Hacettepe University

Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

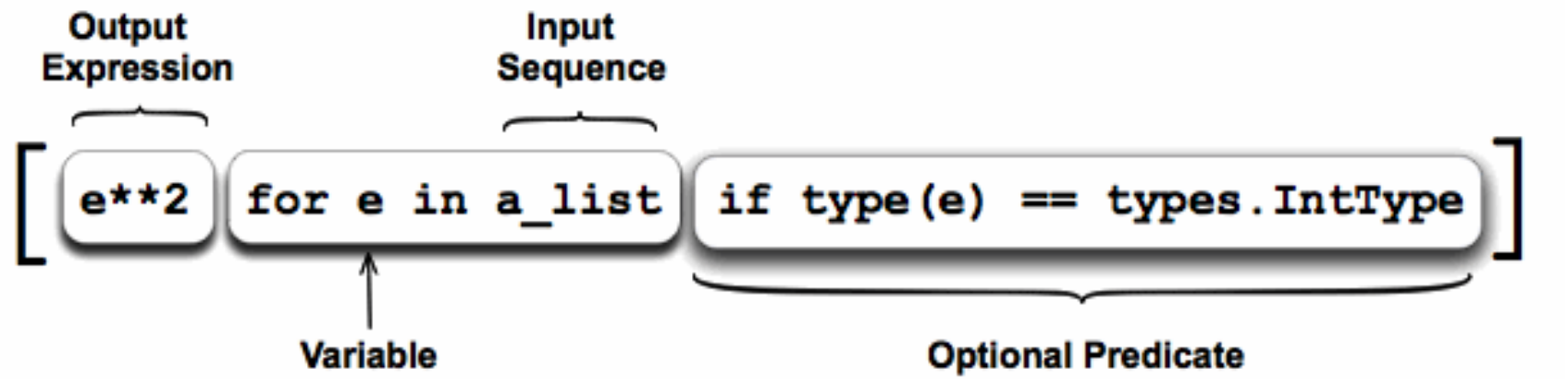
Week 10

Fall 2016

Python Comprehensions

- Python comprehensions are syntactic constructs that enable sequences to be built from other sequences in a clear and concise manner. Python comprehensions are of three types namely:
  - ❑ list comprehensions,
  - ❑ set comprehensions and
  - ❑ dict comprehensions.

Comprehensions



Example:

List Comprehensions

- List comprehensions provide a concise way to create a new list of elements that satisfies a given condition from an **iterable**. An **iterable** is any python construct that can be looped over.

Example: for loop

```
squares = []
for x in range(10):
    squares.append(x**2)
print(squares)
```

list comprehension

```
squares = [x**2 for x in range(10)]
print(squares)
```

Output:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Output:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Example:

```
even_squares = [i**2 for i in range(10) if i % 2 == 0]
print(even_squares)
```

Output:

```
[0, 4, 16, 36, 64]
```

Example:

```
S = [x**2 for x in range(10)]
V = [2**i for i in range(13)]
M = [x for x in S if x % 2 == 0]

print(S)
print(V)
print(M)
```

Output:

```
S: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
V: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
M: [0, 4, 16, 36, 64]
```

Nested *for* loops in List Comprehensions

- List comprehensions can also be used with multiple or nested *for* loops.

Example: nested for loops

```
combs = []
for x in [1,2,3]:
    for y in [3,1,4]:
        if x != y:
            combs.append((x, y))

print(combs)
```

list comprehension

```
combs1=[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
print(combs1)
```

Output:

```
[(1, 3), (1, 4), (2, 3), (2, 1),
(2, 4), (3, 1), (3, 4)]
```

Output:

```
[(1, 3), (1, 4), (2, 3), (2, 1),
(2, 4), (3, 1), (3, 4)]
```

Set Comprehensions

- In set comprehensions, we use the braces rather than square brackets.

