```
{
  "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": "U1PvreR9L9eW"
      },
      "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[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)\u
001b[0m  \u001b[0;31m# KeyError: 'monkey' not a key of d\u001b[0m\u001b[0;34m\u001b[0m\u001b[0;34m\u001b[0m\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[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",
            "[[5 6 7 8]] (1, 4)\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "JXu73pfDL9iN",
```

```
        "outputId": "bbef8bbd-927d-4804-c98a-d1d77e60ed97"
      },
      "source": [
        "# We can make the same distinction when accessing columns of an array:\n",
        "col_r1 = a[:, 1]\n",
        "col_r2 = a[:, 1:2]\n",
        "print(col_r1, col_r1.shape)\n",
        "print()\n",
        "print(col_r2, col_r2.shape)"
      ],
      "execution_count": 67,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[ 2  6 10] (3,)\n",
            "\n",
            "[[ 2]\n",
            " [ 6]\n",
            " [10]] (3, 1)\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "VP3916bOL9iP"
      },
      "source": [
        "Integer array indexing: When you index into numpy arrays using slicing, the resulting array view will always be a subarray of the original array. In contrast, integer array indexing allows you to construct arbitrary arrays using the data from another array. Here is an example:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "TBnWonIDL9iP",
        "outputId": "22e098e0-2207-4a7e-afdd-026ae931e5a2"
      },
      "source": [
        "a = np.array([[1,2], [3, 4], [5, 6]])\n",
        "\n",
        "# An example of integer array indexing.\n",
        "# The returned array will have shape (3,) and \n",
        "print(a[[0, 1, 2], [0, 1, 0]])\n",
        "\n",
        "# The above example of integer array indexing is equivalent to this:\n",
        "print(np.array([a[0, 0], a[1, 1], a[2, 0]]))"
      ],
      "execution_count": 68,
```

```
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[1 4 5]\n",
            "[1 4 5]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "n7vuati-L9iR",
        "outputId": "176ed525-24ef-499f-8cca-c5a0c44ce77d"
      },
      "source": [
        "# When using integer array indexing, you can reuse the same\n",
        "# element from the source array:\n",
        "print(a[[0, 0], [1, 1]])\n",
        "\n",
        "# Equivalent to the previous integer array indexing example\n",
        "print(np.array([a[0, 1], a[0, 1]]))"
      ],
      "execution_count": 69,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[2 2]\n",
            "[2 2]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "kaipSLafL9iU"
      },
      "source": [
        "One useful trick with integer array indexing is selecting or mutating one element from each row of a matrix:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
```

```
        "id": "ehqsV7TXL9iU",
        "outputId": "8440a998-9354-458d-e166-67b2bec93ccb"
      },
      "source": [
        "# Create a new array from which we will select elements\n",
        "a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])\n",
        "print(a)"
      ],
      "execution_count": 70,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[ 1  2  3]\n",
            " [ 4  5  6]\n",
            " [ 7  8  9]\n",
            " [10 11 12]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "pAPOoqy5L9iV",
        "outputId": "c144edc3-d739-4117-84f0-5dbeb3a0c5a4"
      },
      "source": [
        "# Create an array of indices\n",
        "b = np.array([0, 2, 0, 1])\n",
        "\n",
        "# Select one element from each row of a using the indices in b\n",
        "print(a[np.arange(4), b])  # Prints \"[ 1  6  7 11]\""
      ],
      "execution_count": 71,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[ 1  6  7 11]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
```

```
        "id": "6v1PdI1DL9ib",
        "outputId": "dd20062a-bbb5-4e1c-cd06-81eaa218a431"
      },
      "source": [
        "# Mutate one element from each row of a using the indices in b\n",
        "a[np.arange(4), b] += 10\n",
        "print(a)"
      ],
      "execution_count": 72,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[11  2  3]\n",
            " [ 4  5 16]\n",
            " [17  8  9]\n",
            " [10 21 12]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "kaE8dBGgL9id"
      },
      "source": [
        "Boolean array indexing: Boolean array indexing lets you pick out arbitrary elements of an array. Frequently this type of indexing is used to
select the elements of an array that satisfy some condition. Here is an example:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "32PusjtKL9id",
        "outputId": "35fe5d8e-64dd-4550-d62d-2d3d87071e04"
      },
      "source": [
        "import numpy as np\n",
        "\n",
        "a = np.array([[1,2], [3, 4], [5, 6]])\n",
        "\n",
        "bool_idx = (a > 2)  # Find the elements of a that are bigger than 2;\n",
        "                    # this returns a numpy array of Booleans of the same\n",
        "                    # shape as a, where each slot of bool_idx tells\n",
        "                    # whether that element of a is > 2.\n",
        "\n",
        "print(bool_idx)"
      ],
      "execution_count": 73,
      "outputs": [
```

```
      {
        "output_type": "stream",
        "text": [
          "[[False False]\n",
          " [ True  True]\n",
          " [ True  True]]\n"
        ],
        "name": "stdout"
      }
    ]
  },
  {
    "cell_type": "code",
    "metadata": {
      "colab": {
        "base_uri": "https://localhost:8080/"
      },
      "id": "cb2IRMXaL9if",
      "outputId": "ffd7dd2d-0434-4ce4-e7cc-286124669aa6"
    },
    "source": [
      "# We use boolean array indexing to construct a rank 1 array\n",
      "# consisting of the elements of a corresponding to the True values\n",
      "# of bool_idx\n",
      "print(a[bool_idx])\n",
      "\n",
      "# We can do all of the above in a single concise statement:\n",
      "print(a[a > 2])"
    ],
    "execution_count": 74,
    "outputs": [
      {
        "output_type": "stream",
        "text": [
          "[3 4 5 6]\n",
          "[3 4 5 6]\n"
        ],
        "name": "stdout"
      }
    ]
  },
  {
    "cell_type": "markdown",
    "metadata": {
      "id": "CdofMonAL9ih"
    },
    "source": [
      "For brevity we have left out a lot of details about numpy array indexing; if you want to know more you should read the documentation."
    ]
  },
  {
    "cell_type": "markdown",
    "metadata": {
      "id": "jTctwqdQL9ih"
    },
```

```
      "source": [
        "###Datatypes"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "kSZQ1WkIL9ih"
      },
      "source": [
        "Every numpy array is a grid of elements of the same type. Numpy provides a large set of numeric datatypes that you can use to construct arrays. Numpy tries to guess a datatype when you create an array, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype. Here is an example:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "4za4O0m5L9ih",
        "outputId": "aef59fd5-c6e3-44af-b8d7-ef191df7cb52"
      },
      "source": [
        "x = np.array([1, 2])  # Let numpy choose the datatype\n",
        "y = np.array([1.0, 2.0])  # Let numpy choose the datatype\n",
        "z = np.array([1, 2], dtype=np.int64)  # Force a particular datatype\n",
        "\n",
        "print(x.dtype, y.dtype, z.dtype)"
      ],
      "execution_count": 75,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "int64 float64 int64\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "RLVIsZQpL9ik"
      },
      "source": [
        "You can read all about numpy datatypes in the [documentation](http://docs.scipy.org/doc/numpy/reference/arrays.dtypes.html)."
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "TuB-fdhIL9ik"
```

```
      },
      "source": [
        "###Array math"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "18e8V8elL9ik"
      },
      "source": [
        "Basic mathematical functions operate elementwise on arrays, and are available both as operator overloads and as functions in the numpy module:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "gHKvBrSKL9il",
        "outputId": "ded0ad53-844f-407b-855f-86d91b340830"
      },
      "source": [
        "x = np.array([[1,2],[3,4]], dtype=np.float64)\n",
        "y = np.array([[5,6],[7,8]], dtype=np.float64)\n",
        "\n",
        "# Elementwise sum; both produce the array\n",
        "print(x + y)\n",
        "print(np.add(x, y))"
      ],
      "execution_count": 76,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[ 6.  8.]\n",
            " [10. 12.]]\n",
            "[[ 6.  8.]\n",
            " [10. 12.]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "1fZtIAMxL9in",
        "outputId": "fdf421a2-a909-428d-a4d6-ff2de6a4fe11"
      },
      "source": [
```

```
      "# Elementwise difference; both produce the array\n",
      "print(x - y)\n",
      "print(np.subtract(x, y))"
     ],
     "execution_count": 77,
     "outputs": [
      {
       "output_type": "stream",
       "text": [
        "[[-4. -4.]\n",
        " [-4. -4.]]\n",
        "[[-4. -4.]\n",
        " [-4. -4.]]\n"
       ],
       "name": "stdout"
      }
     ]
    },
    {
     "cell_type": "code",
     "metadata": {
      "colab": {
       "base_uri": "https://localhost:8080/"
      },
      "id": "nil4AScML9io",
      "outputId": "1108d019-93fe-424e-8109-c3ddcb065075"
     },
     "source": [
      "# Elementwise product; both produce the array\n",
      "print(x * y)\n",
      "print(np.multiply(x, y))"
     ],
     "execution_count": 78,
     "outputs": [
      {
       "output_type": "stream",
       "text": [
        "[[ 5. 12.]\n",
        " [21. 32.]]\n",
        "[[ 5. 12.]\n",
        " [21. 32.]]\n"
       ],
       "name": "stdout"
      }
     ]
    },
    {
     "cell_type": "code",
     "metadata": {
      "colab": {
       "base_uri": "https://localhost:8080/"
      },
      "id": "0JoA4lH6L9ip",
      "outputId": "bed0a214-d133-44ae-d6fc-172abf852e4f"
     },
```

```
      "source": [
        "# Elementwise division; both produce the array\n",
        "# [[ 0.2         0.33333333]\n",
        "#  [ 0.42857143  0.5       ]]\n",
        "print(x / y)\n",
        "print(np.divide(x, y))"
      ],
      "execution_count": 79,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[0.2        0.33333333]\n",
            " [0.42857143 0.5       ]]\n",
            "[[0.2        0.33333333]\n",
            " [0.42857143 0.5       ]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "g0iZuA6bL9ir",
        "outputId": "da01b0b9-d981-45dd-c085-d138cba07519"
      },
      "source": [
        "# Elementwise square root; produces the array\n",
        "# [[ 1.          1.41421356]\n",
        "#  [ 1.73205081  2.        ]]\n",
        "print(np.sqrt(x))"
      ],
      "execution_count": 80,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[1.         1.41421356]\n",
            " [1.73205081 2.        ]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "a5d_uujuL9it"
      },
      "source": [
        "Note that unlike MATLAB, `*` is elementwise multiplication, not matrix multiplication. We instead use the dot function to compute inner
```

```
products of vectors, to multiply a vector by a matrix, and to multiply matrices. dot is available both as a function in the numpy module and as an
instance method of array objects:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "I3FnmoSeL9iu",
        "outputId": "c2f7c686-67d8-4e1e-a494-f2a723c26295"
      },
      "source": [
        "x = np.array([[1,2],[3,4]])\n",
        "y = np.array([[5,6],[7,8]])\n",
        "\n",
        "v = np.array([9,10])\n",
        "w = np.array([11, 12])\n",
        "\n",
        "# Inner product of vectors; both produce 219\n",
        "print(v.dot(w))\n",
        "print(np.dot(v, w))"
      ],
      "execution_count": 81,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "219\n",
            "219\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "vmxPbrHASVeA"
      },
      "source": [
        "You can also use the `@` operator which is equivalent to numpy's `dot` operator."
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "vyrWA-mXSdtt",
        "outputId": "c334ce3b-6bf3-40d1-f9b2-c6d30f29b7a8"
      },
      "source": [
```

```
        "print(v @ w)"
      ],
      "execution_count": 82,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "219\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "zvUODeTxL9iw",
        "outputId": "ccaf2057-4ac4-4134-acd6-7006d78b74d5"
      },
      "source": [
        "# Matrix / vector product; both produce the rank 1 array [29 67]\n",
        "print(x.dot(v))\n",
        "print(np.dot(x, v))\n",
        "print(x @ v)"
      ],
      "execution_count": 83,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[29 67]\n",
            "[29 67]\n",
            "[29 67]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "3V_3NzNEL9iy",
        "outputId": "74402ec2-6162-400c-b0b4-22af79621a7f"
      },
      "source": [
        "# Matrix / matrix product; both produce the rank 2 array\n",
        "# [[19 22]\n",
        "#  [43 50]]\n",
        "print(x.dot(y))\n",
```

```
        "print(np.dot(x, y))\n",
        "print(x @ y)"
      ],
      "execution_count": 84,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[19 22]\n",
            " [43 50]]\n",
            "[[19 22]\n",
            " [43 50]]\n",
            "[[19 22]\n",
            " [43 50]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "FbE-1If_L9i0"
      },
      "source": [
        "Numpy provides many useful functions for performing computations on arrays; one of the most useful is `sum`:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "DZUdZvPrL9i0",
        "outputId": "381adbd9-51f7-4567-8e2a-21eedc974814"
      },
      "source": [
        "x = np.array([[1,2],[3,4]])\n",
        "\n",
        "print(np.sum(x))  # Compute sum of all elements; prints \"10\"\n",
        "print(np.sum(x, axis=0))  # Compute sum of each column; prints \"[4 6]\"\n",
        "print(np.sum(x, axis=1))  # Compute sum of each row; prints \"[3 7]\""
      ],
      "execution_count": 85,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "10\n",
            "[4 6]\n",
            "[3 7]\n"
          ],
          "name": "stdout"
        }
```

```
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "ahdVW4iUL9i3"
      },
      "source": [
        "You can find the full list of mathematical functions provided by numpy in the [documentation]
(http://docs.scipy.org/doc/numpy/reference/routines.math.html).\n",
        "\n",
        "Apart from computing mathematical functions using arrays, we frequently need to reshape or otherwise manipulate data in arrays. The simplest
example of this type of operation is transposing a matrix; to transpose a matrix, simply use the T attribute of an array object:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "63Yl1f3oL9i3",
        "outputId": "b9574fc8-a9d3-4240-c476-829ed09a84b3"
      },
      "source": [
        "print(x)\n",
        "print(\"transpose\\n\", x.T)"
      ],
      "execution_count": 86,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[1 2]\n",
            " [3 4]]\n",
            "transpose\n",
            " [[1 3]\n",
            " [2 4]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "mkk03eNIL9i4",
        "outputId": "d18e9f69-f128-4884-e4f2-132110cae943"
      },
      "source": [
        "v = np.array([[1,2,3]])\n",
        "print(v )\n",
```

```
        "print(\"transpose\\n\", v.T)"
      ],
      "execution_count": 87,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[1 2 3]]\n",
            "transpose\n",
            " [[1]\n",
            " [2]\n",
            " [3]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "REfLrUTcL9i7"
      },
      "source": [
        "###Broadcasting"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "EygGAMWqL9i7"
      },
      "source": [
        "Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations.
Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger
array.\n",
        "\n",
        "For example, suppose that we want to add a constant vector to each row of a matrix. We could do it like this:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "WEEvkV1ZL9i7",
        "outputId": "28f7a095-448b-4563-b238-f014c2421cfb"
      },
      "source": [
        "# We will add the vector v to each row of the matrix x,\n",
        "# storing the result in the matrix y\n",
        "x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])\n",
        "v = np.array([1, 0, 1])\n",
        "y = np.empty_like(x)   # Create an empty matrix with the same shape as x\n",
        "\n",
```

```
        "# Add the vector v to each row of the matrix x with an explicit loop\n",
        "for i in range(4):\n",
        "    y[i, :] = x[i, :] + v\n",
        "\n",
        "print(y)"
      ],
      "execution_count": 88,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[ 2  2  4]\n",
            " [ 5  5  7]\n",
            " [ 8  8 10]\n",
            " [11 11 13]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "2OlXXupEL9i-"
      },
      "source": [
        "This works; however when the matrix `x` is very large, computing an explicit loop in Python could be slow. Note that adding the vector v to each row of the matrix `x` is equivalent to forming a matrix `vv` by stacking multiple copies of `v` vertically, then performing elementwise summation of `x` and `vv`. We could implement this approach like this:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "vS7UwAQQL9i-",
        "outputId": "5d74fea6-0892-45e7-940d-0d283431d2fa"
      },
      "source": [
        "vv = np.tile(v, (4, 1))  # Stack 4 copies of v on top of each other\n",
        "print(vv)                # Prints \"[[1 0 1]\n",
        "                         #          [1 0 1]\n",
        "                         #          [1 0 1]\n",
        "                         #          [1 0 1]]\""
      ],
      "execution_count": 89,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[1 0 1]\n",
            " [1 0 1]\n",
            " [1 0 1]\n",
```

```
            " [1 0 1]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "N0hJphSIL9jA",
        "outputId": "1f095ded-d101-4566-e03b-9b3f5d4b5402"
      },
      "source": [
        "y = x + vv  # Add x and vv elementwise\n",
        "print(y)"
      ],
      "execution_count": 90,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[ 2  2  4]\n",
            " [ 5  5  7]\n",
            " [ 8  8 10]\n",
            " [11 11 13]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "zHos6RJnL9jB"
      },
      "source": [
        "Numpy broadcasting allows us to perform this computation without actually creating multiple copies of v. Consider this version, using broadcasting:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "vnYFb-gYL9jC",
        "outputId": "fe2f94b3-9ad0-4193-e6b8-47ec2e9f9348"
      },
      "source": [
        "import numpy as np\n",
        "\n",
```

```
        "# We will add the vector v to each row of the matrix x,\n",
        "# storing the result in the matrix y\n",
        "x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])\n",
        "v = np.array([1, 0, 1])\n",
        "y = x + v  # Add v to each row of x using broadcasting\n",
        "print(y)"
      ],
      "execution_count": 91,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[ 2  2  4]\n",
            " [ 5  5  7]\n",
            " [ 8  8 10]\n",
            " [11 11 13]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "08YyIURKL9jH"
      },
      "source": [
        "The line `y = x + v` works even though `x` has shape `(4, 3)` and `v` has shape `(3,)` due to broadcasting; this line works as if v actually had shape `(4, 3)`, where each row was a copy of `v`, and the sum was performed elementwise.\n",
        "\n",
        "Broadcasting two arrays together follows these rules:\n",
        "\n",
        "1. If the arrays do not have the same rank, prepend the shape of the lower rank array with 1s until both shapes have the same length.\n",
        "2. The two arrays are said to be compatible in a dimension if they have the same size in the dimension, or if one of the arrays has size 1 in that dimension.\n",
        "3. The arrays can be broadcast together if they are compatible in all dimensions.\n",
        "4. After broadcasting, each array behaves as if it had shape equal to the elementwise maximum of shapes of the two input arrays.\n",
        "5. In any dimension where one array had size 1 and the other array had size greater than 1, the first array behaves as if it were copied along that dimension\n",
        "\n",
        "If this explanation does not make sense, try reading the explanation from the [documentation](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html) or this [explanation](http://wiki.scipy.org/EricsBroadcastingDoc).\n",
        "\n",
        "Functions that support broadcasting are known as universal functions. You can find the list of all universal functions in the [documentation](http://docs.scipy.org/doc/numpy/reference/ufuncs.html#available-ufuncs).\n",
        "\n",
        "Here are some applications of broadcasting:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
```

```
        "id": "EmQnwoM9L9jH",
        "outputId": "26503c2b-50a3-4be9-ef5b-d96a8d6a7764"
      },
      "source": [
        "# Compute outer product of vectors\n",
        "v = np.array([1,2,3])  # v has shape (3,)\n",
        "w = np.array([4,5])    # w has shape (2,)\n",
        "# To compute an outer product, we first reshape v to be a column\n",
        "# vector of shape (3, 1); we can then broadcast it against w to yield\n",
        "# an output of shape (3, 2), which is the outer product of v and w:\n",
        "\n",
        "print(np.reshape(v, (3, 1)) * w)"
      ],
      "execution_count": 92,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[ 4  5]\n",
            " [ 8 10]\n",
            " [12 15]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "PgotmpcnL9jK",
        "outputId": "36da2fb8-eac8-4b61-b519-f189778995bd"
      },
      "source": [
        "# Add a vector to each row of a matrix\n",
        "x = np.array([[1,2,3], [4,5,6]])\n",
        "# x has shape (2, 3) and v has shape (3,) so they broadcast to (2, 3),\n",
        "# giving the following matrix:\n",
        "\n",
        "print(x + v)"
      ],
      "execution_count": 93,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[2 4 6]\n",
            " [5 7 9]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
```

```
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "T5hKS1QaL9jK",
    "outputId": "6f9ac49b-f02a-4196-d3d8-976e948dbc62"
  },
  "source": [
    "# Add a vector to each column of a matrix\n",
    "# x has shape (2, 3) and w has shape (2,).\n",
    "# If we transpose x then it has shape (3, 2) and can be broadcast\n",
    "# against w to yield a result of shape (3, 2); transposing this result\n",
    "# yields the final result of shape (2, 3) which is the matrix x with\n",
    "# the vector w added to each column. Gives the following matrix:\n",
    "\n",
    "print((x.T + w).T)"
  ],
  "execution_count": 94,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[[ 5  6  7]\n",
        " [ 9 10 11]]\n"
      ],
      "name": "stdout"
    }
  ]
},
{
  "cell_type": "code",
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "JDUrZUl6L9jN",
    "outputId": "e0442155-f890-4d3b-fd6e-e3681f7a531a"
  },
  "source": [
    "# Another solution is to reshape w to be a row vector of shape (2, 1);\n",
    "# we can then broadcast it directly against x to produce the same\n",
    "# output.\n",
    "print(x + np.reshape(w, (2, 1)))"
  ],
  "execution_count": 95,
  "outputs": [
    {
      "output_type": "stream",
      "text": [
        "[[ 5  6  7]\n",
        " [ 9 10 11]]\n"
      ],
      "name": "stdout"
```

```
        }
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "VzrEo4KGL9jP",
        "outputId": "3da2fe9c-2798-48b4-c289-c78d9c5aaa53"
      },
      "source": [
        "# Multiply a matrix by a constant:\n",
        "# x has shape (2, 3). Numpy treats scalars as arrays of shape ();\n",
        "# these can be broadcast together to shape (2, 3), producing the\n",
        "# following array:\n",
        "print(x * 2)"
      ],
      "execution_count": 96,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "[[ 2  4  6]\n",
            " [ 8 10 12]]\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "89e2FXxFL9jQ"
      },
      "source": [
        "Broadcasting typically makes your code more concise and faster, so you should strive to use it where possible."
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "iF3ZtwVNL9jQ"
      },
      "source": [
        "This brief overview has touched on many of the important things that you need to know about numpy, but is far from complete. Check out the [numpy reference](http://docs.scipy.org/doc/numpy/reference/) to find out much more about numpy."
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "tEINf4bEL9jR"
      },
```

```
      "source": [
        "##Matplotlib"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "0hgVWLaXL9jR"
      },
      "source": [
        "Matplotlib is a plotting library. In this section give a brief introduction to the `matplotlib.pyplot` module, which provides a plotting system 
similar to that of MATLAB."
