```
{
  "nbformat": 4,
  "nbformat_minor": 0,
  "metadata": {
    "colab": {
      "name": "Python_tutorial.ipynb",
      "provenance": [],
      "collapsed_sections": [],
      "toc_visible": true
    },
    "kernelspec": {
      "display_name": "Python 3",
      "language": "python",
      "name": "python3"
    },
    "language_info": {
      "codemirror_mode": {
        "name": "ipython",
        "version": 3
      },
      "file_extension": ".py",
      "mimetype": "text/x-python",
      "name": "python",
      "nbconvert_exporter": "python",
      "pygments_lexer": "ipython3",
      "version": "3.7.6"
    }
  },
  "cells": [
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "dzNng6vCL9eP"
      },
      "source": [
        "# Python Tutorial With Google Colab"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "0vJLt3JRL9eR"
      },
      "source": [
        "This tutorial was adapted for Colab by Kevin Zakka for the Spring 2020 edition of [cs231n](https://cs231n.github.io/). It runs Python3 by default."
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "qVrTo-LhL9eS"
      },
      "source": [
        "##Introduction"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "9t1gKp9PL9eV"
      },
      "source": [
        "Python is a great general-purpose programming language on its own, but with the help of a few popular libraries (numpy, scipy, matplotlib) it becomes a powerful environment for scientific computing.\n",
        "\n",
        "We expect that many of you will have some experience with Python and numpy; for the rest of you, this section will serve as a quick crash course both on the Python programming language and on the use of Python for scientific computing.\n",
        "\n",
        "Some of you may have previous knowledge in Matlab, in which case we also recommend the numpy for Matlab users page (https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html)."
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "UlPvreR9L9eW"
      },
      "source": [
        "In this tutorial, we will cover:\n",
        "\n",
        "* Basic Python: Basic data types (Containers, Lists, Dictionaries, Sets, Tuples), Functions, Classes\n",
        "* Numpy: Arrays, Array indexing, Datatypes, Array math, Broadcasting\n",
        "* Matplotlib: Plotting, Subplots, Images\n",
        "* IPython: Creating notebooks, Typical workflows"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "nxvEkGXPM3Xh"
      },
      "source": [
        "## A Brief Note on Python Versions\n",
        "\n",
        "As of Janurary 1, 2020, Python has [officially dropped support](https://www.python.org/doc/sunset-python-2/) for `python2`. We'll be using Python 3.7 for this iteration of the course. You can check your Python version at the command line by running `python --version`. In Colab, we can enforce the Python version by clicking `Runtime -> Change Runtime Type` and selecting `python3`. Note that as of April 2020, Colab uses Python 3.6.9 which should run everything without any errors."
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "1L4Am0QATgOc",
        "outputId": "3b6527dd-933c-4a83-c3e8-4ac46d5ca9ba"
      },
      "source": [
        "!python --version"
      ],
      "execution_count": 6,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "Python 3.6.9\n"
          ],
          "name": "stdout"
```

```
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "JAFKYgrpL9eY"
      },
      "source": [
        "##Basics of Python"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "RbFS6tdgL9ea"
      },
      "source": [
        "Python is a high-level, dynamically typed multiparadigm programming language. Python code is often said to be almost like pseudocode, since it allows you to express very powerful ideas in very few lines of code while being very readable. As an example, here is an implementation of the classic quicksort algorithm in Python:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "cYb0pjh1L9eb",
        "outputId": "800edef0-7817-481e-c5ec-a72d5cfd1a66"
      },
      "source": [
        "def quicksort(arr):\n",
        "    if len(arr) <= 1:\n",
        "        return arr\n",
        "    pivot = arr[len(arr) // 2]\n",
        "    left = [x for x in arr if x < pivot]\n",
        "    middle = [x for x in arr if x == pivot]\n",
        "    right = [x for x in arr if x > pivot]\n",
        "    return quicksort(left) + middle + quicksort(right)\n",
        "\n",
        "print(quicksort([3,6,8,10,1,2,1]))"
      ],
      "execution_count": 7,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[1, 1, 2, 3, 6, 8, 10]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "NwS_hu4xL9eo"
      },
      "source": [
        "###Basic data types"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "DL5sMSZ9L9eq"
      },
      "source": [
        "####Numbers"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "MGS0XEWoL9er"
      },
      "source": [
        "Integers and floats work as you would expect from other languages:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "KheDr_zDL9es",
        "outputId": "e1a76d32-8f0c-43f6-d842-2dbc3ba8a086"
      },
      "source": [
        "x = 3\n",
        "print(x, type(x))"
      ],
      "execution_count": 8,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "3 <class 'int'>\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "sk_8DFcuL9ey",
        "outputId": "b2e19e57-d30b-45dd-ba95-8c673c100589"
      },
      "source": [
        "print(x + 1)   # Addition\n",
        "print(x - 1)   # Subtraction\n",
        "print(x * 2)   # Multiplication\n",
        "print(x ** 2)  # Exponentiation"
      ],
```

```
      "execution_count": 9,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "4\n",
            "2\n",
            "6\n",
            "9\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "U4Jl8K0tL9e4",
        "outputId": "56521728-c9bb-453e-aab2-06e18ac03daf"
      },
      "source": [
        "x += 1\n",
        "print(x)\n",
        "x *= 2\n",
        "print(x)"
      ],
      "execution_count": 10,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "4\n",
            "8\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "w-nZ0Sg_L9e9",
        "outputId": "d922d9c6-4f6c-4e85-8215-2e1b072299bb"
      },
      "source": [
        "y = 2.5\n",
        "print(type(y))\n",
        "print(y, y + 1, y * 2, y ** 2)"
      ],
      "execution_count": 11,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "<class 'float'>\n",
            "2.5 3.5 5.0 6.25\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "r2A9ApyaL9fB"
      },
      "source": [
        "Note that unlike many languages, Python does not have unary increment (x++) or decrement (x--) operators.\n",
        "\n",
        "Python also has built-in types for long integers and complex numbers; you can find all of the details in the [documentation](https://docs.python.org/3.7/library/stdtypes.html#numeric-types-int-float-long-complex)."