Example:

```
x = {i**2 for i in range(10)}
print(type(x))
print(x)
```

Output:

```
<class 'set'>
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
```

Dict Comprehensions

Example:

```
x = {i:i**2 for i in range(10)}
print(type(x))
print(x)
```

Output:

```
<class 'dict'>
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

Example:

```
noprimes = [j for i in range(2, 8) for j in range(i*2, 50, i)]
primes = [x for x in range(2, 50) if x not in noprimes]

print (noprimes)
print (primes)
```

Output:

```
Noprimes: [4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30,
 32,34, 36,38, 40, 42, 44, 46, 48, 6, 9, 12, 15, 18, 21, 24,27,
 30,33,36, 39, 42, 45, 48, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44,
 48, 10,15,20, 25, 30, 35, 40, 45, 12, 18, 24, 30, 36, 42, 48,14,
 21,28,35, 42, 49]
Primes[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

Example:

```
words = 'The quick brown fox jumps over the lazy dog'.split()
print (words)
stuff = [[w.upper(), w.lower(), len(w)] for w in words]
for i in stuff:
    print(i)
```

Output:

```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
['THE', 'the', 3]
['QUICK', 'quick', 5]
['BROWN', 'brown', 5]
['FOX', 'fox', 3]
['JUMPS', 'jumps', 5]
['OVER', 'over', 4]
['THE', 'the', 3]
['LAZY', 'lazy', 4]
['DOG', 'dog', 3]
```

Example:

```
def zip(lst1, lst2):
    """
    Made an assumption both lst1 and lst2 will have the same length.
    Used the range function to get the position the item so that we can use the position
    as the index key for both list.
    """
    return [(lst1[i], lst2[i]) for i in range(len(lst1))]

print(zip([1, 2, 3], ["a", "b", "c"]))
```

Output:

```
[(1, 'a'), (2, 'b'), (3, 'c')]
```

Example:

```
non_flat = [ [1,2,3], [4,5,6], [7,8] ]
list=[y for x in non_flat for y in x]
print(list)
```

Output:

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

Example:

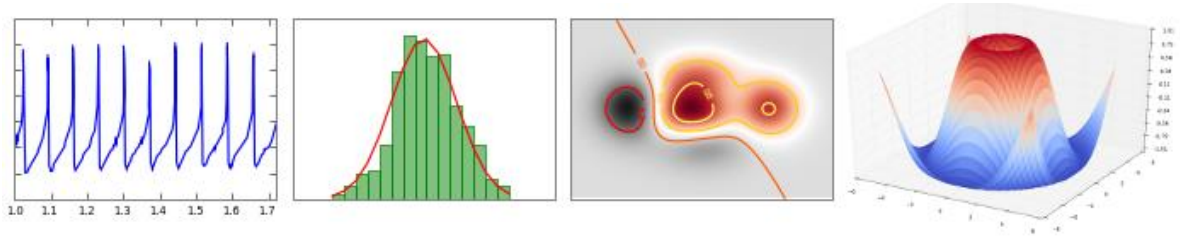
```
def map(func, lst):
    """
    This was pretty simple following the basic formula.
    Since we can pass functions around as an argument, the map function
    receives the the function to be applied.  The function is then applied to
    each item in the list.
    """
    return [func(i) for i in lst]

def square(x):
    return x * x

assert map(square, range(5)) == [0,1,4,9,16]
```

2D Data Plotting in Python: matplotlib

- **matplotlib** is a Python 2D plotting library
- You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc.



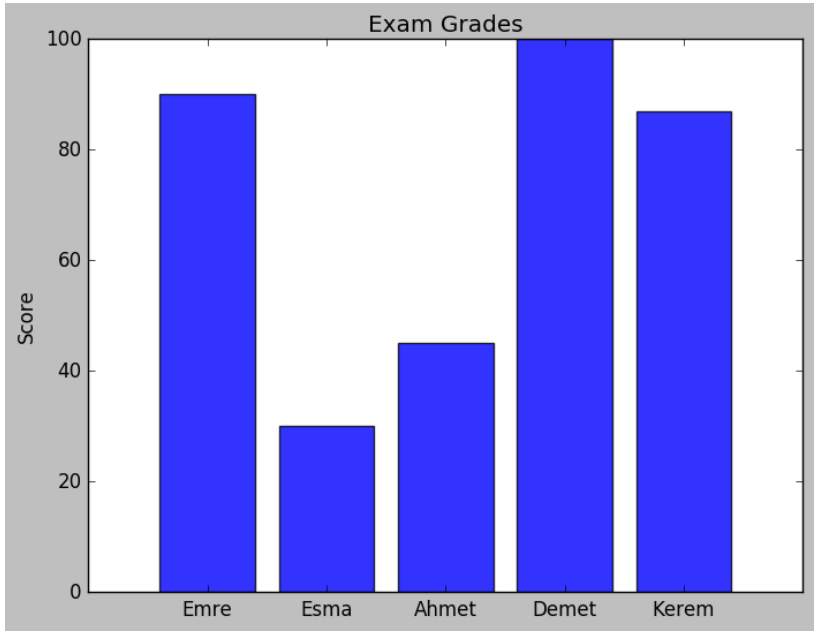
- Installing matplotlib: http://matplotlib.org/users/installing.html
- matplotlib in PyCharm: https://www.jetbrains.com/help/pycharm/2016.1/matplotlib-support.html
- Or use Anaconda that provides numerous built-in Python packages including matplotlib: https://www.continuum.io/downloads