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "id": "cmh_7c6KL9jR"
      },
      "source": [
        "import matplotlib.pyplot as plt"
      ],
      "execution_count": 97,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "jOsaA5hGL9jS"
      },
      "source": [
        "By running this special iPython command, we will be displaying plots inline:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "id": "ijpsmwGnL9jT"
      },
      "source": [
        "%matplotlib inline"
      ],
      "execution_count": 98,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "U5Z_oMoLL9jV"
      },
      "source": [
        "###Plotting"
      ]
    },
    {
```

```
      "cell_type": "markdown",
      "metadata": {
        "id": "6QyFJ7dhL9jV"
      },
      "source": [
        "The most important function in `matplotlib` is plot, which allows you to plot 2D data. Here is a simple example:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/",
          "height": 282
        },
        "id": "pua52BGeL9jW",
        "outputId": "5dcd4321-3f8f-4100-af17-87adfafee963"
      },
      "source": [
        "# Compute the x and y coordinates for points on a sine curve\n",
        "x = np.arange(0, 3 * np.pi, 0.1)\n",
        "y = np.sin(x)\n",
        "\n",
        "# Plot the points using matplotlib\n",
        "plt.plot(x, y)"
      ],
      "execution_count": 99,
      "outputs": [
        {
          "output_type": "execute_result",
          "data": {
            "text/plain": [
              "[<matplotlib.lines.Line2D at 0x7f78639a1748>]"
            ]
          },
          "metadata": {
            "tags": []
          },
          "execution_count": 99
        },
        {
          "output_type": "display_data",
          "data": {
            "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAYIAAAD4CAYAAADhNOGaAAAABHNCSVQICAgIfAhkiAAAAAlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9uMy4yLjIsIGh
0dHA6Ly9tYXRwbG90bGliLm9yZy+WH4yJAAAgAElEQVR4nO3dd3hU95X4//cZVVRBSEKNJnoRCBDNdhIXjCk2uAdXkjjixs4m9qd442d0461+cZLPKY8d2bMfYwdh0Y9wLYBBFEh0hiioSAoSEunR
+f2jIV8GiajR3ynk9zzyauWXukRjm3Hvup4iqYowxJni5nA7AGGOMsywRGGNMkLNEYIwQc4SgTHGBDlLBMYYE+RCnQ7gUiQmJuqgQYOcDsMYY/zK5s2bj6pq0pnL/TIRDBo0iih7p
abqUhY4wJcpYIjDEmyFkiMMaYIGeJwBhjgpwlAmOMCXIeSQQi8hcRqRSR7WdZLyEpFJF8EZnYad1j/WxgmPtxP/AHABFJAB4FpgJTgEdFpI+HYjLGGHMBPNK
PQFU/EJFB59hkPvBX7Rjzeo0BI9BaRVOBKYK2qHgMQkbV0JJSXPBFXoGlubWfjgWNU1jZysqGF2sZWUnv3YurgBDL69EJRL69EJEnA7RGJ9x5GQjH+47yqmmVtralXZVhiTHMD2zL5FhIU6H51O81aEsHSju9Lr
Evexsyz9FRO6n42qCAQMG9EyUPkhVyS+pYcmWEpbllXG8vqXL7VLiIpkxOpmHrh5Gv7hIL0dpjG+oqW/hpU2HeWN7BduKT3S5TWSYi8uHJDIvO40bxqXhctkJlN/0LFbVJ4EnAXJycoJiNp3K2kb+47X
tvLnzCBGhLq4d3Y8bs9MZkhxDQoMZGhHDxaz8aDx9hQVM3Lm4pZvLmEL1+RyQOfyyQ2MszpX8EYr1BVXttayuMrd1F9qpms9Hi+O3M4M0b3IzgmgDx31XJeyQne3V3J27sr+caifG
xDv8GzhJPzVDmLg2tUNWxXaz7E/Ceqr7kfr2HjrLQlcCVqvpAV9udTU5OjgbbYEBOqyrK8Mh5dtoP65ja+OWMGN/OsbvPVLtmtCgNz+
+cex5v9jb25VXt5Tws9W7OV7fzD3TBvKvEzxBdKeDKduaOICA3skpGIbFbVnE8t91imAs8CMyh48bwb1V1ivtm8GBgd6nDCuiLcCk0/cMziaQE0Fbu/KDJQW8nFtMdv/e/OK28QxNjrmo9ygoqeEbi7ZScryBn96cxS2
TMnooWmOcta34BF98ZiNt7cr3Zo/kjskDLqrUU9PQwq/W7uXZdpAn+7JIb5X4F5J92giEJGX6Di7TwSO0NESKAxAVf8oHXcx/4+OG8H1wBdVNde975eAH7Givx
yvn7VEL41YzihIZfWsKumvoV/eXEz6/ZX87Urh/DdmSMIDdmsFmoCygd7q/jqC5tJjIng+fumLBv9CW/1+tbS3l4cR6ZiTE8+6XJpMb38mCkvqPHrwi8KRATQVNrGw/+bStrdx7h+7NnvT7PVva2vn
```

```
h0h28tPEwX7p8MD+8YbQHIjXGecvzyvj2K9sYmhzLc1+cTLIHGkh8tO8oX31hM7GRobz0lWkMSrz0xOKrzpYIrGexD2hrV772whbW7jzCf80b45EkABAW4uInN43li5cP4i8fH+CpD4s88r7GOGnd/qN
86+VtTOjfh0X3T/NIEgC4YlgiLz8wjcaWNr707CZqztJCLxBZIvABP1+zm7d3V/LY/DEsvGyQR99bRPiPuaOZPTaFH6/cxYr8Mo++vzHedODoKf7lhS0MSozmqS94vp4/Ji2eP92TQ/Hxer76wmaaW9s
9+v6+yhKBw5bnlfGn94u4a+oA7p0+qEeOEeISfvX5bCYP6sO3X85j08Fz3os3xifVNLRw33ObcAk8vTDnvK3oLtWUwQn89y3jWF9UzX++vh1/LJ9fLEsEDtpZdpJ/W5xPzsA+PHrDmB49VmRYCH++N4f
0Pr146G9bOVHf3KPHM8aT2tqVh17ayuHqev5w96Ru3Ri+EDdPzOChq4fycm4xz6472KPH8gWWCBxysrGFB17IJa5XKL+/eyLhoT3/T9E7KpzfLpjA0bomvr+kICjOdExgePqjIj7YW8Vj88cyLbOvV47
5rRnDmTEqmZ+u3s2+I7VeOaZTLBE45KerdlN6vIHf3zWJ5FjvDQmRlRHPd2aOYPX2Cv6+ucRrxzXmUu09Ussv1uxl5uh+3DGlv9eO63IJP7tlHLERoXzrlW20tAXu/QJLBA74uPAoL208zFc+k8mkgd4
fbPX+z2YyLTOBHy3bwcGjp7x+fGMuVEtbO995JY+YyFAevynL6wMrJsZE8PhNWWwvPcnv3t7n1WN7kyUCLzvV1Mr3Xs0nMzGab1073JEYQlzCL2/PJtQlfOfvebS3W4nI+KY/vLefgtIafnzjWJJiIxy
JYdbYFG6ZmMET7+1n6+HjjsTQ0ywReNnP39hN6YkGfn7rOEeHwk3r3Yv/vH40mw8dZ/EWKxEZ37Oz7CS/fXsf88anMScr1dFYHp03mpS4SB5enB+QJSJLBF60+dAxnlt/iIXTB5HjA4PB3TIxg4kDevP
fq3cHVecZ4/tUlR8t30FcrzD+a17Ptqi7EHGRYfxo3hgKK+t4fv0hp8PxOEsEXtLerjy2fCcpcZH826wRTocDdNwMe2z+WI7XN/O/a/c4HY4x/7CqoIKNB47xnZnD6RMd7nQ4AMwYlcxnhiXyq7f2Ul3
X5HQ4HmWJwEuW5pWSV1LDw9eNICrcd6aBGJsez93TBvLChkPsKKtxOhxjaGxp4yerdjEyJZYFk31nEioR4dEbRtPQ3MYv3gysEydLBF7Q0NzGz9/YQ1Z6PDdN6HICNkd959oR9IkK54dLd1jfAuO4P39
QROmJBh69YQwhPjZi7tDkWBZeNohFm4opKAmcEydLBF7w1IdFlNc08h9zR/nkUNDxUWE8fN0INh86zpodR5wOxwSx8poGfv/efmaPTWH6EO90HLtY/3rNMBKiwvnR8sA5cbJE0MOOnGzkD+/vZ9aYFKZ
6qUfkpbh1UgaZidH8cu0e2qw5qXHI/765lzZVfjBnlNOhnFV8rzC+M7PjxOntXZVOh+MRlgh62G/e3kdLWzuPzB7pdCjnFBri4tszh7P3SB3L8kqdDscEoQNHT7FkSwn3TBtI/4Qop8M5p9tyMhiQEMU
v1+4NiH44HkkEIjJLRPaISKGIPNLF+l+JyDb3Y6+InOi0rq3TumWeiMdXlJ5o4O+5xXx+cn+/mORizthURqXG8au1+wKyrbTxbb97ex/hoS6+6qH5OHpSWIiLb1wzjJ3lJ1mzo8LpcLqt24lAREKAJ4D
ZwGjgDhH5p6mwVPVbqpqtqtnA74AlnVY3nF6nqvO6G48v+f27hQD8y5VDHY7kwrhcwsPXDefwsXpeyS12OhwTRPZX1fH6tlLunT7IsR7EF+vGCelkJkXzq7f2+n051RNXBFOAQlUtUtVmYBEw/xzb3wG
85IHj+rSyEw28klvMbTn9Se/tP/OfXjUimUkD+/C7twtpbGlzOhwTJH779j4iw0J44LOZTodywUJcwjdndJRTVxaUOx1Ot3giEaQDnU8fS9zLPkVEBgKDgXc6LY4UkVwR2SAiN57tICJyv3u73KqqKg+
E3bP+8N5+AL52pe9f5nYmInx35ggqTjbaVYHxin1HalmWV8a90wfRN8Y/rgZOuz4rlRH9Yvn1W3tp9eNyqrdvFi8AFqtq51PNge7JlO8Efi0iXX5zquqTqpqjqjlJSUneiPWSldc08PKmYm6dlEFGH9+
+6dWVaZkJTBzQmyc/KPLrD7fxD795ex9RYSHc70dXA6e5XMK3rh1GUdUpv74q8EQiKAU6DxKe4V7WlQWcURZS1VL3zyLgPWCCB2Jy1J/eL6Jdla/5yb2BM4kIX/3cEEqON/j1h9v4vkPVp1hVUM490we
R4CNDSVysmaNTGJIUzZMfFPltvwJPJIJNwDARGSwi4XR82X+q9Y+IjAT6AOs7LesjIhHu54nA5cBOD8TkmGOnmlm06TA3TUj3+SZw5zJjVD+GJsfwx/f998NtfN/THx0gxCV86fJBTodyyVwu4f7PZrK
j7CQfF1Y7Hc4l6XYiUNVW4EFgDbALeEVVd4jIYyLSuRXQAmCR/vO3yiggV0TygHeBn6mqXyeCFzccorGlna/44WVuZy6X8MBnM9lVfpL39/r+PRnjf46fauaV3GJuzE4nOc57s/T1hBsnpJMUG8GfPtj
vdCiXxCOjn6nqKmDVGct+eMbrH3Wx3zogyxMx+ILGljaeW3+Izw1PYni/WKfD6bb52en8cu1e/vj+fq4ckex0OCbAPB8gJ00AEaEhfOGyQfzPmj3sKKthTFq80yFdFOtZ7EHL8so4WtfEVz7j/x9sgPB
QF/ddMZgNRccCdmYm44zGljaeW3eQq0YExkkTwN1TBxIdHsKfPyhyOpSLZonAQ1SVpz88wMiUWC4f6rtjCl2sBVMGEBcZytMfHXA6FBNAlmwppfpUM/d/1r+aV59LfFQYC6YMYHl+OSXH650O56JYIvC