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "EqRS7qhBL9fC"
      },
      "source": [
        "####Booleans"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "Nv_LIVOJL9fD"
      },
      "source": [
        "Python implements all of the usual operators for Boolean logic, but uses English words rather than symbols (`&&`, `||`, etc.):"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "RvoImwgGL9fE",
        "outputId": "51de44d8-b4cc-4d2a-b7e6-6b9cda21249d"
      },
      "source": [
        "t, f = True, False\n",
        "print(type(t))"
      ],
      "execution_count": 12,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "<class 'bool'>\n"
          ],
          "name": "stdout"
        }
      ]
```

```
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "YQgmQfOgL9fI"
  },
  "source": [
    "Now we let's look at the operations:"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "6zYm7WzCL9fK",
    "outputId": "f02031ae-26c2-4b1f-8ebe-7b0b59c758b8"
  },
  "source": [
    "print(t and f) # Logical AND;\n",
    "print(t or f)  # Logical OR;\n",
    "print(not t)   # Logical NOT;\n",
    "print(t != f)  # Logical XOR;"
  ],
  "execution_count": 13,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "False\n",
        "True\n",
        "False\n",
        "True\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "UQnQWFEyL9fP"
  },
  "source": [
    "####Strings"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "AijEDtPFL9fP",
    "outputId": "7c7f414f-1313-48f3-ca65-ca2fa5e3de12"
  },
  "source": [
    "hello = 'hello'   # String literals can use single quotes\n",
    "world = \"world\"   # or double quotes; it does not matter\n",
    "print(hello, len(hello))"
  ],
  "execution_count": 14,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "hello 5\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "saDeaA7hL9fT",
    "outputId": "0074d11c-28cd-4350-a4e8-219ee13ba7cb"
  },
  "source": [
    "hw = hello + ' ' + world  # String concatenation\n",
    "print(hw)"
  ],
  "execution_count": 15,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "hello world\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "Nji1_UjYL9fY",
    "outputId": "2e7f8d0c-b301-4378-a919-3443e92a3c8f"
  },
  "source": [
    "hw12 = '{} {} {}'.format(hello, world, 12)  # string formatting\n",
    "print(hw12)"
  ],
  "execution_count": 16,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "hello world 12\n"
      ],
      "name": "stdout"
    }
  ]
```

```
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "bUpl35bIL9fc"
  },
  "source": [
    "String objects have a bunch of useful methods; for example:"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "VOxGatlsL9fd",
    "outputId": "8887cd48-a930-47e6-e07c-2056ac0358d4"
  },
  "source": [
    "s = \"hello\"\n",
    "print(s.capitalize())  # Capitalize a string\n",
    "print(s.upper())       # Convert a string to uppercase; prints \"HELLO\"\n",
    "print(s.rjust(7))      # Right-justify a string, padding with spaces\n",
    "print(s.center(7))     # Center a string, padding with spaces\n",
    "print(s.replace('l', '(ell)'))  # Replace all instances of one substring with another\n",
    "print('  world '.strip())  # Strip leading and trailing whitespace"
  ],
  "execution_count": 17,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "Hello\n",
        "HELLO\n",
        "  hello\n",
        " hello \n",
        "he(ell)(ell)o\n",
        "world\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "06cayXLtL9fi"
  },
  "source": [
    "You can find a list of all string methods in the [documentation](https://docs.python.org/3.7/library/stdtypes.html#string-methods)."
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "p-6hClFjL9fk"
  },
  "source": [
    "###Containers"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "FD9H18eQL9fk"
  },
  "source": [
    "Python includes several **built-in container types**: lists, dictionaries, sets, and tuples.\n",
    "\n",
    "1.   List item\n",
    "\n",
    "1.   List item\n",
    "2.   List item\n",
    "\n",
    "\n",
    "2.   List item\n",
    "\n"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "UsIWOe0LL9fn"
  },
  "source": [
    "####Lists"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "wzxX7rgWL9fn"
  },
  "source": [
    "A list is the Python equivalent of an array, but is resizeable and can contain elements of different types:"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "hk3A8pPcL9fp",
    "outputId": "f6d7fc43-2002-4c3e-9c81-92ef93fe390d"
  },
  "source": [
    "xs = [3, 1, 2]   # Create a list\n",
    "print(xs, xs[2])\n",
    "print(xs[-1])     # Negative indices count from the end of the list; prints \"2\""
  ],
  "execution_count": 18,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[3, 1, 2] 2\n",
        "2\n"
      ],
```

```
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "WrjhBUPsCY-N"
  },
  "source": [
    "Lists can be generated from arrays, as follows:"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "OCcmg_8u4oNc",
    "outputId": "8dbd2d3e-9c92-404b-cdc7-2e68b44b7c20"
  },
  "source": [
    "import numpy as np\n",
    "\n",
    "int_list = [] # list initialization\n",
    "int_list = [0,0,1,2,3] # list with commas\n",
