Introduction to Python

Ex Sheet 1

Introduction to Python - Practical Exercises 1

Create a jupyter notebook for each of the different exercises. Call your notebook `ex1_1` for the first exercise, `ex1_2` for the second and so on. Use this naming convention for each of your practical exercises that follow.

a) Write a one-line program that prints your name.

b) Add a one-line comment to the program describing what it does.

c) Write a program that stores your name in a variable called `my_name`, then prints it out.

d) Write a program that stores an integer in a variable called `my_integer`, then prints out "This number is: " followed by the contents of the variable.

e) Write a program that allows the user to type in their name, then prints it out.

f) Write a program that takes an integer as input from a user and stores the value in a variable, then prints out that number multiplied by 2.

g) Write a program that takes two integers as inputs, storing them in different variables, then prints out the value of the numbers multiplied together.

h) Write a program that takes an integer as an input, then checks if that number is 10 or more. If so, print out that the number is a multiple-digit number.

i) Write a program that takes an integer as input, then checks if the number is 10 or more. If so, print out that the number is multiple-digit, otherwise, print out that it is single-digit and on a separate line, print the number.

j) Write a program that takes an integer as input, then uses a loop to subtract 10 from the integer until it is less than 10. Then print the number.

Reference for Practical Exercises 1

Functions

`print(string)`: Prints the contents of `string`.

`input(string)`: Prints `string` and then waits for input from the user, ending with the enter key. It returns the value of the input.

`int(x)`: Casts x as an integer.

`str(x)`: Casts x as an string.

`float(x)`: Casts x as a float number.

Operators

`x + y`: If x and y are both numbers, add them together. If x and y are both strings, concatenate them.

`x - y`: If x and y are both numbers, subtract y from x.

`x * y`: If x and y are both numbers, multiply them together. If x is a string and y is an integer, repeat x a total of y times. `x * y`: If x and y are both numbers, multiply them together. If x is a string and y is an integer, repeat x a total of y times.

`x / y`: If x and y are both numbers, divide x by y, returning a float.

`x ** y`: If x and y are both numbers, return x to the power of y. Returns an integer for integer results, a float if necessary.

`x = y`: Assigns the value (and data type) of y to the variable x.

`x == y`: If x and y have the same value, return True, otherwise return False.

`x != y`: If x and y have the same value, return False, otherwise return True.

`x > y`: If x is greater than y, return True, otherwise return False.

`x >= y`: If x is greater than or equal to y, return True, otherwise return False.

`x < y`: If x is less than y, return True, otherwise return False.

`x <= y`: If x is less than or equal to y, return True, otherwise return False.

Statements

```
if condition:
    statement_block_1
else:
    statement_block_2
```

If `condition` is True, then perform all statements in `statement_block_1`, otherwise perform all statements in `statement_block_2`. The `else` is completely optional and need not be included.

```
while condition:
    statement_block
```

If `condition` is True, perform all statements in `statement_block`, then repeat.

Introduction to Python - Practical Exercises 2

a) Create a list with five elements in it of things you might buy for groceries. Print out the third element.

b) Create a variable holding your full name, then print the 5th character.

c) Using a list of five grocery items, print out each item on its own line using a for loop.

d) Use a for loop and the range function to print every number from 0 to 10, including 10, squared.

e) Import the datetime module giving it an alias of dt. Use the time function to assign a variable the time of 5:25pm. Print the variable.

f) Import numpy with the alias np. Create a 2x2 array matching the matrix $\left[\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right]$. Print the array.

g) Create a 2x3x2 array in a variable containing ones in every element. Print the variable.

h) Create a 3x3 array matching the matrix $\left[\begin{smallmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{smallmatrix}\right]$. Print the element from the intersection of the 2nd row and 3rd column.

i) Using the array from the previous question, print out its shape, then print every element in it by iterating along its rows and columns with a for loop.

j) Using the same array, square every number in the array, then print out the maximum value contained in it.

Reference for Practical Exercises 2

Functions

`numpy.amin(`*`array`*`)`: Prints the lowest value in `array`.
`numpy.amax(`*`array`*`)`: Prints the highest value in `array`.
`numpy.ones((`*`shape`*`))`: `shape` should be in the form of integers separated by commas. Creates an array of shape defined by you, with every element a 1.
`numpy.zeros((`*`shape`*`))`: As above, but every element is a 0.
`numpy.array(`$[x_1, x_2, x_3, \ldots, x_n]$`)`: Creates an 1-dimensional array holding values $x_1, x_2, x_3, \ldots, x_n$.
`numpy.array(`$[[x_{11}, x_{12}, \ldots, x_{1n}], [x_{21}, x_{22}, \ldots, x_{2n}], \ldots, [x_{m1}, x_{m2}, \ldots, x_{mn}]]$`)`: Creates an 2-dimensional array of size $m \times n$, holding the values above. It is possible to create 3-dimensional or higher dimensional arrays by enclosing them all in outer square brackets and separating each lower-dimensional entry by commas.
`range(`*`n`*`)`: Returns a list of integers from 0 to `n - 1`.
`datetime.time(`*`hour, minute, second`*`)`: Returns a time with hour, minute, second as specified.