QD/YdZc+RWr78mUyvT7Ddk2IiQrk9pz9vbK/gyMlGp8MxAaC9XXnqwyKy0uOZlun8TH2e9KUrBiPAc+sOOh3KRbFE4CFPfVhEcmwE88anOR2Kx907fRBtqry4IfCm6DPe98G+KoqOnuLLnxkcUCdNAOm
9e3HdmBReyS2hodl/euZbIvCAwspaPtx3lHunDyQ8NPD+pAP6RnH1iGT+tvEwTa3+8+E2vun59YdIjIlg9lhnJ6TvKfdOH0hNQ4tfjeIbeN9aDnhhw2HCQoQFU3xnWj1PW3jZII7WNbPKOpiZbig+Vs8
7eyq5Y0r/gDxpApgyOIER/WJ5bt0hv+mDE5j/El5U39zKq5tLmJOVSqKfjZNyMa4YmkhmUjTPrrPykLl0L35yGJcId04N3JMmEeGe6QPZWX6SLX7S2s4SQTct21ZGbVMrd08b6HQoPcrlEhZOH0Re8Qm
2FZ84/w7GnKGxpY2XNx1mxqhkUuP9Z0TeS3HThHRiI0L563r/OHGyRNANqsrzGw4xMiWWnIF9nA6nx90yKYOYiFD+6mctIoxvWJlfzvH6Fu6dPsjpUHpcdEQot0zKYFVBOVW1TU6Hc16WCLphW/EJdpS
d5K5pAwOu9UNXYiJCuWlCOisKyjlR3+x0OMbP/HXDITKTornMRyel97R7pg+kpU1ZtPGw06GclyWCbnhhw2Giw0O4aUKX0y8EpAVT+tPc2s5rW/2nRYRxXkFJDXnFJ7gnSE6aAIYkxXDF0EQWbSr2+Rn
MLBFcouOnmlmeX8ZNE9OJifDIkE1+YUxaPOMy4lm0sdhvWkQY5y3adJiIUBc3T8xwOhSvWjClP6UnGvio8KjToZyTJYJL9NrWUppb27lramDfJO7KgskD2HOklq1209hcgIbmNpZtK2NuVirxvcKcDse
rrh3djz5RYby8ybfLQ5YILoGq8kpuMeMz4hmVGud0OF43LzuNqPAQv6h9GuetKiintqmV2yf3P//GASYiNISbJ2awducRqut896axJYJLUFBaw+6KWm7LCb4PNnTcNL5hXBrL88qpbWxxOhzj417eVMy
gvlFMHRxY4wpdqM9P7k9Lm/r0fTWPJAIRmSUie0SkUEQe6WL9F0SkSkS2uR9f7rRuoYjscz8WeiKenvZKbjERoS7mZQfeuEIXasGU/jS0tLEsr8zpUIwPK6qqY+PBY9w+uX/Q3CQ+0/B+sUwc0JtFm3z
3vlq3E4GIhABPALOB0cAdIjK6i01fVtVs9+Mp974JwKPAVGAK8KiI+HSD/MaWNpZuK2NOVipxkcFV7+wsu39vRqbEsmhjsdOhGB/2Sm4JIS7h1iC7SXymBZMHUFhZ57M9jT1xRTAFKFTVIlVtBhYB8y9
w3+uAtap6TFWPA2uBWR6Iqce8sb2C2sZWbg/SstBpIsKCyf0pKK1hV/lJp8MxPqilrZ3Fm0u4akSy309F2V1zx6USHR7isydOnkgE6UDn367EvexMt4hIvogsFpHT36IXuq/PeHlTMQMSgrfe2dm87HT
CQoRXN5c4HYrxQe/uruRoXRMLgvAm8ZmiI0KZl53Givxy6ppanQ7nU7x1s3g5MEhVx9Fx1v/cxb6BiNwvIrkikltV5cxk6oer61lfVM3tORm4XMFZ7+wsITqcq0Yk8/q2Mlrb2p0Ox/iYxZtLSIqN4Mo
RSU6H4hNunZRBQ0sbb2yvcDqUT/FEIigFOqf8DPeyf1DValU93XbqKWDShe7b6T2eVNUcVc1JSnLmg7V4SwkiHWPumA63TMrgaF0TH+xzJjkb33TsVDPv7qnkxuw0QkOscSLAxAF9GNg3iiVbfO8K2hP
/QpuAYSIyWETCgQXAss4biEjnGSjmAbvcz9cAM0Wkj/sm8Uz3Mp+jqry2tYQrhiYG/MiJF+OqEcn0iQrj1c2+2zTOeN+K/DJa2jToehKfi4hw84QM1hdVU3qiwelw/km3E4GqtgIP0vEFvgt4RVV3iMh
jIjLPvdm/isgOEckD/hX4gnvfY8D/R0cy2QQ85l7mc3IPHaf4WENQjSt0IcJDXczPTmftziPU1FufAtPh1S2ljEqNC8oOl+dy88R0VOF1H+tT4JFrNlVdparDVXWIqj7uXvZDVV3mfv59VR2jquNV9Sp
V3d1p37+o6lD34xlPxNMTlmwppVdYCNeNSXE6FJ9z66QMmtvaWZ5vfQoMFFbWkVd8glsm2knTmfonRDFlcAKvbinxqT4FVry7AI0tbazIL2PW2BSig2iAuQs1Ji2OEf1iWWythwzw2tYSXEJQd7g8l1s
mplNUdcqnJniyRHAB3tldSW1jq5WFzkJEuGVSOtuKT7C/qs7pcIyD2tuV17aU8tnhSSTHBnffgbOZk5VKRKiLJVt8pzxkieACLNlSSnJsBJcPTXQ6FJ91Y3Y6IrDUx2qfxrs2HKimrKbRbhKfQ2xkGNe
NSWF5fhlNrW1OhwNYIjivY6eaeW9PJfOz0wixvgNnlRwXyWVD+rI0r8ynap/Gu5ZsKSU2IpSZo/s5HYpPu3liOifqW3hvj280u7ZEcB4r8stobbdmcBdifnY6h6rrfar2abyn0d1ZatbYFCLDQpwOx6d
dMTSRvtHhLNvmGw0sLBGcx2tbSxmZEmvN4C7ArLEphIe6WOojH27jXe/srqSuqZUb7V7aeYWGuLh+XCpv7TriE0O5WyI4h8PV9Ww9fIL52fbBvhBxkWFcMzK54yrKhpwIOku3lZIUG8G0zOCYnL675mW
n09Tazps7jjgdiiWCczndLv6G8ann2dKcNj87naN1zT4/R6vxrJqGFt7dU8UN4+xe2oWaOKA3GX16sdQH5vSwRHAOS7eVkjOwDxl9opwOxW9cNTKJuMhQn6l9Gu9Ys6OC5tZ26ztwEUSE+dlpfLSviqp
aZ6extERwFrsrTrL3SJ19sC9SRGgIc7JSWbOjgoZm32gaZ3resm1lDOwbxfiMeKdD8Svzs9NpV1jpcK98SwRnsWxbGSEuYU6WlYUu1rzsNE41t7F2l/O1T9PzKk82sm7/UeaPTwva6Sgv1fB+HQ1RnC4
PWSLogqqyLK+My4cmkhgT4XQ4fmfa4L70i4tguQ/UPk3PW5FfTrvakBKXan52GlsPn+Bwdb1jMVgi6MKWwycoOd7A/PH2wb4ULpcwNyuN9/dUcdIHmsaZnrU0r4wxaXEMTY51OhS/dIP7e2ZZnnO98i0
RdGF5XhkRoS5mjrHekZfq+vGpNLf5RtM403MOV9eTV3yCeXbSdMnSe/di0sA+rMgvdywGSwRnaG1rZ0V+OVePTCY2MszpcPzWhP69Se/dixU2NHVAO93Eeu44u5fWHTeMS2V3RS2FlbWOHN8SwRk2Hjj
G0bqmf1yumUsjItwwPo2P9h3l+Klmp8MxPWRFfjkTBvS2JtbdNCcrFRFYnufMVYFHEoGIzBKRPSJSKCKPdLH+2yKyU0TyReRtERnYaV2biGxzP5adua+3Lc8vJyo8hKtGJDsdit+7flwqre3KGzt8b7J
u0337q+rYVX6S68fZSVN3JcdFMnVwAivynRm0sduJQERCgCeA2cBo4A4RGX3GZluBHFUdBywGft5pXYOqZrsf83BQa1s7b2wvZ8aofvQKt0GzumtMWhyZidHWeihArcwvRwTmWhNrj7h+XBr7q06xu8L
75SFPXBFMAQpVtUhVm4FFwPzOG6jqu6p6um3UBsAnh/Jct7+a4/UtVu/0EBHh+nGpbCiqprK20elwjIetyC9j8sAEUuJtAhpPmD02hRCXOHLi5IlEkA4Ud3pd4l52NvcBqzu9jhSRXBHZICI3nm0nEbn
fvV1uVVXPjOG9Mr+cmIhQPjc8qUfePxjdMD6NdoXVBVYeCiR7j9Sy90gd19s4XB7TNyaCy4b0ZUV+udfLQ169WSwidwM5wP90WjxQVXOAO4Ffi8iQrvZV1SdVNUdVc5KSPP9F3dzazhs7Krh2dD8bS92
DhvWLZUS/WGs9FGBW5JXhEpg91hKBJ10/LpXDx+opKK3x6nE9kQhKgf6dXme4l/0TEZkB/DswT1X/McKSqpa6fxYB7wETPBDTRfu48Cg1DS1cb2Uhj5s7LpXcQ8epqLHyUCBQVVbklzMtsy9Jsdbz3pO
uG5NCqEu83qfAE4lgEzBMRAaLSDiwAPin1j8iMgH4Ex1JoLLT8j4iEuF+nghcDuz0QEwXbUV+ObGRoVwxzOYl9rQ5WamowurtznWYMZ6zq7yWoqOnrLVQD+gdFc5nhiWy0svloW4nAlVtBR4E1gC7gFd
UdYeIPCYip1sB/Q8QA/z9jGaio4BcEckD3gV+pqpeTwRNrW28ubOC68akEBFqZSFPG5ocw8iUWFYVWCIIBCsLOgZkvM563veIOVmplJ5oIK/Ee+WhUE+8iaquAladseyHnZ7POMt+64AsT8TQHR/uPUp
tY6u1FupBc7JS+eXavVTUNForEz+mqqwqqGB6Zl/62oCMPWLm6BR+EFLAqoJysvv39soxrWcxsKqgnLjIUC4fYmWhnnJ6OG8rD/m3XeW1HDh6yoZn70HxUWFcMdS75aGgTwRNrR3j5s8c0zHxuukZVh4
KDKsKyq0s5AXeLg8F/Tffx4XuspCd4fS4uVmpbDporYf8VUdZqJxpmQlWFuphM0enEBYiXjtxCvpEsDK/oqMsNNTKQj1tzjgrD/mz3RUdrYWsLNTz4qPCuNyL5aGgTgTNre2s3VnBtaOtLOQNQ5I6ykM
rHRx33Vy6VQXluKSjrbvpeafLQ/leKA8F9bffx4VHOdnYytxx9sH2lrlZ1rnMH6kqKws6OpHZ9K3ecZ27PLTSC+WhoE4EKwvKiY2wspA3zXaXFd6w8pBf2V1RS1GVlYW8yZvloaBNBM2t7bzpHlvIOpF
5z9DkGEb0i2XVdhuEzp+sdpeFZo21q2dvOl0e6umxh4I2Eazb31EWsjMc75udlcKmg8dsaGo/smp7BVMHW1nI22aO7keoS1jVw6P3Bm0iWF1QQUxEKJ8ZbmUhbzs99tAauyrwC/uO1FJYWcecLLsa8Lb
eUeFMH9KX1dt7tjwUlImgpa2dNTsrmDEq2cpCDhiWHMOQpOgeP8sxnrGqoAKx1kKOmZOVyqHqenaWn+yxYwRlIvik6Bgn6lv+cePSeJeIMDcrlU8OVHO0run8OxhHrd5ezuSBCSTH2RhRTpg5uh8hLun
RyZ2CMhGs2t4xQb3NROac2VmptCussYntfdr+qjp2V9Qy28pCjukbE8G0zARWFfRceSjoEkFbu7JmewVXj0y2mcgcNDIllsGJ0TaFpY97w30fx1oLOWv22FSKjp5i75G6Hnn/oEsEGw8co/pUs7UWcpi
```

```
IMHtsCuuLqjl2qtnpcMxZrCooZ+KA3qTG93I6lKB23ZgUROixsYeCLhGs3l5OZJiLK0dYWchpc7JSaWtX1u60qwJfdKj6FDvKTtpJkw9Iio1gyqCEHhunyyOJQERmicgeESkUkUe6WB8hIi+7138iIoM
6rfu+e/keEbnOE/GcTXu7snp7BVeNSCYq3CNz8phuGJMWx4CEKGs95KNWW1nIp8zJSmXvkToKK2s9/t7d/jYUkRDgCeBaoATYJCLLzphy8j7guKoOFZEFwH8DnxeR0XTMcTwGSAPePAdRyKoeW3bi6svn
wcapqm6y1kI8QEWZnpfD0hweoqW8hPirM6ZBMJ6sLyhmXEU9GnyinQzF0dMSMjgglpQfKdJ64IpgCFKpqkao2A4uA+WdsMx94zv18MXCNiIh7+SJVbVLVA0Ch+/16xKqCcsJDXVw9MrmnDmEu0uyxqbS
2K2t3HXE6FNNJyfF68kpqrCzkQ5JjI7l1UgYxEZ6vZngiEaQDxZ1el7iXdbmNe7L7GqDvBe4LgIjcLyK5IpJbVVV1SYG2tSuzxqT0yB/SXJrxGfGkxUfaIHQ+5nRrodlWFgoKfvONqKpPAk8C5OTkXFJ
j2sfmj/XaHKDmwnSUh1J5fv0hahtbiI208pAvWFVQzpi0OAb2jXY6FOMFnrgiKAX6d3qd4V7W5TYiEgrEA9UXuK9HdVSkjC+Zk5VCc1s77+yudDoUA5TXNLDl8AkrCwURTySCTcAwERksIuF03PxddsY
2y4CF7ue3Au9ox6n5MmCBu1XRYGAYsNEDMRk/MqF/H/rFRdjE9j7CykLBp9ulIVVtFZEHgTVACPAXVd0hIo8Buaq6DHgaeF5ECoFjdCQL3Nu9AuwEWoGv91SLIeO7XC5h9thUXtp4mFNNrUTbPRxHrS6