    "int_list.append(4) # add 4 to end of the list\n",
    "int_list.pop(2) # remove element with index 2\n",
    "\n",
    "int_list2 = list(range(5)) # make list [0,1,2,3,4]\n",
    "int_array = np.array(int_list) # make array [] with no commas: [0 1 2 3 4]\n",
    "int_array2 = np.arange(5)  # make array [] with no commas: [0 1 2 3 4]\n",
    "int_list2 = int_array.tolist()  # convert array to list\n",
    "\n",
    "first = 0\n",
    "last = 4\n",
    "float_array = np.linspace(first,last,num=5)\n",
    "\n",
    "print('int_list=',int_list)\n",
    "print('int_list2=',int_list2)\n",
    "print('int_array=',int_array)\n",
    "print('int_array2=',int_array2)\n",
    "print('float_array=',float_array)"
  ],
  "execution_count": 20,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "int_list= [0, 0, 2, 3, 4]\n",
        "int_list2= [0, 0, 2, 3, 4]\n",
        "int_array= [0 0 2 3 4]\n",
        "int_array2= [0 1 2 3 4]\n",
        "float_array= [0. 1. 2. 3. 4.]\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "YCjCy_0_L9ft",
    "outputId": "454b22d4-37a0-4b23-db30-dfa3f673eaf5"
  },
  "source": [
    "xs[2] = 'foo'    # Lists can contain elements of different types\n",
    "print(xs)"
  ],
  "execution_count": 21,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[3, 1, 'foo']\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "de3ZD1ODykdX"
  },
  "source": [
    "Lists have methods, including append, insert, remove, sort"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "vJ0x5cF-L9fx",
    "outputId": "194ad53c-e12e-4355-ef1b-e431f5212c00"
  },
  "source": [
    "xs.append('bar') # Add a new element to the end of the list\n",
    "print(xs)  "
  ],
  "execution_count": 22,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[3, 1, 'foo', 'bar']\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
```

```
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "cxVCNRTNL9f1",
    "outputId": "ce90bc34-5dfa-49ea-bb0a-55ddd8320edd"
  },
  "source": [
    "x = xs.pop()     # Remove and return the last element of the list\n",
    "print(x, xs)"
  ],
  "execution_count": 23,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "bar [3, 1, 'foo']\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "ilyoyO34L9f4"
  },
  "source": [
    "As usual, you can find all the gory details about lists in the [documentation](https://docs.python.org/3.7/tutorial/datastructures.html#more-on-lists)."
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "ovahhxd_L9f5"
  },
  "source": [
    "####Slicing \n",
    "In addition to accessing list elements one at a time, Python provides concise\n",
    "syntax to access sublists; this is known as slicing:"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "ninq666bL9f6",
    "outputId": "8755a84d-be50-4e1b-ecf0-645c49cf1c27"
  },
  "source": [
    "nums = list(range(5))    # range is a built-in function that creates a list of integers\n",
    "print(nums)         # Prints \"[0, 1, 2, 3, 4]\"\n",
    "print(nums[2:4])    # Get a slice from index 2 to 4 (exclusive); prints \"[2, 3]\"\n",
    "print(nums[2:])     # Get a slice from index 2 to the end; prints \"[2, 3, 4]\"\n",
    "print(nums[:2])     # Get a slice from the start to index 2 (exclusive); prints \"[0, 1]\"\n",
    "print(nums[:])      # Get a slice of the whole list; prints [\"0, 1, 2, 3, 4]\"\n",
    "print(nums[:-1])    # Slice indices can be negative; prints [\"0, 1, 2, 3]\"\n",
    "nums[2:4] = [8, 9] # Assign a new sublist to a slice\n",
    "print(nums)         # Prints \"[0, 1, 8, 9, 4]\""
  ],
  "execution_count": 24,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[0, 1, 2, 3, 4]\n",
        "[2, 3]\n",
        "[2, 3, 4]\n",
        "[0, 1]\n",
        "[0, 1, 2, 3, 4]\n",
        "[0, 1, 2, 3]\n",
        "[0, 1, 8, 9, 4]\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "UONpMhF4L9f_"
  },
  "source": [
    "####Loops"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "_DYz1j6QL9f_"
  },
  "source": [
    "You can loop over the elements of a list like this:"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "4cCOysfWL9gA",
    "outputId": "90b74b80-70ee-4079-b590-422cfbe83095"
  },
  "source": [
    "animals = ['cat', 'dog', 'monkey']\n",
    "for animal in animals:\n",
    "    print(animal)"
  ],
  "execution_count": 25,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "cat\n",
        "dog\n",
        "monkey\n"
      ],
      "name": "stdout"
```

```
    }
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "KxIaQs7pL9gE"
   },
   "source": [
    "If you want access to the index of each element within the body of a loop, use the built-in `enumerate` function:"
   ]
  },
  {
   "cell_type": "code",
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "JjGnDluWL9gF",
    "outputId": "0de80943-807b-46c9-db66-8a1df79fcc32"
   },
   "source": [
    "animals = ['cat', 'dog', 'monkey']\n",
    "for idx, animal in enumerate(animals):\n",
    "    print('#{}: {}'.format(idx + 1, animal))"
   ],
   "execution_count": 26,
   "outputs": [
    {
     "output_type": "stream",
     "text": [
      "#1: cat\n",
      "#2: dog\n",
      "#3: monkey\n"
     ],
     "name": "stdout"
    }
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "arrLCcMyL9gK"
