Python Playground

The task: White noise

Suppose we want to generate a white noise process $\epsilon_0, \epsilon_1, \epsilon_2, \ldots \epsilon_T$ where each ϵ_i is drawn from an independent standard normal distribution.

How do we do this in Python?

Version 1: Fast and concise

```
import numpy as np
import matplotlib.pyplot as plt
x = np.random.randn(100)
plt.plot(x)
plt.show()
```

- `import` statements are necessary to get
  - a plotting (`matplotlib`) and a numerical computation (`numpy`) **packages**.
    - packages are folder with code files, and a `__init__.py` file.
    - Python's core functionality is small by design, so it needs to be enriched with packages.
- `import numpy as np` creates an alias for `numpy`
- `random` is a **subpackage** of `numpy` , and `randn` is a function inside this subpackage.

A geeky joke

```
import antigravity

def main():
    antigravity.fly()

if __name__ == '__main__':
    main()
```

Version 2: `for` loops

```
import numpy as np
import matplotlib.pyplot as plt
ts_length = 100
epsilon_values = []
# Empty list
for i in range(ts_length):
    e = np.random.randn()
    epsilon_values.append(e)
plt.plot(epsilon_values, 'b-')
plt.show()
```

- Note the indentation on the `for` loop. In Python, **all code blocks** (things that happen inside loops, function definitions, etc) are delimited by indentation => Great source of errors for beginners!

Version 3: `while`

```
import numpy as np
import matplotlib.pyplot as plt
ts_length = 100
epsilon_values = []
i = 0
while i < ts_length:
    e = np.random.randn()
    epsilon_values.append(e)
    i = i + 1 # same as i+=1
plt.plot(epsilon_values, 'b-')
plt.show()
```

Version 4: User defined function

```
import numpy as np
import matplotlib.pyplot as plt
def generate_data(n):
    epsilon_values = []
    for i in range(n):
        e = np.random.randn()
        epsilon_values.append(e)
    return epsilon_values

data = generate_data(100)
plt.plot(data, 'b-')
plt.show()
```

Version 5: Random uniform vs random normal

```
import numpy as np
import matplotlib.pyplot as plt
def generate_data(n, generator_type):
    epsilon_values = []
    for i in range(n):
        if generator_type == 'U':
            e = np.random.uniform(0, 1)
        else:
            e = np.random.randn()
        epsilon_values.append(e)
    return epsilon_values

data = generate_data(100, 'U')
plt.plot(data, 'b-')
plt.show()
```

Version 6: Pass a function as argument

```
import numpy as np
import matplotlib.pyplot as plt
def generate_data(n, generator_type):
    epsilon_values = []
    for i in range(n):
        e = generator_type()
        epsilon_values.append(e)
    return epsilon_values
data = generate_data(100, np.random.uniform)
plt.plot(data, 'b-')
plt.show()
```

List comprehensions

We can write the same function as above with less code using **list comprehensions**.

```
animals = ['cat','dog','bird']
plurals = [animal+'s' for animal in animals]
plurals
```

and it works for numbers:

```
doubles = [2*x for x in range(8)]
doubles
```

Version 7: V6 + list comprehension

```
import numpy as np
import matplotlib.pyplot as plt
def generate_data(n, generator_type):
    epsilon_values = [generator_type() for _ in range(n)]
    return epsilon_values
data = generate_data(100, np.random.uniform)
plt.plot(data, 'b-')
plt.show()
```

More about Python functions

- A few functions included by default.

```
max(19,20)
min(19,20)
sum([19,20])
str(22)
type(22)
len([19,20,21])
18 % 2
19 % 2
```

Defining functions

```
def square(x):
    """
    This function returns the square of a number
    """
    return x**2

square?
square??
square(2)
```

Anonymous functions

```
square = lambda x: x**2
square(2)
```

They are useful when we don't really need to write the function separately, for instance, to calculate $\int_0^1 x^2 dx$.

```
from scipy.integrate import quad
quad(lambda x: x**2, 0,1)
```

Exercises

1. Write a function that computes the inner product of two vectors, x, y of the same size.
2. For a given natural number n, write a function that computes $n!$.
3. Given as input a value x and a list of coefficients $[a_0, a_1, \ldots a_n]$, write a function that returns the value of $a_0 + a_1 + \ldots + a_n x^n$.
4. Simulate and plot the correlated time series $x_{t+1} = \alpha x_t + \epsilon_{t+1}$, for $T = 200$ and $\alpha = 0.9$.
5. Compute an approximation to π using Monte Carlo. Only import allowed: `import numpy as np`.