oYGRKLJlJMU6HYrzEI/0IVHWVqg5X1SGq+rh72Q/dSQBVbVTV21R1qKpOUdWiTvs+7t5vhKqu9kQ8xv/MHptCU2s77+6x8pCTKk82sunQMWaPtbJQMAm6nsXGN+UMSiAxJsLGHnLYmh0VqGJzDwQZSwT
GJ4S4hFlj+/HO7koamq066OKZgxsAABUDSURBVJRVBRUMTY5hWL9Yp0MxXmSJwPiMOVmpNLS08Z6VhxxxtK6JTw5U203iIGSJwPiMKYMS6BsdbhPbO2TNjgraFWs2GoQsERifERriYuaYFN7ZdYTGFis
PeduqgnIyE6MZmWJloWBjicD4lLlZqZxqbuP9vZc2jIi5NNV1TWwoOsbsrBTrdBmELBEYnzI1M4E+UWGsts5lXvXmzi00tauVhYKUJQLjU8JCXMwcncJbuyqtPORFqwrKGdQ3itGpcU6HYhxgicD4nDn
jUqlrauWjfUedDiUoHD/VzLr91czOSrWyUJCyRGB8zmVD+hLfK8zGHvKSN3dW0NauzLWyUNCyRGB8Tkd5qB9rdx6hqdXKQz1tZUEFAxKiGJNmZaFgZYnA+KS541KpbWrlw71WHupJJ+qbWVd4lDlWFgp
qlgiMT7p8aCLxvcJYaeWhHvXmji00tquNLRTkLBEYnxQW4uK6Mf14a6d1LutJKwrKGZAQRVZ6vNOhGAdZIjA+a+64tI7ykLUe6hHHTzXzceFR5o6zslCws0RgfNZlQ/rSOyqMlfllTocSkNbssNZCpkO
3EoGIJIjIWhHZ5/7Zp4ttskVkvYjsEJF8Efl8p3XPisgBEdnmfmR3Jx4TWMJCXMwak8JaKw/1iJUF5QxOjLbWQqbbVwSPAG+r6jDgbffrM9UD96rqGGAW8GsR6d1p/cOqmu1+bOtmHxPCbAzLGxh3pEdV0
T6/ZXM9daCxm6nwjmA8+5nz8H3HjmBqq6V1X3uZ+XAZVAUjePa4LE9CF96RNlncs87Y3TZaFxVhYy3U8E/VT19P/QCqDfuTYWkSlAOLC/0+LH3SWjX4lIxDn2vV9EckUkt6rKzg6DRViIi1ljO8pDNnO
Z56zMLyczyYacNh3OmwhE5C0R2d7FY37n7VRVAT3H+6QCzwNfVNV29+LvAyOByUAC8L2z7a+qT6pqjqrmJCXZBUUwuWFcGvXNbTaxvYdU1Taxoaia660sZNxCz7eBqs442zoROSIiqapa7v6i7/J/qoj
EASuBf1fVDZ3e+/TVRJOIPAN896KiN0FhamZfEmMiWJ5XZsMke8Ab28tp147mucZA90tDy4CF7ucLgaVnbiAi4cBrwF9VdfEZ61LdP4WO+wvbuxmPCUAhLuH6cam8s7uS2sYWp8Pxe8vyyhjeL4YRVhY
ybt1NBD8DrhWRfcAM92tEJEdEnnJvczvwWeALXTQTfVFECoACIBH4cTfjMQHqhvGpNLW289auI06H4tdKTzSw6eBx5o23qwHz/5y3NHQuqloNXNPF8lzgy+7nLwAvnGX/q7tzfBM8JvTvQ3rvXizPK+e
mCRlOh+O3TnfOu97KQqYT61ls/ILLXR76YG8VJ+qbnQ7Hby3LK2N8RjyDEqOdDsX4EEsExm/cMD6N1nblje0VTofil4qq6theepIbrCxkzmCJwPiNMWlxDE6MZrmNPXRJluWVIYIlAvMplgiM3xARbhi
Xyvr91VSebHQ6HL+iqizLK2Pq4AT6xUU6HY7xMZYIjF+Zl51Ou8LyfBty4mLsKDtJUdUp5o1PdzoU44MsERi/MjQ5hrHpcSzdVup0KH5leV4ZoS5h9libicx8miUC43duzE4nv6SG/VV1TofiF9rbO8p
Cnx2eRJ/ocKfDMT7IEoHxOzeMT8MlsHSrXRVciA0HqimvaeSmCVYWMl2zRGD8Tr+4SC4bksjr28roGOvQnMvrW0uJiQhlxqhzDg5sgpglAuOX5mencfhYPVsOn3A6FJ/W2NLG6oIKZo1NoVd4iNPhGB9
licD4pVljU4gIddlN4/N4a9cRaptaudnKQuYcLBEYvxQbGcaM0f1YkV9OS1v7+XcIUq9vLSUlLpKpmX2dDsX4MEsExm/dmJ3OsVPNfGDzGXfp2Klm3ttTxfzsNEJcNgGNOTtLBMZvfW54EgnR4SzZYuW
hrqzIL6O1XbnRykLmPCwRGL8VHupifnYaa3cesRFJu/Da1lJGpsQyKjXO6VCMj+tWIhCRBBFZKyL73D/7nGW7tk6T0izrtHywiHiwIoUi8rJ7NjNjLtitkzJobmtnWZ4NRNdZYWUdWw+f4OaJdjVgzq+
7VwSPAG+r6jDgbffrrjSoarb7Ma/T8v8GfqWqQ4HjwH3djMcEmTFp8YxKjWPx5hKnQ/Epf99cTIhLbBIfc0G6mwjmA8+5nz9Hx7zDF8Q9T/HVwOl5jC9qf2NOu3VSBvklNeypqHU6FJ/Q2tbOki2lXDU
imaTYCKfDMX6gu4mgn6qeHgayAjhb18VIEckVkQ0icvrLvi9wQlVb3a9LgLNex4rI/e73yK2qslYi5v+Zn51GqEt4dYtdFQC8v7eKqtombs+xqwFzYc6bCETkLRHZ3sVjfufttKOv/9n6+w9U1RzgTuD
XIjLkYgNV1SdVNUdVc5KSki52dxPAEmMiuGpkMku2lNJqfQr4e24JiTHhXDUy2elQjJ84byJQ1RmqOraLx1LgiIikArh/Vp7lPUrdP4uAd4EJwFGgt4icmhNhAFDa7d/IBKVbJg6gsraJ9/dVOx2KMRKT
tOsJNE9IJC7FGgebCdPeTsgxY6H6+EFh65gYi0kdEItzPE4HLgZ3uK4h3gVvPtb8xF+KqEckkRIfz8qZip0Nx1OvbOvoO3JbT3+lQjB/pbiL4GXCtiOwDZrhfIyI5IvKUe5tRQK6I5NHxxf8zVd3pXvc
94NsiUkjHPYOnuxmPCVLhoS5unZTB27sqg3YaS1Xl77nFjO/fm+H9Yp0Ox/iRbiUCVa1W1WtUdZi7hHTMvTxXVb/sfr5OVbNUdbz759Od9i9S1SmqOlRVb1PVpu79OiaYLZjcn9Z25e9B2pQ0v6SG3RW
13DbJbhKbi2NFRBMwMpNimJaZwKJNh2lvD755Cl785BBR4SHMz05zOhTjZywRmIByx5QBFB9r4KPCo06H4lU1DS0syytjfnY6sZFhTodj/IwlAhNQZo1NoU9UGC9tPOx0KF712pYSGlvauWvqAKdDMX7
IEoEJKBGhIdw6KYO1O49QWRscN41VlRc/Ocz4jHjGpsc7HY7xQ5YITMBZMGUAre0aNOMPbTp4nH2Vddw1daDToRg/ZYnABJwh7pvGf/vkMG1BcNP4xU8OERsZyvXjU50OxfgpSwQmIN07fRAlxxt4e9c
Rp0PpUdV1TawuqOCWiRlEhYeefwdjumCJwASkmaP7kRYfybPrDjodSo96ObeY5rZ27rSbxKYbLBGYgBQa4uKe6YNYt7+aPRW1TofjM9ZAaN9VQ0OAt1IcLJTj/MkdI2uKPadRrwbJHCVZiHLIMGr6GIMp8NmR7a6x60HXFzN+nS2cEDHTBSgsBTUc+M40UZqnZdUEM3GOyRB4nXbj5zQpz8yNnqvCVkU9DS
R8pE4RhQ2DTm68qTxZPNZNMTUNZ6p/Wi/rGt8fzgpOJm9wXMU2zlFKw4Q3YNkWDSk5sW2FIwt8RpqovM7JJ+7IhDjbfeb+TWe2mJ5vs2H/MQSMQRT52IM7W/3YqvVP5+Kmkdqm+ubqF8d4Mqf8fAfvZ9NhkNH/2CrcnQPdW+mBWr0UNhxOoxdHKv0lmo1KZzpsJuzOwnRxfxFaMvLHm61n9PDwuKPGOeosHtHsnoSvpGBXeNKXCXYp/WwPjHyzbEGsmAGgXU8dVcFy9Drx8tTLPq+0xZKMBC5sN6LBkMCj5b+R+AcNHpcJRX2ZqUL5VAb3Bk+3M/L4ckwrNdoATyTgmh2MXBYnfDwxcLNVKu6OKYaUIw5ndrXhw7zDpuF7xs7GAl5A3RVxqYNEg3sixvKRRgczvrcsh6t+fsw5W8AgrVjn1+BWmY4M/qmZ5ItnwHtk0wVPFpL0NfUd2d4Fi9WZFM9mVuuqKgxmAdbMLPpC1mKAN9v/PjZYn7e+6FwrDmtAWJWvTGE0KKsIbcGK3AnM9V/hnCZAyVaUu0mq2OaDpT6NbB5AlRKuDnNpn/a6zz7pF6sMlo9lECmYiOMHRaRPNXAyOePw+NgnIcOQdjN8F0kPAZXQIj3TNQfrF4A2D/Fa6UzAtT08+2q8mJ1ldP3Xn3pwpmgkvzo29dQ0PL7pB1eXpb1LwfRtKNx4oK8VSRjr8JytjDkSb7JLV9g2kodqiQbfKjS+KBzv9eSixSDB+aNtSbMgb29hNWt/3MvGM/Uv0Nuc1lfnR3sO3M3OUOCjGYSukaWkMw+n/P6b6WYeqKs5Uzq7bRkFBTbsuVy/aMVNZeQZP8ug37HRqG4Uju8WOQUfqfiiXUk8xNUoDmgIWnR4v9NcHBHDk/a28RFoD9cmyrUcvOwfxzP0Ib/29jFM3yIHrlLCJqbRPQqzTCIE2j3tWr4x7V3l0bKlO1Y6c4AMPwtJ8rKHNQIOrqbBlbaOGydnRUmLO4tGChnIPGBKRUQZSCu+oNp0nmgUT/ETU1IgaAeUbmJnT85Akt/Gd3BhDqDk0+DDiYY0Nwo5TxZWUCxaz1WkuNbSIt6FFn40Vw/N0rJT3d0o8OnNaGpDd6Fb08s2BG3kMiFJsmq4hf7Dpio8Pn37bY8fZrgd7ajkfklkG9gNQpyAVmaHLhyb/Cr52lP2NLiGmUsQzOUPN32xUp3XNrXyVb4I2nTWaxUF2NaOTEg9x2NXGS5C2RAH6PePSUd+NqZpJx91RzelO4nGeSIzQJG2ZafpMbdwOQuRsGcdLe7C3ko8VuzI2JYP1I0ygOCzx/WWQQTWj6Px/qu0SPiqIWITsydV3fI0IV/2zkN0PQjKbyCTg8y7y+DEYOaHfAw5eCy2GRoikP8Bc/aPAXKoP/6IBSc9zv0BlYSeTqU8SeHW3o9QmDHvX5HQn6aLJhUPhrQ0w7ngdShnHPcwhYNDAKhzEvoIM3JsZbb9A3I62J0WOXyDKANpCCJn3bfkHUV+ZQkGvfHIwUvgoF8wGfCbUJ4LYH2Nkh2N4vYH/YVLXXbnDbhwQZmBVqfEwDGmdoskZNb8Qoot5wZgd/2LKbZlp0l7OpohGIl1HvEvyoYVcA5mp9rdjjbQfUs9cyC6B5ScKRqeQI/I3GQiDdz4qixh/F5iYgBovTJy0cmMoQuzhD6uUX5GjNlE5pNSpK8D2gjSYRXpQd8wyExeJ0CQOuZeIjzJBIqNYwbuJIhfAUlUe9DU0j6XXt0DbrSNnXKcmW5/9ZRrfNwfRncs6oHajxNJ5nDD6qnRMNGp5o2r8cITySE/2UaQ/SaNsHvZ6m7Hb7gZDSU1Mxp1zu3rc9BmaDNPO9vdHFlLvhHvR8jcfiKwjdAtQaxZ9YWnbNkLZXrMuaCKAXmQ83E2Ds06oN6+APx6Gzaf+jmBVSvqmI8QjJFG60YChTo0R5sjmOaBxS30R8mIDYLtrdfxEqWu5bZGGCdWvMDRfC3WttXUBjfiR4zqTyxZhI9dzFl4o1BfRhNRD9jiSCcUaMuxWDPY6cfBOvtXRHQMQfXPbW0uqXwYZ6aMo0kCrB0wy3Ch63S4Sb4eaIH2lSP6FF6j9eaanZZsgwINwH7l2tW3TiryOKHo7v1C5UgRyllIAaKLFsE8IqC2oUd7RKbPc1IuH6e5wl2lUS3jwtIjhS5hKbl7owNV2nlmjlQHswvlp5xLvTnbmGeNjUkVXwmDvwchQdrYz4tItbfbUGfx2k0OuR49MffsRdFP/bf5HfIBXYRhwg1bvyGX+J3dJRU1jaIO1q1S44ZgUn/eEXeLjRsgn3L9qC1fY6RQ2Sw0Ly2G7bXO3cfRBmYeFd8bWxqnT1kdAAl5Oy3VBMz1KZhkH+ofPYXsUZ0MT5PWE/E0cEN2m0f8PcJdXONa+k9GKEKLL8wN1B8JlO6o+uKuMX9YNLgYePc37eEm7wLK7BSHKvWGEUAUfjU0ZQBVJAxn0mO0CvW8F2XHV5y6YNNs5Ocm5Ujp0ceFm1VqoshYlOSt4TmNqq2aSWB1edsuSK/GB5KZmsLXTW2Ihw+RnhhFGEORrpFX4HAhLe9RUupR2LvZpYxHLBiO0WjEGa9ixpSe2PoyjVBUj8q3hyxvQ36ECgxC2mUUtbMZ6o8oSYu5wVnsujsDN6V+0FYbpcv0KvtRa7nSoXEV4sXRKv6Di3rqH3wexULmUGhiqSJDFAnlwJaMWD3Uvsd2ChVrq2IQbP1UTm4Atud4E1fZtfX9vTDevlv2b0WQYdhpILCK6tjUyDHpBUvhFUiWxNxYlfi4s5ffYXtX9kI8wgMHAp4zRiFvHn9sv8zE0TOGvLEXnH3Gq4FOhbURqWSSLE0ZF4CHmBqOfCMY04uXf5VDpGrCf7Tpy2FPWDXYXhRvD1GlfxuCCH+X3KopIPwSRANUa7nzrCj7HLNS2KwBQ9XBPZW1PzNNH6Ssk2UBFGpHU1fWHRAl35QsrqumOfwgW0gmc0zqcShYDEBHZFYe6BQKgVOdwXW/KHWUvHCdtk0DGqCAL5UsIQqqAaI3ebRNJkT79k2rADdk1+DSaCmI0T5Fuf6xF09Brqq5GKEOVP1zHvnzyGIzGdvWbnFZi+BxJgC/BQl/ZATdiRFQdddkdmp1b3Rf1n+NJ9BWhGXy+xZbD0lsR9HodjH2OnNy/4DCD7VJBjdG7yvXeJSwnu+qwP2VFOyPxtGfq/KyN29gqcSIUyk5A5ll1CJEStkRUbJ3MDIJ2wnFXrpfx6+7FwVPShQEXGV/e1jKbABUOn/zZaiIRWKGwIRJxnKHZjUuBzwHZpCK+3LWpCEsy8eEsZ+lkCBtCzcHS4hGGuSnY7kN4DUrHY4HmeAT9xomIkIjZjPo4qkJjd7qWe5hg8tyT3QMh29EpzJ1gxAN0WD6s3TX+ueyB+WEZCkgmDW7uc44xwKh0ubYrD9sCnlzbSYvKAJqQxeK2a5eDAJHuF7o+2p3kLXwtGiEehZE+SIW2AjBlfXRwL0I8KtwTQzyVNJ2cXJTJPPjIW5Vyj0rjivhRE0S5KT2lbXtkqGEzqqGQ8+gYwJ3OBkuNNnmRWn9l63JRGlIRPYtRB0YgqJKM7a1jJy4JOVAAU1LVhvUT9AV1yAT6Qc7wYn1tWxt3VhOPCkHhJGqjGaQBcvNzuSJTdjBnGb+fVZ7NT1u4vJHeFAVAOzPnQl2GfhqmzjJa1uhzzYU/3XrL9xoSmBh53+FJKSUQSBhLw55CVqlwvbaxI2NNFrjS4qT1nRXW4eLtrfe4t4PTDhnKZY8ufyKFqxwXuoOT17Q4A9ejP+mG0lQhz6NLhbywt5EcsdJkSdCDzKUDsjFchfnyXfU9Bt2kRtLHiUu8t6ORRG9xbZCNRYDzhSwcV2aL9Me0ZVWzdVwhQAAAABJRU5ErkJggg==
```

```
            "text/plain": [
              "<Figure size 432x288 with 1 Axes>"
            ]
          },
          "metadata": {
            "tags": [],
            "needs_background": "light"
          }
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "9W2VAcLiL9jX"
      },
      "source": [
        "With just a little bit of extra work we can easily plot multiple lines at once, and add a title, legend, and axis labels:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/",
          "height": 312
        },
        "id": "TfCQHJ5AL9jY",
        "outputId": "1ad5975a-29a9-4cec-cdbc-4008aeaad889"
      },
      "source": [
        "y_sin = np.sin(x)\n",
        "y_cos = np.cos(x)\n",
        "\n",
        "# Plot the points using matplotlib\n",
        "plt.plot(x, y_sin)\n",
        "plt.plot(x, y_cos)\n",
        "plt.xlabel('x axis label')\n",
        "plt.ylabel('y axis label')\n",
        "plt.title('Sine and Cosine')\n",
        "plt.legend(['Sine', 'Cosine'])"
      ],
      "execution_count": 100,
      "outputs": [
        {
          "output_type": "execute_result",
          "data": {
            "text/plain": [
              "<matplotlib.legend.Legend at 0x7f78634f2860>"
            ]
          },
          "metadata": {
            "tags": []
          },
          "execution_count": 100
```

```
        },
        {
          "output_type": "display_data",
          "data": {
            "image/png": 
"iVBORw0KGgoAAAANSUhEUgAAAZAAAAAEWCAYAAABIVsEJAAAABHNCSVQICAgIfAhkiAAAAAlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9uMy4yLjIsIGh
0dHA6Ly9tYXRwbG90bGliLm9yZy+WH4yJAAAgAElEQVR4nOydd3hU55X/P0ddQgXUCyCaKOp0424DphmEjXuKneYkm2w2cZZJ9mNvcl6f042m7KJN1mn2Ym7sU3HBveCMYgiVChCNElIQkggVFA/vz/
uyJGxJISm3Lmj+3me+8zMrd8B3Tn3Pee854iqYmNjY2Njc6n4mS3AxsbGxsaa2AbExsbGxmZI2AbExsbGxmZI2AbExsbGxmZI2AbExsbGxmZI2AbExsbGxmZI2AbEZlggIneLyBazdVwMEXXlHRL7k4Wv
+QET+6Mlr2vgGtgGx8RlE5EoR2SYiDSJSLyIfishshsAFV9RlVvMFujs4jIZBF5SUROO77nPhF5UET8h3pOVf1PVfWo0bLxDWwDYuMTiEgksAH4DRANpAD/DrSZqcuViMhE4GOgGmHMhS1SjgVmVmAHGmHNpv
hiW1AbHyFyQCq+pyqdqnqeVXdoqr7AETkfH5oGdnEVER+aqIlIrIWRF5XESk1/YviMh+ETkjIq+LSGp/F3aMCKodI4L3RCSj17YnHefeKCKNIvKxwxD0bF8oIgccx/4WkD4vYvDvwDZVfVBVqxzf96C
q3qWqZx3nWyEixY7v9I6ITOt1rX8RkUqHjoMiMt+x/hEredrxfpzj3+Y+ETnhGOn8sNc5/ETkIREpE6E6EXlRRKIv+r9j45PYBsTGVzgEdInIUyKyREGDeKYG4HZQDZwG7AIQETygBB8ANwNxwPvAcwO
cZzOQBsQDu4FnLth+Ba8aP/yjgMPCo4zqxwCvAvwKxQBlwxQDXWQCs7m+jiEx26PyWQDzOAowGxVjXB812MDXOtKYAowH/hRL0P0j8BK4BogGTgDPD7AeWx8GNuA2PgEqnoO40dPgT8AtSK
yTkQSBjjsMVU9q6ongLeBXMf6rwL/T1X3q2on8J9Abn+jEfX9s6o2qmob8ELjF/EJT1X3A7oEEY4NhN7Lj8DjeZbKhWUjOAV/kwOSfKMYTiKMH5ELwlVfRXj6fx5h1uqCFj
GWoHzQPcla03h98CjPe48EYlzxIxshiG2AbHxFRqBucDHItKMYTiKMH5ELwlVfRXj6fx5h1uqCFjSz+5/xXAbVQIlXILBUtXTGGm4j2EYoDTgwwH2LwPmYYxyikWkAXgZyAcaVfUgcA9GKvNpYDmwXFX
bMeIfjznWV2ME/L8/WK29+DWwDtgiIo0Y33Y33fuEM5j4wOI3VDKxsbGxmYo2CMQGxsbG5shYRsQGxsbG5shYRsQGxsbG5shYRsQGxsbG5shMaxmj8bGxuq4cePMlmFjY2NjKXbt2nVaVeMuXD+sDMi4cePYuWPS4XMlmFjY2NjMdD+sDMi4ceP
Iz883W4aNjY2NpRCR432tt11YNjY2NjZDwjYgNjY2NjZDwjYgNjY2NjZDwjYgNjY2NjZDwlQDIiJ/FpFTIlLUz3YRkf8RkcMisk9eZVTadp+jHWmpiNznOdU2NjY2NmD+CORJYPEAr25PA4ePKB9ep+E5Te25d
gVChNAx4Afgfg6MH8MEYV0DnAw4NsYWpjY2Nj4yJMnQeiqu+JyLgBdskD/uro1LZdREaKSBJwLbBVVesBRGQrhiEaqG/1Cl4HpprISYNYtNgZCr4W2cKTVe3srf8LLWNrZw738m51g6SR4Yye1w0cRH
BZsuz8QUaa6DuMDRWQWM1BARD8gxIzDTeW4izLe1sK6ujqa2T7m6lW2Fi3Ahmpo4iwN/sZ27vwtt/BVP4dIvPCse6/tZ/BhF5AGP0wtixY4emovhVOPTa3z8HR8Gs+2Hu1yByoA6j5nKoppGXd1ewZk8
lNefa+txnQuwIFqQn8NVrJhI9IsjDCm0sTXcXlG6F/D9D6RaMJocX4BcIKTPhym/B5MUgg2nU6Hma2jp5ZXcFrxVV8/HRerJxGVZ8AngCYNWvW0W0Jqf3PU
CtNTD6VKoKzVumm2/gY/+F7JvgwWPQHi860Q7ydmWdh5ZV8yavSfx9xXcFxOnrRzHD5ZV8yavSfx9xOunRzHD5elMDFuBJEhgUSEBHD0dDM7j9Wz/Ug9f3z/CM99fIKVvXjuRL1wxntAgf7O/go23U/oGbPg2NJyA8AS46jsw7kqISIK
IBGhrgpN7jKX4FXjuDkieDtf9ECYt8BpDoqq8XlzNI+tKqD7XysS4EXzl6gkS8gLjwYfz9D576Ks7x14BRvH6xlw74qrpwUyyMr0pkUH2HyNzAX0xtKOVxYG1T1M72rReT/gHdU9TnH54MY7qtrgWt
V9St97dcfs2bNUpeVMqk/Ctv/F3F3Y9BaGj4JY/GTeQyWwtqeEHrxZyprmdr107kfsuH0HyNzAX0xtKOVxYG1T1M72rReT/gHdU9TnH54MY7qtrgWt
M+Bws/bnprq3Ks+f5tzVFvHXgFNOSIvmPlRnMTI0e8JjOrm6e3XGCn79+kJb2Lr541Xj+kJb2Lr541Xj++YYpBPq4a0tEdqnqrM+s93oDrAUI2D+P6o6xxFE3wX0ZGXtBmb2xET6w6UGpIfqInjpPqg/YjxdXfk
g+Hn+j6m7W/nxhhKe3HaMqYkR/PdtOWQkR13SOXOYcrefbL+zldFMbv7gtl2XZ3uueszGB2kPwj1w+iBc9nWY/yMIDBn88Z3t8t8O5j8P5/w+jZcNvfTHMBF59s4L4/76SlvZMHF07m9ZTWhFd+sxHXVJ843RTGz9
77QAv5ldw7ZQ4Hr9rBiOCfdeh45UGRESewxhNxAI1GJlVgQCq+nsxnIy/xQiQtwCfV9V8V8x7FfAH7gONWjqvqXi13PLQYEoK0R1n8LilZD9u2w8vceNSJd3cpDL+/+/jpV0uqXo8LoU5T+uOAY2sLCv3FXK6nFsa4+1dnM+6psw81XdnMpjj5fFrvBxUfzpqb/Kh0
EOTVDyaBnbdEcw8ljhTXU+ymbRO4xZsW/SWVdzhWcaXsnbSX/sv5fPBDbvOKuA1/Zx/HwFFWuT1Jb0BUSCDmKPrjxp8T3pj4lgAEEkrFsnN54VwB3mGSuTiz0UMSg1X//R//nXeljUtwtI+LjdY9sbnhdzmVEHvDyWU0
mGdiQV3r4RyOpkWhbwz4Q00D+/SEkJaY2gq0UNqhsIXpCqaSZUyIGmgfD5SzNaaa4Tnh2Ij3bM6JoPtqEnPmOoFcMctaD4eGzTdGe1Ge0tPmdmsEg73+Of3rFWTz4j8YA/5dZJTkUyijmjDyMtXcdh/K
xO4N64GrR7OLRTftJiw/n7stSPSBwcIgID69I59z5Dn6x9ZDZclyKbUA8zawvQOwUeOOPfoatzwF1b07p4bPMBpiZGcMvMMR4SOHi+u2gqESEB/GhtkR1Qtyr71OP5drj+h0aszpsYEWM0azv+AZSsuej
uf/nwGMfrWvjR8nSv688xNTGSey9L5ZmPj7O/6pzZcl.yGd/0rDwf8A4yboq4U9vx1wF3/8uExKs6c51+XpX/SGc2biB4RxHcWTubjo/W8deCU2XJsLpXOdnjjYYibZriLvJHp9xr63np0wAeuU42t/Ob
NUhZMi/fYhMFL5dsLJxMVGsgj64p95oHLNiBmMGUJjLkM3nms3yyT001tPP72YeZPjefKNPMD5/1xx5yxjI0O4+dbDtHdRx9pGy8m/89GI7SFPzYebLwRP3+Y/2/GA1fBs/3u9suth2jv6uaHy9I9KO7