Vertical Bar Chart Plotting

- **Example:**

```
1 import matplotlib.pyplot as plot
2
3 students = ['Emre', 'Esma', 'Ahmet', 'Demet', 'Kerem']
4 grades = [90,30,45,100,87]
5 x_pos = [x for x in range(len(students))]
6
7 plot.bar(x_pos, grades, align='center', color='b', alpha=0.8)
8 plot.xticks(x_pos, students)
9 plot.ylabel('Score')
10 plot.title('Exam Grades')
11
12 plot.show()
```

- **Output:**

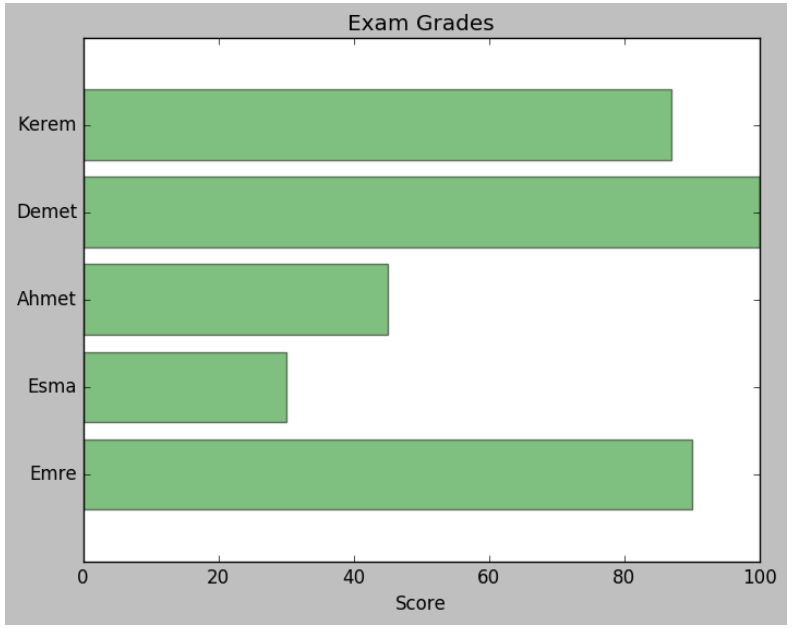


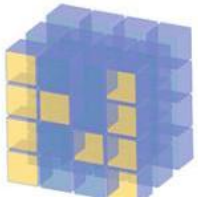
Horizontal Bar Chart Plotting

- **Example:**

```
1  import matplotlib.pyplot as plot
2
3  students = ['Emre', 'Esma', 'Ahmet', 'Demet', 'Kerem']
4  grades = [90,30,45,100,87]
5  y_pos = [x for x in range(len(students))]
6
7  plot.barh(y_pos, grades, align='center', color='g', alpha=0.5)
8  plot.yticks(y_pos, students)
9  plot.xlabel('Score')
10 plot.title('Exam Grades')
11
12 plot.show()
```

- **Output:**





NumPy - scientific computing with Python

- **NumPy** (http://www.numpy.org) is the fundamental package for scientific computing with Python. It supports among other things:
  - a powerful N-dimensional array object,
  - sophisticated (broadcasting) functions,
  - useful linear algebra, Fourier transform, and random number capabilities,
  - efficient multi-dimensional container of generic data,
  - arbitrary data-types.