   },
   "source": [
    "####List comprehensions:"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "5Qn2jU_pL9gL"
   },
   "source": [
    "When programming, frequently we want to transform one type of data into another. As a simple example, consider the following code that computes square numbers:"
   ]
  },
  {
   "cell_type": "code",
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "IVNEwoMXL9gL",
    "outputId": "8f6c909e-b0c7-45da-fe02-3eb8f07b237e"
   },
   "source": [
    "nums = [0, 1, 2, 3, 4]\n",
    "squares = []\n",
    "for x in nums:\n",
    "    squares.append(x ** 2)\n",
    "print(squares)"
   ],
   "execution_count": 27,
   "outputs": [
    {
     "output_type": "stream",
     "text": [
      "[0, 1, 4, 9, 16]\n"
     ],
     "name": "stdout"
    }
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "7DmKVUFaL9gQ"
   },
   "source": [
    "You can make this code simpler using a list comprehension:"
   ]
  },
  {
   "cell_type": "code",
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "kZxsUfV6L9gR",
    "outputId": "1485ca70-6584-4c55-c76d-465e03265e28"
   },
   "source": [
    "nums = [0, 1, 2, 3, 4]\n",
    "squares = [x ** 2 for x in nums]\n",
    "print(squares)"
   ],
   "execution_count": 28,
   "outputs": [
    {
     "output_type": "stream",
     "text": [
      "[0, 1, 4, 9, 16]\n"
     ],
     "name": "stdout"
    }
   ]
  },
  {
   "cell_type": "markdown",
```

```
  "metadata": {
    "id": "-D8ARK7tL9gV"
  },
  "source": [
    "List comprehensions can also contain conditions:"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "yUtgOyyYL9gV",
    "outputId": "4cb1ac5e-6da0-4a78-db64-a189796dc8b0"
  },
  "source": [
    "nums = [0, 1, 2, 3, 4]\n",
    "even_squares = [x ** 2 for x in nums if x % 2 == 0]\n",
    "print(even_squares)"
  ],
  "execution_count": 29,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[0, 4, 16]\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "H8xsUEFpL9gZ"
  },
  "source": [
    "####Dictionaries"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "kkjAGMAJL9ga"
  },
  "source": [
    "A dictionary stores (key, value) pairs, similar to a `Map` in Java or an object in Javascript. You can use it like this:"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "XBYI1MrYL9gb",
    "outputId": "66b38373-3ae8-40ee-f534-a8618c503043"
  },
  "source": [
    "d = {}\n",
    "d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with some data\n",
    "print(d['cat'])       # Get an entry from a dictionary; prints \"cute\"\n",
    "print('cat' in d)     # Check if a dictionary has a given key; prints \"True\""
  ],
  "execution_count": 30,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "cute\n",
        "True\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "pS7e-G-HL9gf",
    "outputId": "fd954fcf-e04a-4148-9417-c29db51be7e2"
  },
  "source": [
    "d['fish'] = 'wet'    # Set an entry in a dictionary\n",
    "print(d['fish'])      # Prints \"wet\""
  ],
  "execution_count": 31,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "wet\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 198
    },
    "id": "tFY065ItL9gi",
    "outputId": "636b3b14-e8af-4e5b-e26f-7daba745e1b7"
  },
  "source": [
    "print(d['monkey'])  # KeyError: 'monkey' not a key of d"
  ],
  "execution_count": 32,
  "outputs": [
    {
      "output_type": "error",
      "ename": "KeyError",
      "evalue": "ignored",
```

```
          "traceback": [
            "\u001b[0;31m---------------------------------------------------------------------------\u001b[0m",
            "\u001b[0;31mKeyError\u001b[0m                                  Traceback (most recent call last)",
            "\u001b[0;32m<ipython-input-32-78fc9745d9cf>\u001b[0m in \u001b[0;36m<module>\u001b[0;34m()\u001b[0m\n\u001b[0;32m----> 1\u001b[0;31m \u001b[0mprint\u001b[0m\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0md\u001b[0m\u001b[0;34m[\u001b[0m\u001b[0;34m'monkey'\u001b[0m\u001b[0;34m]\u001b[0m\u001b[0;34m)\u001b[0m \u001b[0;31m# KeyError: 'monkey' not a key of d\u001b[0m\n\u001b[0;34m\u001b[0m\n\u001b[0m",
            "\u001b[0;31mKeyError\u001b[0m: 'monkey'"
          ]
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "8TjbEWqML9gl",
        "outputId": "f7e15301-e2f4-4cc0-ce2f-10892727a8d5"
      },
      "source": [
        "print(d.get('monkey', 'N/A'))  # Get an element with a default; prints \"N/A\"\n",
        "print(d.get('fish', 'N/A'))    # Get an element with a default; prints \"wet\""
      ],
      "execution_count": 33,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "N/A\n",
            "wet\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "0EItdNBJL9go",
        "outputId": "5c8ee432-85f3-4a49-d057-c31af371d7de"
      },
      "source": [
        "del d['fish']        # Remove an element from a dictionary\n",
        "print(d.get('fish', 'N/A')) # \"fish\" is no longer a key; prints \"N/A\""
      ],
      "execution_count": 34,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "N/A\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "wqm4dRZNL9gr"
      },
      "source": [
        "You can find all you need to know about dictionaries in the [documentation](https://docs.python.org/2/library/stdtypes.html#dict)."
