a TAB character.
- Note, this is a commonly asked question at job interviews.
- The goal, rather than just making it work, is to try to write it as clearly and efficiently as possible.

FizzBuzz Output

00: FizzBuzz

01:

02:

03: Fizz

.

15: FizzBuzz

.

25: Buzz

Homework Challenge 2: Nested Loops

- Write a program which uses nested loops to create a multiplication table.
- Use tabs (escape them) to achieve placement.
- Use formatting to format numbers if necessary.

Multiplication Table									
	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81



CONCORDIA.CA