SGBkWxLcdD1zvHqo1W45LsA2IGYgYN21TjdFXvQ9+82YprR1d/GDZNA+LuzQC/f349sI09ledY1NRldlybAbL+bNGRuD4ayBtodlqBmbKUqOD4TuPGS11L6C8voWX8iu4a85Yxse0MEHg4Llj9lhSRob
yi62HfGIUYqoBEZHFInJQRA6LyEN9bP+1iOx1LIdE5Gyvbv29tq3rHIXMHYuTL3RmKHefPpTm6obWnluRzm3zBzNxDgv80v3wYqcFCYnhPOLLYd8MtfdJ/not3C+Hm74iefKlQwVEZj/MJyrhJ1//Mz
m37xVip+f8A/XTTJB3KURFODHN+dPYl9FA2/st77b1zQDIiL+wOPAEiAduFNEPjX+VNVvq2ququCvwFe6bX5fM82VV3hMeGuZP7D0NEMH/zyU6t//24Z3ap83QI3BBhpvd+5YQpHTjfzfzyh67TpbXc/4
sfPx/xrykpByz1QyO8VfBxPnw/i+MSY8Ojp1u5uXdldw9d6xpM84vlZtnjCY1JoxfbLW+29fMEcgc4LCqHlVduB5IG+A+E0W8v3zSTKfmXCvP7jjBzTNSGBNtndGqzt7MQNSGBNtndaYN6QnkDM6il+
/UUpbp+t7Oti4kB1PQNs5uPp7Ziu5NOb/yBg1bf/dJ6v+561SAv2Fr13r2oZq7iTQ349/mm+4fV8rrjZbjlOYaUBSgPJenysc6z6DiKQC44G3eq0OEZF8EdkuIiv7u4iIPOA4Lr+2ttYVum3g2/qBEfgVeDD
F798to6tb+cZ1bmrg4yZEjFFI5dnzvLzLzL2HoV4LW2N8NHjMHkJJGWtbEc2v2VHLuZanER1hj9NFDXm4KE+NG8Muth+iy8CjEKkH0O4DVqtr70TbV0eT9LuBXItLnI6Gq/p+qzlLVWXFxcZ7Qal7cdWL4k6WI15Xc6q+Ztanh6l4vJUr4NcKXWu6oEoYdbNHX7OWMRL3SwLBQ2l6lYaxTkTnqosLOdbYZIkSOp4tNgJnydCTbDGqj
Xb22Sp/NdlyWtbLelTNm3DzgTLVtalMR1jL0pjcFnWJ2cHczpcxoPkS6eetYKb2OQgV/3IMo0K92s9eAlX/2xWcKWU1z/nmSI8y6d/SuCdSi+4DZxIYbMCQ6/N3JqBRtJyvvBMIDXuTWK0AxoJGkWJtG4m
1VlycF6b3xU8zYiE7/o/a2lqe/fgEN09PYWyMdUYFPyVFkv26CieeK/M0jeFT7PjD9B6Fq75rtlKhsZVDxql3nc9xW/eOkxwgD9fcXE7Z0/g7yd8a8FkSk818VqRdUchZhqQSqD37LjRRnV9CqcXuK9
UtdLxegR4B5jueoke4up/htYG9q35bzq7lW9cb43Yx4WICF+7ZiLH6losfVP4LO3NRvB80gKj6q0VGTMHUq+k88PfsLngBPdcNtZrSpZcKkuzkhgXE8YT75SUiU4F
RwEe91o0SkWDH+1jgCsC6TYiTp9Mx/npyK57hlqxoUmO8OxVxIG7ISGR87Ah+/651bwqfZdeT0FJnvdjHhVz1bQKaTrLS732+cOV4s9UMGX8/4UtXTaCgooHtR+rNljMkTDMgqtoJfAN4HdgPvKiqxSL
yYxHpnVV1B/C8qmqvddOAfBEpAN4GHlXV4I0EnYbYM/z1h4H/S8ZMZ5LjXXzUMn8M/dGtRtnk6cR2zyp5kWRHE5wDjI8t0iZirmnZt6EdfStg9s/1Tre2xJr9M00eCs3iaVVaf2FZvj3U7+A15UDDs+cQqBXJTNVdDXUKNgBqsdHDGo7WTOZtRbbS4F3wJ4mKrhH3KDpAHBBaZoqsUpgsmNTpSoQ6GJgRhpMt/j7wL8oWsvlnzqZqTB1XzE6uPkTTvrRtOebcsV8fXgxJSM9MtRHKxIMyZRuidvTXQ8mxq6eJLm5YkeWl5P3dZDVnJdHt8E1dNSXdA/CvN8HcVfdmnWAyq4nt6Y2gbVF3vTDo7ChgFnRtmc7CrbSPNkOcjWgtEWd5RmFFDN2lPF0/7rdZutTWfebBWnp/7yrmYl9TKhVpaB2wQt8R4kFzGANJ35qh/L1kHF5VcNZYK8nk7rXfw8JEI5IkHh1vWUk/79Uf3xZS/Ck6a25ecnqffp8SQB+p8urE+4pb/L4PGdJtLvsWN/nNH8YsHXyj8h0k8Gw4/D8HHQxTK/hCLC6UW/sk5LWKbVx0kAWeH3qFh/mAngqHE1WuyEJdRHDJ5BmfBf5NT1P5pLKqvHQEThz+m7tiShRVI/4GbwdyMDeMr8Kv3MOPnc8mMoV4A5EipCrQ6vi3FnVT59HPRcJtR/i9Yx7pzvLdB/vLWnrqkGwCO+JzktYuWPxxpyWcnMHIPymyY1m3VZ2JGTnbQF4sCNNYqTRmEzrhHmTyL4AgDtHBgP/36hjfKF6ZH1SFk0U3S4pfbM7zq3ZLhdzDgD9Bak5w9Kg2Ofu6rxRXzIuyIcGc5IOvPZBvg8Zxuz5MQNqeSBxbOt7HMAQuIPLhPpcCLkF7QvWeUJCuAQDPm7B2GpPFJ7R6GpL8LUQJJIzdaxS8HhFZWFjyWu+zMNMZN8ROgAXV5Y7VVSB6TTaqvzGHWkZp8/SzLTZeiw5y/PZ8jlMXmQZuKBxKqAH0kf9bGa83Osa6ad6gRsIZ7AgFHsObuQXfy0xkVF2v7NyQqK
RwEe91o0SkWDH+1jgCsC6TYiTp9Mx/npyK57hlqxoUmO8OxVxIG7ISGR87Ah+/651bwqfZdeT0FJnvdjHhVz1bQKaTrLS732+cOV4s9UMGX8/4UtXTaCgooHtR+rNljMkTDMgqtoJfAN4HdgPvKiqxSL
yYxHpnVV1B/C8fvrXaBqQLyIFwNvAY6pqXQMCrI+6ixg5x7djPrr4zl6Mv5/wwNUTKKxsYFtZndlybHro6jTmHI27ykghtzANSVdTouN4MGwzSRFBZstxiltmjiZmRBBBBPvGfNplOmxkBUdZOqTlbViar
```

6qGPdj1R1Xa99HlHVhy44bpuqZqlqjuP1T57W7kraO7v5ackoDgRlklj854vWyPJ2bp6RQnxEML97x5o3hU+yfy2cq4B5XzdbidM8veMEj3esIL693KjlZWFCAv257/JxvH2w1pJFSa0SRPdpNuw7Sc2
5NjrnfA0aTjjVic0bCA7w5wtXjueDw6cprPDNTmyWQtXIvIqeCGmLzFbjFG2dXTy57RiNE5YY32fb/5gtyWnuvSyV0EB/nnjviNlSLhnbgJiMqvKH94+SFh9OxnV3wMixn8pztyp3zx1LRHAAf/zAeje
Fz1G+Ayp3wWVfAz9r3/Jr95yktrGNB66ZDHO/anyvinyzZTnFqBFB3D57DOsKKqlq+GypFm/G2n9NPsC2sjr2V53jS1eNR/wDYM5X4MQ2OLnXbGlOERESyKqZo9lUWMWpxlaz5QxvPvothIyE3LvMVuI
U3d3KH94/QnpSJFdMioHcOyEowphVb3G+eOV4urqVJ7cdM1vKJWEbEJP5w/tHiA0PIi/XMYdyxr0QFG5MlrI4n5uXSkeX8uzHvtfK0zKcOWY0L5t5PwRZN7sP4MOy05SeajIetkSMZlPT74HiV6HR2mn
jY6LDuCE9kRd3ltPaYZ1KDrYBMZEjtU28c7CWey5LJSTQ31gZEgW5d0PhamisMVegk0yIC+eayXE88/EJ2jvtIoum8PETIH4w5wGzlTjN3z46TvSIIJZlJ/195ZwvQ3enUZre4nxuXipnWjrYsM86Va1
tA2Iiz3x8ggA/4a65Yz+9Ye5XfOamuP/ycdQ2tlm+5o8laW+GPX+D9JUQ1WeVIMtQefY8b+yv4fbZYwgO8P/7hpiJMHmRca90tpkn0AXMmxjDpPhw/vrRMbOlDBrbgJjE+fYuVu+qYFFm4mfr+HxyU/z
J6BpnYa6ZHMe4mDCesphv1ycoXG0UTZzzZbOVOM2zHx8HjOSMzzD3K9Bca7iyLIyI8Ll5qeyraGBv+dmLH+AF2AbEJNbvO0nD+Q7uvSy17x1mf8m4KQ5s8KwwF+PnJ9w7bxy7jp+hqNJO6fUYqsYDSHw
6jLH2xMG2zi6e31HO9VMTGD2qjxpxE66D2Ck+ETe8aXoKI4L8LTMKsQ2ISTyz/Thp8eHMHR/d9w4TrzdSenf9xbPC3MCts0YTFuRvuQwTS3NyN1QVwKwveH/DqIuwubCauuZ27p3Xz8OWiPHAdXKPsVi
YnuzFDQVV1DV5v0vONiAmsK/iLAUVDdxzWaqRTdIXfv4w4z44+h6cPuxZgS4mMiSQldNT2OAYddl4gJ1/hsARkH272Uqc5m/bjzMuJoyrJsX2v1P2bRAQCrue8pwwN3HvZam0d3Xz/M7yi+9sMrYBMYG
ntx8nNNCfm2ZcJLA5/V7wC/CJUcids8fS2tHN2r12rxC3c/4MFL0MWbdASKTZapyi+GQDu46f4Z7LUvHzG2AkFToSMm+GwpegrclzAt1AWkIE8ybE8NyOE17fsdA2IB6moaWDdQUnWTk9mciQwIF3jki
AKUth77PQYe3JeFmjo8hIjuS5HeV2lV53U/ACdJ6H2V80W4nTvLCznKAAP26ZOfriO8+8H9qbDONpce6YM4aKM+e9viCpbUA8zJq9lbR2dHP33H78uRcy6/NGG0+LF40DuGPOWPZXnWOfXR/LfagaKa0
pM63T77wfWju6eHVPJYszEhkZNoiqu6NnQ9w0o2y9xVmUkUhUaCAv5Hu3G8s2IB7mhZ3lZKZEkpkSNbgDxl8Lo8b5hBsrLzeZ0EB/nt9pz0x3Gye2w+mDMPPzZitxmteKqmls7eSO2WMuvjMYwfSZ9zs
SCPa5VZu7CQn056bpKbxeVM2ZZu9N5bcNiAcpqmygpOoct80a5A0BRvG7mffD8Q+h9pDbtHmCyJBAlmUnsW7vSZrbrF2y3mvZ87RRCifjJrOVOM3zO08wNjqMyybEDP6g7NsgIAR2Wz+YfvvsMbR3dfP
qHu+NG9oGxIO8lG/4c/NyLnFWcM5dIP6w92n3CPMgd84ZQ3N7F+sLTpotxfdoazIm02WshOBws9U4xfG6ZrYfqee2WaMHDp5fSFi0MfN+34vGTHwLMy0pkpwxI3lhp/fGDU01ICKyWEQOishhEXmoj+3
3i0itiOx1LF/qte0+ESl1LPd5Vvml09rRxZq9J1mckUhU2EWC5xcSkQBpNxjBUYs3m5oxdhRp8eE8Z4EURctRsgY6mo3sPYvzYn45fgK3zLyE0XoPM+8zZuD7Qtxw9hgO1jR67cx00wyIiPgDjwNLgHT
gThFJ72PXF1Q117H80XFsNPAwMBeYAzwsIqM8JH1IbCmpoeF8B7cP1p97IdPvhqZqKHvLtcI8jIhw++wxFJSf5VCN9TqweTV7noaYSZafed7Z1c1L+RVcOyWexKiQix9wIWPnGXHDvc+4XJunWZ6TTFi
QPy946QOXmSOQOcBhVT2iqu3A80DeII9dBGxV1XpVPQNsBRa7SadLeCm/nJSRocy7FH9ub9IWQViMT7ixVk5PIcBPeHlXhdlSfIfTh+HER0Z5c4vPPH/3UC2nGtuG/rAlYlS0PvoenLV2wkZ4cAA3Zie
xvuAkLe3e530w04CkAL3NaoVj3YWsEpF9IrJaRHr+ogZ7LCLygIjki0h+bW2tK3RfMhVnWvjg8GluvVR/bm8CgiDrNji4GVrqXSvQw8SGB3PtlDhe3VNJZ5dd5t0l7H3GiJPl3Gm2EqdZvauC2PAgrp8
aP/ST5NxhvBa84BpRJnLLTCNu+LoXVrT29iD6emCcqmZjjDIuObVCVZ9Q1VmqOisuLs7lAgfDy7uMLIpBTYYaiOl3Q1e7UWXV4qyaMZpTjW18cPi02VKsT3cXFDwHkxZARKLZapzibEs7b+4/xYqcFAL
9nfh5GjkWxl9tGFYvDUAPllmpoxgTHfrJ74g3YaYBqQR6j1FHO9Z9gqrWqWpPRbE/AjMHe6y3oKq8sqeCeRNi+q4keikkZkFitk+4sa6fFk9UaCAv7/bK/zZrUfYWNFYZ7iuLs2FfFe1d3dx8sTI/gyH
nLjhz1JgbY2H8/ISbpo/mw7LTXtcz3UwDshNIE5HxIhIE3AGs672DiPRqPcYKYL/j/evADSIyyhE8v8GxzuvYfeIsx+tauGm6ixr6TL/HqLJaXeSa85lEcIA/ebnJbCmu5lyrXWDRKQqeh9BRMNmrw4C
D4pXdFUxJiCAj2QU1vNJXGHNifCCYvmpGCqqwZo93pb+bZkBUtRP4BsYP/37gRVUtFpEfi8gKx27fFJFiESkAvgnc7zi2HvgJhhHaCfzYsc7reGV3BSGBfizJSrr4zoMh61bwC4R9z7vmfCayasZo2jq