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "IxwEqHlGL9gr"
      },
      "source": [
        "It is easy to iterate over the keys in a dictionary:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "rYfz7ZKNL9gs",
        "outputId": "abb23acb-c8c0-4ddf-8843-909c7d9a019a"
      },
      "source": [
        "d = {'person': 2, 'cat': 4, 'spider': 8}\n",
        "for animal, legs in d.items():\n",
        "    print('A {} has {} legs'.format(animal, legs))"
      ],
      "execution_count": 35,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "A person has 2 legs\n",
            "A cat has 4 legs\n",
            "A spider has 8 legs\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "dMNRxZ7013SH"
      },
      "source": [
        "Add pairs to the dictionary"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "id": "ojawHCIw19pz"
      },
      "source": [
```

```
    "d['bird']=2"
  ],
  "execution_count": 36,
  "outputs": []
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "E3CDH3bg2KLI"
  },
  "source": [
    "List keys"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "drutolgb2Mir",
    "outputId": "5ed7a9ac-2720-4081-89b8-f92e8eebd9cb"
  },
  "source": [
    "d.keys()"
  ],
  "execution_count": 37,
  "outputs": [
    {
      "output_type": "execute_result",
      "data": {
        "text/plain": [
          "dict_keys(['person', 'cat', 'spider', 'bird'])"
        ]
      },
      "metadata": {
        "tags": []
      },
      "execution_count": 37
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "HBGAaUIf2Ot7"
  },
  "source": [
    "List Values"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "lV0_kjg92QbA",
    "outputId": "0e8f6d4f-acab-46e2-e6b1-6fbb01abc1d4"
  },
  "source": [
    "d.values()"
  ],
  "execution_count": 38,
  "outputs": [
    {
      "output_type": "execute_result",
      "data": {
        "text/plain": [
          "dict_values([2, 4, 8, 2])"
        ]
      },
      "metadata": {
        "tags": []
      },
      "execution_count": 38
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "LzUGtMoq2dUG"
  },
  "source": [
    "Query values from keys"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "M9pjIh1_2fod",
    "outputId": "c0b7d545-5437-4f2b-8706-dd53e700d5c6"
  },
  "source": [
    "d['bird']"
  ],
  "execution_count": 39,
  "outputs": [
    {
      "output_type": "execute_result",
      "data": {
        "text/plain": [
          "2"
        ]
      },
      "metadata": {
        "tags": []
      },
      "execution_count": 39
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "17sxiOpzL9gz"
```

```
  },
  "source": [
    "Dictionary comprehensions: These are similar to list comprehensions, but allow you to easily construct dictionaries. For example:"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "8PB07imLL9gz",
    "outputId": "81e0df46-8229-4cea-de1e-928c7f430ec2"
  },
  "source": [
    "nums = [0, 1, 2, 3, 4]\n",
    "even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}\n",
    "print(even_num_to_square)"
  ],
  "execution_count": 40,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "{0: 0, 2: 4, 4: 16}\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "2ESxM-9j4nRD"
  },
  "source": [
    "Convert array to list"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "V9MHfUdvL9g2"
  },
  "source": [
    "####Sets (like dictionaries but with no values, add & remove)"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "Rpm4UtNpL9g2"
  },
  "source": [
    "A set is an unordered collection of distinct elements. As a simple example, consider the following:"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "MmyaniLsL9g2",
    "outputId": "38da4012-65af-40bc-913c-93cd3acc5744"
  },
  "source": [
    "animals = {'cat', 'dog'}\n",
    "print('cat' in animals)   # Check if an element is in a set; prints \"True\"\n",
    "print('fish' in animals)  # prints \"False\"\n"
  ],
  "execution_count": 41,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "True\n",
        "False\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "ElJEyK86L9g6",
    "outputId": "7141d06b-0ca7-490f-e23c-9ea9875eefdb"
  },
  "source": [
    "animals.add('fish')      # Add an element to a set\n",
    "print('fish' in animals)\n",
    "print(len(animals))       # Number of elements in a set;"
  ],
  "execution_count": 42,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "True\n",
        "3\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "5uGmrxdPL9g9",
    "outputId": "a4d44ca2-c862-4d82-f0b2-0160fdaf54b2"
  },
  "source": [
    "animals.add('cat')       # Adding an element that is already in the set does nothing\n",
```

```
        "print(len(animals))       \n",
        "animals.remove('cat')    # Remove an element from a set\n",
        "print(len(animals))       "
      ],
      "execution_count": 43,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "3\n",
            "2\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "zk2DbvLKL9g_"
      },
      "source": [
        "_Loops_: Iterating over a set has the same syntax as iterating over a list; however since sets are unordered, you cannot make assumptions about the order in which you visit the elements of the set:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "K47KYNGyL9hA",
        "outputId": "afd5c1d4-ce09-467b-d768-e0cd9cfd889a"
      },
      "source": [
        "animals = {'cat', 'dog', 'fish'}\n",
        "for idx, animal in enumerate(animals):\n",
        "    print('#{}: {}'.format(idx + 1, animal))"
      ],
      "execution_count": 44,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "#1: fish\n",
            "#2: dog\n",