Attributes and Methods of the Array Object

`numpy_array``.shape`: Returns a list of integers containing the size of `numpy_array` in every dimension.
`numpy_array``[`*`n`*`]`: Returns the nth member of the 1-dimensional `numpy_array`. You can return elements from higher dimensional arrays by adding integers separated by commas.

Statements

```
for item in list :
    statement_block
```

Assign the first value in `list` to the variable `item`. Perform all the statements in `statement_block`. Then move to the next value in `list` and repeat until there are no more.
`import `*`module`*` as `*`alias`*: Imports a module into a program and gives it an alias for ease of reference.

Advanced Python - Practical Exercises 3

We are going to simulate some highly simplified movement behaviour and plot it with Python. The goal is to use `matplotlib`, a Python library for plotting diagrams, to show 100 individuals moving in a square area.

Each individual should move in a fixed heading chosen at random for that individual at the start, e.g. all individuals start with an initial random heading between 0 and 2π.

At every timestep each individual's position is updated by moving them at a fixed speed in the direction of their heading. Use a set speed which is equal for all individuals and low enough that movement can clearly be observed.

You will need to create 3 `numpy` arrays, two for the xy coordinates and one for the heading.

The area they live in should be $1x1$ square and if an individual crosses a boundary of the area, they should wrap around to the opposite side and continue moving. The simulation should be able to demonstrate how the individuals move over a period of time, at least over 100 steps of motion.

The end result should be an animation that looks like this:

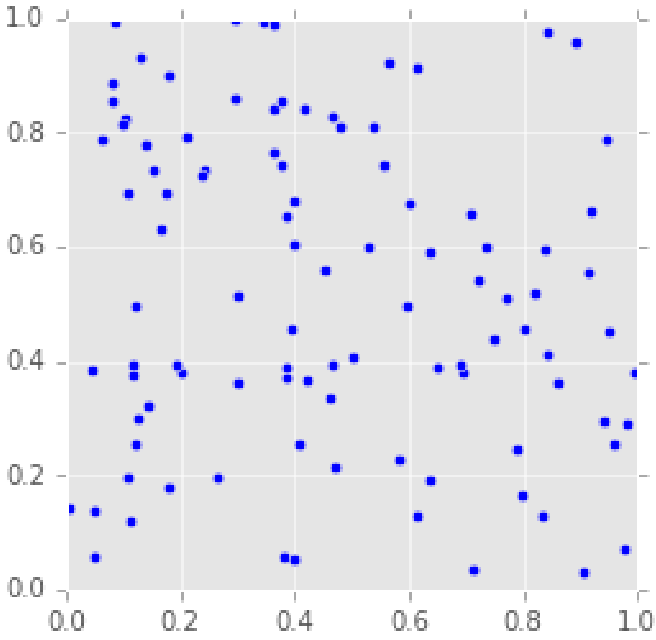

Figure 1: Locations of simulated animals moving around a square domain

Reference for Practical Exercises 3

`import matplotlib.pyplot as plt`: This imports the `matplotlib.pyplot` module and gives it an alias of `plt`.

To animate our simulated moving animals we’re going to use some tricks. The following code is an example of plotting a figure within a loop so that is updated constantly. Before attempting the movement task run this piece of code so that you can copy the lines you need for your task.

```
import matplotlib.pyplot as plt
import numpy as np
from IPython import display

%matplotlib inline
plt.style.use(’ggplot’)

x = np.arange(0,2*np.pi,0.02*np.pi)

for k in range(100):
    plt.clf()
    y = np.sin(x+k/10.0)
    plt.plot(x,y)
    plt.axis([0,6.3,-1.1,1.1])
    display.display(plt.gcf())
    display.clear_output(wait=True)
```

`plt.scatter(`*x,y*`)`: This builds a scatterplot of points located at the co-ordinates held in x and y. Note that x and y should both be numpy arrays of floats or integers.
`np.random.uniform(`*low,high,size*`)`: returns a numpy array of length *size* consisting of random selections from the uniform distribution which starts at *low* and ends at *high*.
`np.cos(`*heading*`)`: returns a numpy array of the *cosine* of the heading array.
`from math import pi`: once this is executed `pi` returns the value of π.
`plt.clf()`: this clears any current plot being built.

As a challenge, we are going to try to implement Conway's Game of Life. This is a "game" of simple rules played across a board consisting of square cells that can be either live or dead. To play the game, you set up an initial state (a particular pattern of live cells) and observe the subsequent behaviour as generations pass.

Each cell has eight neighbours, the eight cells adjacent to them (either vertically, horizontally or diagonally).

The game's rules are very simple:

- Any live cell with fewer than two live neighbours dies on the next generation.
- Any live cell with two or three live neighbours lives on to the next generation.
- Any live cell with more than three live neighbours dies on the next generation.
- Any dead cell with exactly three live neighbours becomes live on the next generation.

From one generation to the next, these rules must all be applied simultaneously (in other words, don't apply the rules a cell at a time, but instead to all cells at once using only the information from the preceding generation).

Create and implement the game of life on a 200x200 set of cells.

Try testing your implementation by beginning with live cells in the formation shown below. They should "glide" across the map once you begin the generations, changing shape slightly as they do.