- Installing Packages in PyCharm (search for numpy): https://www.jetbrains.com/help/pycharm/2016.1/installing-uninstalling-and-upgrading-packages.html

- Or use Anaconda that provides numerous built-in Python packages including NumPy: https://www.continuum.io/downloads

A simple plot with a custom dashed line

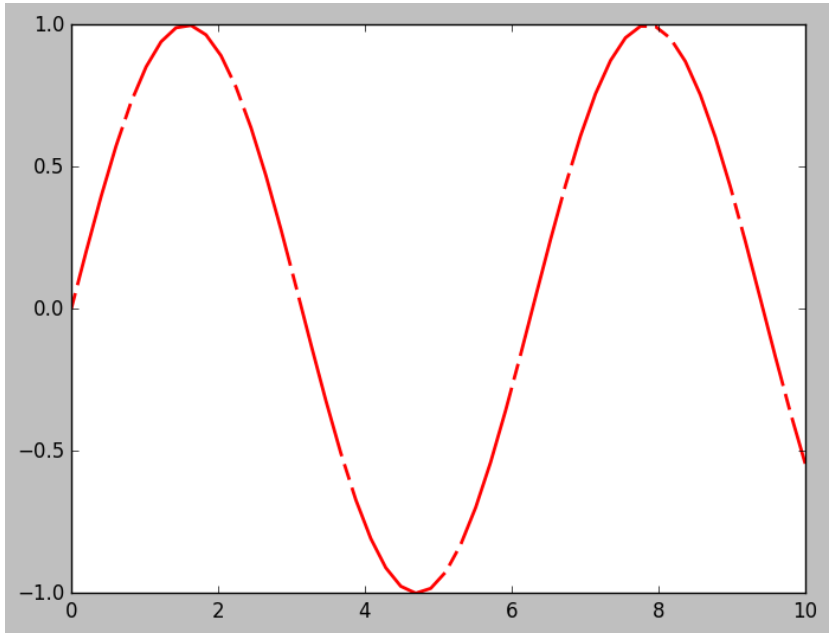
- **Example:**

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.linspace(0, 10)
5
6 line, = plt.plot(x, np.sin(x), '--', linewidth=2, color="r")
7
8 dashes = [10, 5, 100, 5]  # 10 points on, 5 off, 100 on, 5 off
9 line.set_dashes(dashes)
10
11 plt.show()
```

New function: `numpy.linspace(start, stop)`

Returns evenly spaced numbers over a specified interval [*start*, *stop*].

- **Output:**



A simple plot of fill function

- **Example:**

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x = np.linspace(0, 1)
5 y = np.sin(4 * np.pi * x) * np.exp(-5 * x)
6
7 plt.fill(x, y, 'y')
8 plt.grid(True)
9 plt.show()
```

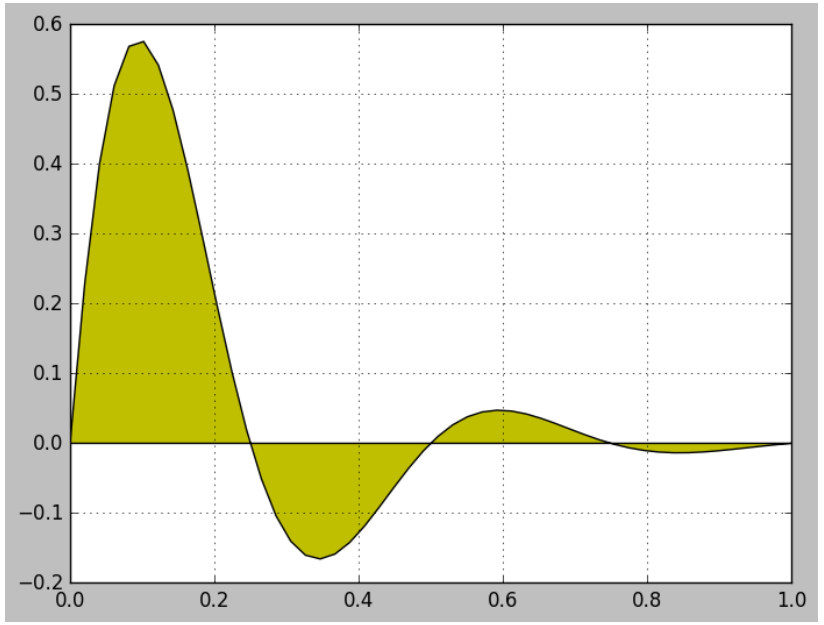
New functions:

`numpy.sin()` – Trigonometric sine, element-wise

`numpy.exp()` – Calculate the exponential of all elements in the input array

`numpy.pi()` – π mathematical constant

- **Output:**



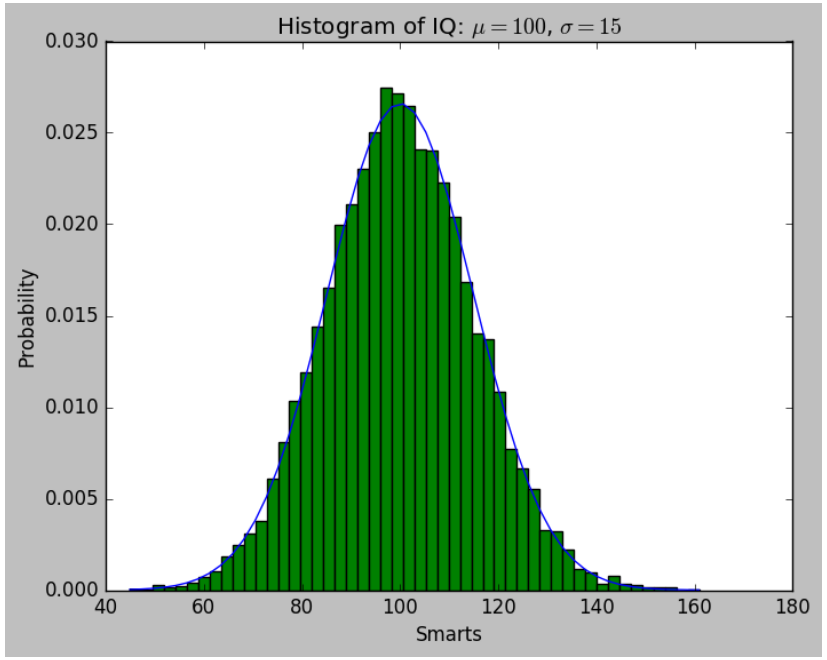
Histogram Plotting

A *histogram* is a graphical representation of the distribution of numerical data.

- **Example:**

```
1 import numpy as np
2 import matplotlib.mlab as mlab
3 import matplotlib.pyplot as plt
4
5 mu = 100  # mean of distribution
6 sigma = 15  # standard deviation of distribution
7 x = mu + sigma * np.random.randn(10000)
8
9 num_bins = 50
10 # the histogram of the data
11 n, bins, patches = plt.hist(x, num_bins, normed=1, facecolor='green')
12 # add a 'best fit' line
13 y = mlab.normpdf(bins, mu, sigma)
14 plt.plot(bins, y, 'b-')
15 plt.xlabel('Smarts')
16 plt.ylabel('Probability')
17 plt.title(r'Histogram of IQ: $\mu=100$, $\sigma=15$')
18
19 # Tweak spacing to prevent clipping of ylabel
20 plt.subplots_adjust(left=0.15)
21 plt.show()
```

- **Output:**



New function:
`numpy.random.randn(dimension)`

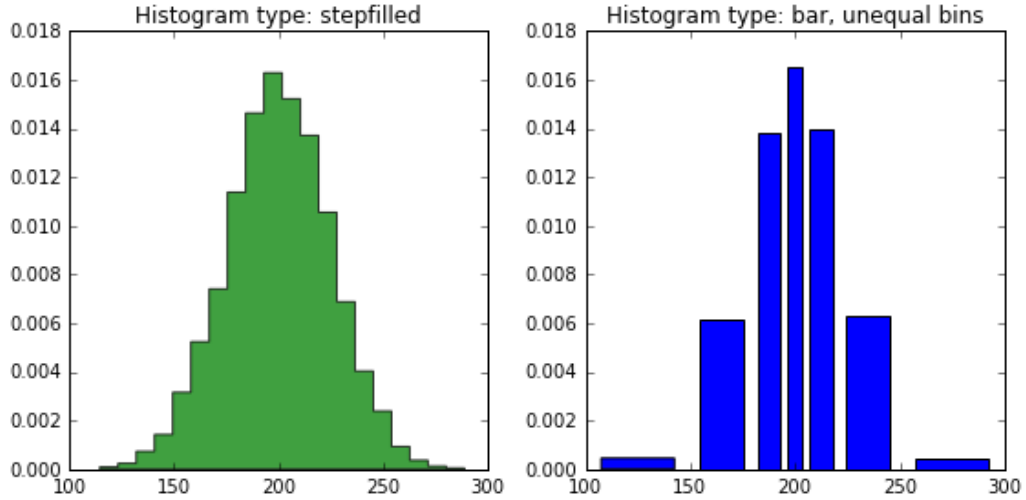
Returns a sample (or samples) from the "standard normal" distribution

Histogram Plotting Continued (Subplots)