            "#3: cat\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "puq4S8buL9hC"
      },
      "source": [
        "Set comprehensions: Like lists and dictionaries, we can easily construct sets using set comprehensions:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "iw7k90k3L9hC",
        "outputId": "13e8d292-939d-4c18-850c-e9c9d82a46dc"
      },
      "source": [
        "from math import sqrt\n",
        "print({int(sqrt(x)) for x in range(30)})"
      ],
      "execution_count": 45,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "{0, 1, 2, 3, 4, 5}\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "qPsHSKB1L9hF"
      },
      "source": [
        "####Tuples"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "kucc0LKVL9hG"
      },
      "source": [
        "A tuple is an (immutable) ordered list of values. A tuple is in many ways similar to a list; one of the most important differences is that tuples can be used as keys in dictionaries and as elements of sets, while lists cannot. Here is a simple example:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "9wHUyTKxL9hH",
        "outputId": "cfafde43-ca2f-4736-fddb-0c8ef6595987"
      },
      "source": [
        "d = {(x, x + 1): x for x in range(10)}  # Create a dictionary with tuple keys\n",
        "print(d)\n",
        "\n",
        "tt = ()  # initialization of empty tuple\n",
        "t1 = (66,)  # initialization of tuple with a single value\n",
```

```
        "t = (5, 6)       # Create a tuple\n",
        "tt = tt+t1+t\n",
        "print(\"tt=\",tt)\n",
        "print(\"tt[2]=\",tt[2])\n",
        "print(\"tt[1:3]=\",tt[1:3])\n",
        "print(\"66 in tt\", 66 in tt)\n",
        "\n",
        "print(type(t))\n",
        "print(d[t])       \n",
        "print(d[(1, 2)])"
      ],
      "execution_count": 46,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "{(0, 1): 0, (1, 2): 1, (2, 3): 2, (3, 4): 3, (4, 5): 4, (5, 6): 5, (6, 7): 6, (7, 8): 7, (8, 9): 8, (9, 10): 9}\n",
            "tt= (66, 5, 6)\n",
            "tt[2]= 6\n",
            "tt[1:3]= (5, 6)\n",
            "66 in tt True\n",
            "<class 'tuple'>\n",
            "5\n",
            "1\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/",
          "height": 198
        },
        "id": "HoO8zYKzL9hJ",
        "outputId": "33bc5d9b-b462-4bcd-a59b-a6dbf5bbb944"
      },
      "source": [
        "t[0] = 1"
      ],
      "execution_count": 47,
      "outputs": [
        {
          "output_type": "error",
          "ename": "TypeError",
          "evalue": "ignored",
          "traceback": [
            "\u001b[0;31m---------------------------------------------------------------------------\u001b[0m",
            "\u001b[0;31mTypeError\u001b[0m                                 Traceback (most recent call last)",
            "\u001b[0;32m<ipython-input-47-c8aeb8cd20ae>\u001b[0m in \u001b[0;36m<module>\u001b[0;34m()\u001b[0m\n\u001b[0;32m----> 1\u001b[0;31m
\u001b[0mt\u001b[0m\u001b[0;34m[\u001b[0m\u001b[0;36m0\u001b[0m\u001b[0;34m]\u001b[0m \u001b[0;34m=\u001b[0m
\u001b[0;36m1\u001b[0m\u001b[0;34m\u001b[0m\u001b[0;34m\u001b[0m\u001b[0;34m\u001b[0m\n\u001b[0m",
            "\u001b[0;31mTypeError\u001b[0m: 'tuple' object does not support item assignment"
          ]
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "AXA4jrEOL9hM"
      },
      "source": [
        "###Functions"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "WaRms-QfL9hN"
      },
      "source": [
        "Python functions are defined using the `def` keyword. For example:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "kiMDUr58L9hN",
        "outputId": "6b9fdd36-7f70-4a66-b94e-445878721b55"
      },
      "source": [
        "def sign(x):\n",
        "    if x > 0:\n",
        "        return 'positive'\n",
        "    elif x < 0:\n",
        "        return 'negative'\n",
        "    else:\n",
        "        return 'zero'\n",
        "\n",
        "for x in [-1, 0, 1]:\n",
        "    print(sign(x))"
      ],
      "execution_count": 48,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "negative\n",
            "zero\n",
            "positive\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "U-QJFt8TL9hR"
      },
      "source": [
        "We will often define functions to take optional keyword arguments, like this:"
      ]
    },
```

```
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "PfsZ3DazL9hR",
        "outputId": "752a08ba-91cd-4a16-df0c-e85e7102c4fd"
      },
      "source": [
        "def hello(name, loud=False):\n",
        "    if loud:\n",
        "        print('HELLO, {}'.format(name.upper()))\n",
        "    else:\n",
        "        print('Hello, {}!'.format(name))\n",
        "\n",
        "hello('Bob')\n",
        "hello('Fred', loud=True)"
      ],
      "execution_count": 49,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "Hello, Bob!\n",
            "HELLO, FRED\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "ObA9PRtQL9hT"
      },
      "source": [
        "###Classes"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "riWbiWUTnqEj"
      },
      "source": [
        "Creating a new class creates a new\n",
        "type of object, bunding data and functionality that allowing new\n",
        "instances of the type made. Each class instance can have attributes attached to it, so we can make class instances as well as instances to variables and methods for
maintaining the state of the class.  Instances of the method can have attributes and can modifying the state of the class, as clearly described the [documentation]([class]
(https://docs.python.org/3/tutorial/classes.html) )."
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "hAzL_lTkL9hU"
      },
      "source": [
        "The syntax for defining classes in Python is straightforward:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/",
          "height": 137
        },
        "id": "RWdbaGigL9hU",
        "outputId": "d8f57384-dc9a-47b6-95b8-042005cd2b19"
      },
      "source": [
        "class Greeter:\n",
        "    \"\"\" My greeter class \"\"\"\n",
        "    # Constructor (method of construction of class in a specific state)\n",
        "    v1 ='papa' # class variable shared by all instances\n",
        "    def __init__(self, name_inp): # name_inp: argument given to Greeter for class instantiation\n",
        "        self.name = name_inp  # Create an instance variable maintaining the state\n",
        "                              # instance variables are unique to each instance\n",
        "    # Instance method\n",
        "    # note that the first argument of the function method is the instance object\n",
        "    def greet(self, loud=False): \n",
        "        if loud:\n",
        "          print('HELLO, {}'.format(self.name.upper()))\n",
        "          self.name = 'Haote'\n",
        "        else:\n",
        "          print('Hello, {}!'.format(self.name))\n",
        "          self.name = 'Victor'\n",
        "\n",
        "# Class instantiation (returning a new instance of the class assigned to g): \n",
        "# Constructs g of type Greeter & initialzes its state \n",
        "# as defined by the class variables (does not execute methods)\n",
        "g = Greeter('Fred')  \n",
        "\n",
        "# Call an instance method of the class in its current state: \n",
        "# prints \"Hello, Fred!\" and updates state varible to 'Victor' since loud=False\n",
        "g.greet() # equivalent to Greeter.greet(g) since the first arg of greet is g   \n",
        "\n",
        "# Call an instance method; prints \"HELLO, VICTOR\" and updates variable to 'Haote'\n",
        "g.greet(loud=True)   # equivalent to Greeter.greet(g,loud=True) \n",
        "                     #since the first arg of greet is g \n",
        "print(g.v1)\n",
        "g.greet()            # Call an instance method; prints \"Hello, Haote!\"\n",
        "                     # A method object is created by packing (pointers to) the \n",
        "                     # instance object g and the function object greet\n",
        "\n",
        "g2 = Greeter('Lea')   # Class instance reinitializes variable to 'Lea'\n",
        "\n",
        "g2.greet()            # Call an instance method; prints \"Hello, Lea!\"\n",
        "g2.__doc__\n",
        "g2.x=20               # Data attributes spring into existence upon assignment\n",
        "print(g2.x)\n",
        "del g2.x              # Deletes attribute\n",
        "g2.v1"
      ],
      "execution_count": 50,
      "outputs": [
        {
          "output_type": "stream",
```

```
          "text": [
            "Hello, Fred!\n",
            "HELLO, VICTOR\n",
            "papa\n",
            "Hello, Haote!\n",
            "Hello, Lea!\n",
            "20\n"
          ],
          "name": "stdout"
        },
        {
          "output_type": "execute_result",
          "data": {
            "application/vnd.google.colaboratory.intrinsic+json": {
              "type": "string"
            },
            "text/plain": [
              "'papa'"
            ]
          },
          "metadata": {
            "tags": []
          },
          "execution_count": 50
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "ejkev803U6ch"
      },
      "source": [
        "For loops (iterators). Behind the scenes, the\n",
        "for statement calls iter() on the container object."
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "UX5RI4XJU8Dy",
        "outputId": "b9b1e876-e64c-43b4-8425-d737cde52279"
      },
      "source": [
        "for element in [1,2,3]: # elements of list\n",
        "  print(element)\n",
        "for element in (1,2,3): # elements of tuple\n",
        "  print(element)\n",
        "for key in {'first':1, 'second':2, 'third':3}: # elements of dictionary\n",
        "  print('key=',key)\n",
        "for char in '1234':\n",
        "  print(char)\n",
        "#for line in open(''myfile.txt)\n",
        "#  print(line,end='')"
      ],
      "execution_count": 51,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "1\n",
            "2\n",
            "3\n",
            "1\n",
            "2\n",
            "3\n",
            "key= first\n",
            "key= second\n",
            "key= third\n",
            "1\n",
            "2\n",
            "3\n",
            "4\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "FOph0sLxV4p_"
      },
      "source": [
        "#Modules"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "oSBsJmQ9V7oE"
      },
      "source": [
        "A module is a .py file containing Python definitions and statements that can be imported into a Python script, as described in the Python [documentation]
(https://docs.python.org/3/tutorial/modules.html). \n",
        "\n",
        "As an example, use a text editor and write a module with the line:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "id": "58dMT2PYckVH"
      },
      "source": [
        "greeting = \"Good Morning!\""
      ],
      "execution_count": 52,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "6eDSBXtzctX6"
      },
      "source": [
        "Save the document with the name  mymod.py\n",
```

```
        "\n",
        "Next, go the the folder where you saved that file and open a notebook with the lines:\n"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "id": "ufLNkYdPB2-q"
      },
      "source": [
        "import mymod as my\n",
        "print(my.greeting)"
      ],
      "execution_count": null,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "iKRIgaKGc7LC"
      },
      "source": [
        "you will see that the notebook has imported the variable greetingfrom the module ``mymod.py`` and has invoked the variable as an attribute of the module mymod that was imported as my when printing ``Good Morning!!``.\n",
        "\n",
        "Modules are very convenient since they allow you to import variables, functions and classes that you might have developed for previous projects, without having to copy them into each program. So, you can build from previous projects, or split your work into several files for easier maintenance. \n",
        "\n",
        "Within a module, the module’s name (as a string) is available as the value of the global variable ``__name__``. "
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "3cfrOV4dL9hW"
      },
      "source": [
        "##Numpy"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "fY12nHhyL9hX"
      },
      "source": [
        "Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. If you are already familiar with MATLAB, you might find this [tutorial](http://wiki.scipy.org/NumPy_for_Matlab_Users) useful to get started with Numpy."
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "lZMyAdqhL9hY"
      },
      "source": [
        "To use Numpy, we first need to import the `numpy` package:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "id": "58QdX8BLL9hZ"
      },
      "source": [
        "import numpy as np"
      ],
      "execution_count": 54,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "DDx6v1EdL9hb"
      },
      "source": [
        "###Arrays"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "f-Zv3f7LL9hc"
      },
      "source": [
        "A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension."