- **Example:**

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 mu = 200
5 sigma = 25
6 x = mu + sigma*np.random.randn(10000)
7 print(x)
8 fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(8, 4))
9
10 ax0.hist(x, 20, normed=1, histtype='stepfilled', facecolor='g', alpha=0.75)
11 ax0.set_title('Histogram type: stepfilled')
12
13 #Create a histogram by providing the bin edges (unequally spaced).
14 bins = [100, 150, 180, 195, 205, 220, 250, 300]
15 ax1.hist(x, bins, normed=1, histtype='bar', rwidth=0.7)
16 ax1.set_title('Histogram type: bar, unequal bins')
17
18 plt.tight_layout()
19 plt.show()
```

- **Output:**



2D Plotting and Scientific Computing in Python

- **For more matplotlib examples:** http://matplotlib.org/examples/index.html
- **Plotting Commands Summary:** http://matplotlib.org/api/pyplot_summary.html
- **NumPy Manual:** https://docs.scipy.org/doc/numpy/index.html

Debugging

- ***Debugging*** is the process of identifying and removing errors that prevent correct operation of computer software or a system.

- PyCharm provides a full range of facilities for debugging your source code:
  - Breakpoints in Python.
  - Customizable breakpoint properties: conditions, pass count, etc.
  - Frames, variables, and watches views in the debugger UI.
  - Runtime evaluation of expressions.

- For detailed explanation of the debugging process in PyCharm: https://www.jetbrains.com/help/pycharm/2016.1/debugging.html

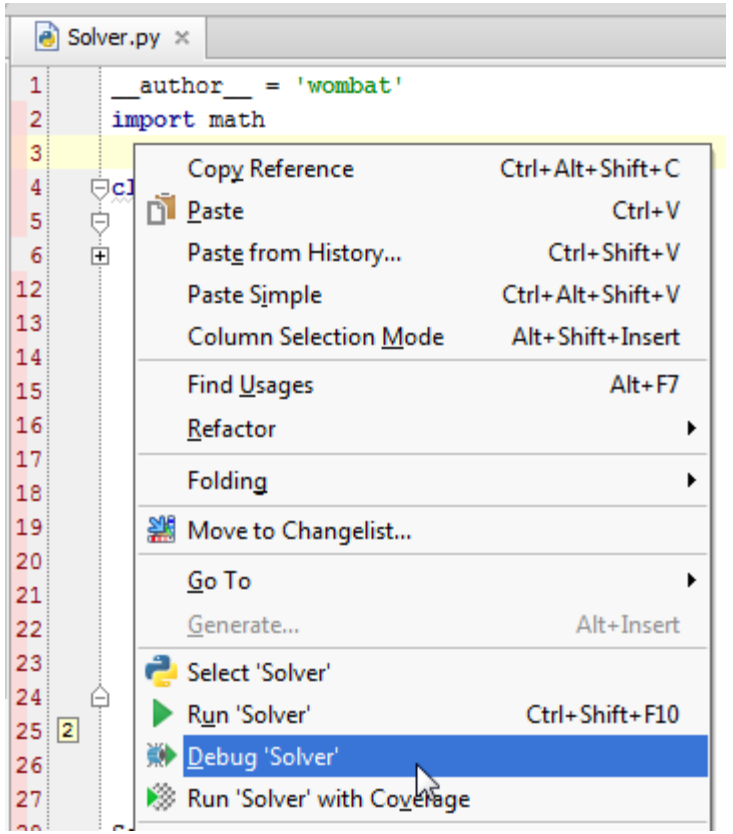
Debugging Cont.

- **General debugging steps:**

1. Configure the debugger options.
2. Define a run/debug configuration.
3. Create breakpoints in the source code.
4. Launch a debugging session.
5. Pause or resume the debugging session as required.
6. During the debugger session, step through the breakpoints, evaluate expressions, change values on-the-fly , examine suspended program, explore frames, and set watches .

Starting the Debugger Session

- Set breakpoints in the source code.
- Open the desired Python script in the editor, or select it in the Project tool window.
- On the context menu, choose Debug <script name>:



PyCharm Debug Tool Window

View | Tool Windows | Debug

Alt+5

- The Debug tool window becomes available when you start debugging.
- It displays the output generated by the debugging session for your application.
- For Toolbars and Items descriptions: https://www.jetbrains.com/help/pycharm/2016.1/debug-tool-window.html