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "_eMTRnZRL9hc"
      },
      "source": [
        "We can initialize numpy arrays from nested Python lists, and access elements using square brackets:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "-l3JrGxCL9hc",
        "outputId": "e912ba5b-a909-4eed-f799-bdbcc5c451fe"
      },
      "source": [
        "a = np.array([1, 2, 3])  # Create a rank 1 array\n",
        "print(type(a), a.shape, a[0], a[1], a[2])\n",
        "a[0] = 5                 # Change an element of the array\n",
        "print(a)                  "
      ],
      "execution_count": 55,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "<class 'numpy.ndarray'> (3,) 1 2 3\n",
            "[5 2 3]\n"
```

```
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "ma6mk-kdL9hh",
    "outputId": "fc9ba04f-0fa5-4a51-ed9d-121a4ba52309"
  },
  "source": [
    "b = np.array([[1,2,3],[4,5,6]])   # Create a rank 2 array\n",
    "print(b)"
  ],
  "execution_count": 56,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[[1 2 3]\n",
        " [4 5 6]]\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "ymfSHAwtL9hj",
    "outputId": "bd575e4f-78cc-4e4a-e7a5-acf87af2f784"
  },
  "source": [
    "print(b.shape)\n",
    "print(b[0, 0], b[0, 1], b[1, 0])"
  ],
  "execution_count": 57,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "(2, 3)\n",
        "1 2 4\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "F2qwdyvuL9hn"
  },
  "source": [
    "Numpy also provides many functions to create arrays:"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "mVTN_EBqL9hn",
    "outputId": "a28bea7e-59ee-4299-f659-81801a6c3e61"
  },
  "source": [
    "a = np.zeros((2,2))  # Create an array of all zeros\n",
    "print(a)"
  ],
  "execution_count": 58,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[[0. 0.]\n",
        " [0. 0.]]\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "skiKlNmlL9h5",
    "outputId": "bc175b58-8425-443d-eeb7-d715aa80129d"
  },
  "source": [
    "b = np.ones((1,2))   # Create an array of all ones\n",
    "print(b)"
  ],
  "execution_count": 59,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[[1. 1.]]\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "HtFsr03bL9h7",
```

```
    "outputId": "4532d76d-4f54-4e17-d40e-29e2d4a01c0d"
  },
  "source": [
    "c = np.full((2,2), 7) # Create a constant array\n",
    "print(c)"
  ],
  "execution_count": 60,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[[7 7]\n",
        " [7 7]]\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "-QcALHvkL9h9",
    "outputId": "e81e34df-d6da-4d9d-9e21-ac7bd941a0a1"
  },
  "source": [
    "d = np.eye(2)        # Create a 2x2 identity matrix\n",
    "print(d)"
  ],
  "execution_count": 61,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[[1. 0.]\n",
        " [0. 1.]]\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "RCpaYg9qL9iA",
    "outputId": "acdaaf02-701e-4ae7-c5ce-37cb59905e87"
  },
  "source": [
    "e = np.random.random((2,2)) # Create an array filled with random values\n",
    "print(e)"
  ],
  "execution_count": 62,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[[0.32071297 0.96986179]\n",
        " [0.32331846 0.50510489]]\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "jI5qcSDfL9iC"
  },
  "source": [
    "###Array indexing"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "M-E4MUeVL9iC"
  },
  "source": [
    "Numpy offers several ways to index into arrays."
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "QYv4JyIEL9iD"
  },
  "source": [
    "Slicing: Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array:"
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "wLWA0udwL9iD",
    "outputId": "54ea3af5-4cc1-4e1a-9318-29c73eddae7c"
  },
  "source": [
    "import numpy as np\n",
    "\n",
    "# Create the following rank 2 array with shape (3, 4)\n",
    "# [[ 1  2  3  4]\n",
    "#  [ 5  6  7  8]\n",
    "#  [ 9 10 11 12]]\n",
    "a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])\n",
    "\n",
    "# Use slicing to pull out the subarray consisting of the first 2 rows\n",
    "# and columns 1 and 2; b is the following array of shape (2, 2):\n",
    "# [[2 3]\n",
    "#  [6 7]]\n",
    "b = a[:2, 1:3]\n",
    "print(b)"
  ],
```

```
      "execution_count": 63,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[2 3]\n",
            " [6 7]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "KahhtZKYL9iF"
      },
      "source": [
        "A slice of an array is a view into the same data, so modifying it will modify the original array."
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "1kmtaFHuL9iG",
        "outputId": "79a911cc-70b7-4727-b398-e0cecca564ca"
      },
      "source": [
        "print(a[0, 1])\n",
        "b[0, 0] = 77    # b[0, 0] is the same piece of data as a[0, 1]\n",
        "print(a[0, 1]) "
      ],
      "execution_count": 64,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "2\n",
            "77\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "_Zcf3zi-L9iI"
      },
      "source": [
        "You can also mix integer indexing with slice indexing. However, doing so will yield an array of lower rank than the original array. Note that this is quite different from the way that MATLAB handles array slicing:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "G6lfbPuxL9iJ",
        "outputId": "d8345f5c-c940-430c-b6aa-feec890a14cb"
      },
      "source": [
        "# Create the following rank 2 array with shape (3, 4)\n",
        "a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])\n",
        "print(a)"
      ],
      "execution_count": 65,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[ 1  2  3  4]\n",
            " [ 5  6  7  8]\n",
            " [ 9 10 11 12]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "NCye3NXhL9iL"
      },
      "source": [
        "Two ways of accessing the data in the middle row of the array.\n",
        "Mixing integer indexing with slices yields an array of lower rank,\n",
        "while using only slices yields an array of the same rank as the\n",
        "original array:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "EOiEMsmNL9iL",
        "outputId": "4a333ff0-9306-47d8-f205-419698d3a311"
      },
      "source": [
        "row_r1 = a[1, :]    # Rank 1 view of the second row of a  \n",
        "row_r2 = a[1:2, :]  # Rank 2 view of the second row of a\n",
        "row_r3 = a[[1], :]  # Rank 2 view of the second row of a\n",
        "print(row_r1, row_r1.shape)\n",
        "print(row_r2, row_r2.shape)\n",
        "print(row_r3, row_r3.shape)"
      ],
      "execution_count": 66,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[5 6 7 8] (4,)\n",
            "[[5 6 7 8]] (1, 4)\n",
```