72WihFp5eR1sjHNgIGTcbcTILc/R0M7tPnOXmGSn9V6m+FIJGGHNCitdAe4vz5zOR1JgRzEodxcu7K7xqToipMRBV3aSqk1V1oqo+6lj3I1Vd53j/fVXNUNUcVb1OVQ/0OvbPqjrJsXhlnY+2zi427Kt
iUUYi4cEBrjlpWLQxJ6RwtUhlsUcAACAASURBVOH7tjDZo6OYFB/Oajsba+jsX28UTvSBsu2v7q7AT4wsPZeRexe0N1q+MRvAzTNGc/hUE4Ve1JjN24PolubtA7U0nO9wnfuqh+zbDJ/3sfdde14PIyK
smjGaXcfPcPS0tWcNm0bB8zBqPIyZY7YSp+juVl7ZU8kVk2JJiBzC3I/+GDsPosbCPutnYy3LTiIowI9XvChuaBsQN/Lqngpiw4O5cqBGOENh8mIIjjTKNVicldOTEcHuEzIUGiqNuQ7Zt1t+7sfOY/V
UnDnPqhlOZipeiJ+f0Rel7G1oMieN31VEhQayMD2BtXsrae/0jvR324C4ibMt7bx14BR5uckEOJOO2BeBIUaAsGSd5X27SVGhzB0fzdq9J73Kt2sJilYDaoxILc4ruysZEeTPDRkJrj959u2gXVD8iuv
P7WFWzUjhTEsH7x7yDmNoGxA3sWFfFR1d6nr3VQ/Ztxu+3UOb3XN+D7IyN4Wjp5vtPiGXSsELRg+MmIlmK3GK1o4uNhVVsTgzibAgF8UKexM/1UiB9wE31lVpcYwKC2SdlxQjtQ2Im1izp5LJCeGuSUf
si9QrITLFJ9xYS7KSCPL3Y+1e77gpLEF1IZwq9ong+TsHa2ls7SQvN9l9F8m6DSp3QV2Z+67hAQL9/ViWncTWkmqavKAlgm1A3EB5fQv5x8+Ql+uidMS+6PHtHn4Dmq09mzsqNJDrpsaxft9Jury8B7T
XsO8F8Asw0nctzrqCSmLDg7h84hDrxA2GrFsAMXqmW5y83BRaO7rZWmJ+aRPbgLiB9fuMJ+kVOW58ogLj6bO70+gBYXFW5qZQ29jGtjJrG0OP0N0NRa8YpUtGuPFH1wM0tnbwxv5T3JjthlhhbyKTYfx
VhuG1eKxt5thRpIwM9YoRu21A3MC6vSeZMXYkY6KdLF1yMRIyID7DJ9xY102NJyI4wOtm2nol5dvhXCVkrjJbidO8XlxDe2c3K9zpvuoh6zaoPwKVu91/LTfi5yesyE3m/dLT1DW1XfwAd2ox9eo+yKG
aRg5UN7p/9NFD1iqo2GH5stUhgf4szkzk9eJqWjusPUHS7RSuhoBQmLLUbCVOs3ZvJWOiQ5k+ZqT7L5a+AvyDodD6D1x5ucl0dSubCs2t4mAbEBezbu9J/ASWZXvIgPQ8hRZZP0Vx5fQUmto6eWN/jdl
SvJeuDqPz4JTFlm9bW9vYxoeHT5OX48ZYYW9ComDyDYbL1+JVHKYmRjIlIYI1JruxbAPiQlSVdQUnuWJSLHERwZ656KhxkDLLMSfA2lw2IYa4iGA2FNi1sfrlyLvQUgeZt5itxGk2FVbRrbg3++pCMld
BUw0c/9Bz13QTK3KT2XX8DOX15s0Fsw2IC9lbfpYT9S0s95T7qofMVUZaZ+0hz17Xxfj7Ccuyknjr4Cka7Qq9fVP0MgRHQdpCs5U4zdq9lUxNjCAtIcJzF01bBIEjjH9Hi9PjJjdzTohtQFzIuoKTBAX
4sTjTw019Mm4CxCduiuU5SbR3drO1xHZjfYaOVqMo4LTlEOChEa6bKK9vYfeJs54JnvcmKAymLoWStYY70MKMiQ5j+tiRbDCxmrVtQFxEV7eyYV8V102JIzIk0LMXj0yCcVcaBsTiKYrTxxgpimbeFF5
L6RZoO2ckTlicjY7g73JPxQp7k7kKzp+BI+94/tou5sbsZPZXnaOstsmU69sGxEXsOFpPbWOb591XPWTeDHWlhivLwvj5Ccuyk3jvUC1nW9rNluNdFK2GEXEw7mqzlTjNhn0nyRnjgVT3vph4vRFQ94E
R+7KsJEQwLW5oqgERkcUiclBEDovIQ31sf1BESkRkn4i8KSKpvbZ1ichex7LuwmM9zYZ9JwkN9Of6qfHmCJiWZ8xM9oFg+vLsZDq7ldeLzZ9p6zW0NcGhLUaDJH831IvyIMdON1NUeY7l2S5qsnapBAQ
bbsD9Gwy3oIVJjAphdmo0G/aZEwcxzYCIiD/wOLAESAfuFJH0C3bbA8xS1WxgNfCzXtvoq2quY1mBiXR2dfNaUTXXT4t3TzG4wTAiBiZcZ6TzWtyNlZkSSWpMGOvtbKy/c+g1o3FUpvVLl/T82C11VZf
OoZC5yihGenireRpcxI05SZSeauJgdaPHr92vARGR34jI//S3uODac4DDqnpEVduB54G83juo6tuq2pOjth1wcbMA17D9SD11ze3mPVH1kHkzNJQbReMsjIiwPDuZbWWnOW3yTFuvofhVCE+EMZeZrcR
pNuyrYlbqKJJHhponYtzVEBbrE26sJZlJ+AmmjEIGGoHkA7sGWJwlBSjv9bnCsa4/vgj0rl0eIiL5IrJdRFb2d5CIPODYL7+21j019DcWnmREkD/XTjHJfdXDlKVGv3QfqI11Y04S3QqbTZ5p6xW0noP
SrZCx0iiiaWEOnzIqNdxo9sOWfwCk58HB16Dd2t0w4yKCmTcxhg37qjzeU6ffv0ZVfar3Arx0wWePISL3ALOA/+q1OlVVZwF3Ab8SkT6bIqjqE6o6S1VnxcXFu9ukVPuvxbR1c3m4uqWZCeQEigv8vPf0mEjoR
J86F4jVFwz8JMSYggLT6c9XY2luG+6mrzicq76wuqEDHZfdVDxk2GW7B0i9lKnObG7GSOnm6m+OQ5j173oo8zIjJPREqAA47POSLyvy64diUwptfn0Y51F15/AfBDYIWqfuLPUNVKx+sR4B1gugs0XTL
byuo429LBjWakI/ZFxk1wrgIq88lW4hQiRjbWzmP1nDpn7UCn0xS9YvR+GT3bbCVOoaps2HeSueOjiXdl3/Ohkno5jIj3iRH74oxEAvzkk0rgnmIw4+FfAYuAOgBVLQBckUe4E0gTkfEiEgTcAXwqm0p
EpgP/h2E8TvVaP0pEgh3vY4ErgBIXaLpkNhScJCI4gKsnu7jv+VCZsgT8g3zipliWlYQqbC4axtlY589C2ZtG9pXF3VcHaxopq232XJ24i+HnbxRYPLTF8m6sUSOCuGJSLJsKPevGGtRfpKqWX7DK6Up
kqtoJfAN4HdgPvKiqxSLyYxHpyar6UmeqscNSHtnN68XV7MwI4HgAJPdVz2ERMGkhT7hxkpLiGByQvgnk86GJQc3QVe7o9qAtdm4rwo/gSWertQwED1urEOvm63EaZZlJVF
ef57CSs+1hh6MASkXkcsBFZFAEflnljB98p1HVTao6WVUnquqjjnU/UtN4/UtV1jvcLVDXhwnRdVd2mqlmqmuN4/ZMr9FwqHx4+zbnWTvMDgheScRM0njTKvFucpVnD3I1V/CpEjYHRs8xW4hSqysbCKi6bEEN
suBeVYRk7D8ITfGLEfkNGAgF+4tEHrsEYkK8CX8fIkDoJ5Do+D3s2FlYRERLAlZNcH5x3iimLjb4HPnBTDGs31vkzUPa2kS3kiXLnbuRgTSNHapu9I3jeGz9/49+3dIsxWdPCjAwL4nIPu7EuakBU9bS
q3u0YCcSp6j2qWucJcd5Me2c3W4qrWZieQFCAl/mmgyOMaq0la203lpU5sAm6O3wi+2qTw33l8UKjgyF9JXS2Qqn13Vg3etiNNZgsrAkisl5EakXklIisFZEJnhDnzXxYZrivlnnbE1UPGTdBY5XR/tT
iLMtKHp5urJI1EDUWUmaYrcQpetxXc8d7mfuqh7GXGZM0fWDE7mk31mAenZ8FXgSSgGTgJeA5d4qyApv2VRERHMCVaV6SfXUhkxcZbqyStWYrcZpl2YnDz411/qzDfbXC8u6rQzVNlNU2s9TbYoU9fOL
G2gptni8H4ko87cYajAEJU9W/qWqnY3ka8IIkbvPo6OpmS0kNC9O9KPvqQoIjYNICKFlneTfWpPgIpiREsHE4TSo8uNlwX6X3W2TBMmwsdLivMrzQfdVDRo8by/qTCpdlJVJef56iSvdPKhyoFla0iEQ
Dm0XkIREZJyKpIvI9YJPblXkxHx4+TcP5Du8LCF5IxkojG8vikwrBkY11fBi5sUrWQORon8i+2lRYxZzx0Z5r8zwUxsx1ZGOtMVuJ09yQbkwq3FDo/kmFA41AdmHUw7oN+ArGfIt3gK8Bt7tdmRezubC
aiOAArvKWyYP9MXmRY1Kh9W+KHjfWa8OhxHtrA5S95RPZV4dqmjh8qsl7Y4U9+PnDtBWGG8sHJhV6yo01UC2s8ao6wfF64TJsg+gdXd28XmLUvvJa91UPIVEwcb4RB7F4ifdJ8UZtrE3DIRvr4GvG5MH
0vIvv6+VsKjRqXy3yxuyrC0nPc9TGsn6J9x43lrtrYw0q/1REMkXkNhH5XM/iVlVezEeO2ldeNZt2INLzHLWxrF3iHWBJVtInnR99mpI1EJFs+dpXAJuLqpgzLpr4CAuEVMvN0q8l1h/xL4wPRF/P3H
7A9dg0ngfBn7jWK7DaOpkagMnM9lcVEV4cABXT/ayyYP9MWWJUeLdB26KpVmJdCu+3amw9RwcftMw/BavfXX4VCOHapq8P1bYg5+/0anw0BZob7n4/l5M9Igg5k2IcbsbazB/obcA84FqVf08kANEuU2
RF9PZ1c3rxTXMnxZvfun2wRI6EiZeB8XWd2NNSYhgQtwINhf5sBvr0OtG6XYfcF9tLjQMvVdOHuyPjJXQ0WwUsLQ4S7ISOVbXwgE3diocjAE5r6rdQKeIRAKn+HQZ9mHDjqP11De3syTTIk9UPaTnQcM
JOLnbbCVOISIszUzio7I66ny1U2HJGkfnwblmK3GaTUXVzEodRYI3lG4fLKlXQmi0TySeLMpIxE/c25RtMAYkX0RGAn/AyMzaDXzkNkVezKaiKkID/bnGKu6rHqYsBb8AY06IxVnicGNtKakxW4rraWu

```
Cw28Ykwct7r46erqZ/VXnWGIV91UP/gEw7UajiVeHtVPGY8ODmTs+hk1unIA7mFpY/6CqZ1X198BC4D6HK2tY0dWtvFZUw/VT4wkNsoj7qoewaBh/jU9kY6UnRZIaE+ab2VilW4zJbL7gvnK4GS3lvuo
hfSW0N/mEG2tpViKHTzVRWuMeN9ZAEwlnXLgA0UCA4/2wIv9YPaeb2liSZcEbAowfpTNHobrQbCVOISIszUpiW1kdZ5rbzZbjWkrWwog4o8S4xdlUWEXumJGkjAw1W8qlM/5qCBnpEyP2RRmJiMCmQve
MQgYagfz3AMvP3aLGi9lcVE1wgB/XTYk3W8rQmHojiL9P1MZamplEV7eydb8PubHaW4wRyLTlRjaQhTlR10JR5TmWWvVhyz/QuF8OboZOa8fa4iNDmJ0a7bbEk4EmEl43wHK9Ky4uIotF5KCIHBaRh/r
YHiwiLzi2fywi43pt+75j/UERWeQKPf3R3a1sLqri2ilxjAgOcOel3MeIGBh3pRGktbgbKzMlktGjQt0aHPQ4ZW9CR4tPua8sl2zSm/Q8aGuAI++arcRplmQlcqC6kbJa1/c7MS1SJyL+wOPAEiAduFN
E0i/Y7YvAGVWdBPwS+Knj2HSMHuoZwGLgfx3ncwt7ys9Qc67NOvns/ZGeB3WH4ZRLGkqaRo8b6wNHTTKfoGStkf2TeqXZSpxmU1E1mSmRjIkOM1vK0JlwDQRH+sSIfVlWEv/v5iy31CIzM9VjDnBYVY+
oajvwPHDh41ce8JTj/WpgvoiIY/3zqtqmqkeBw47zuYVNhdUE+ftx/VSLuq96mLYcEJ+4KZZkJtLRpbzpC26sjlajfMm0G40sIAtTcaaFgvKz1n/YCgg2JuEe2ABd1n5IiY8M4c45Y4kMCXT5uc00ICl
Aea/PFY51fe6jqp1AAxAzyGMBEJEHRCRfRPJra2uHJPR8RxcL0uOJcMN/gEcJj4fUK3zCgOSOGUlyVIjbgoMe5cjb0N7oE+6r1xwpo5Z2X/WQngetZ+Hoe2Yr8VoGU8rkChEZ4Xh/j4j8QkRS3S/NNaj
qE6o6S1VnxcUNbf7Gf96UxeN3+UjiWfoKqN0PtQfNVuIUIsLizCTeK62lqa3TbDnOUbLWKHw57mqzlTjN5qJqpiVFMj52hNlSnGfi9RAUDvutn43lLgYzAvkd0CIiOcB3gDLgry64diWfntE+2rGuz31
EJACjhErdII91KWLxstqfMG258eoDo5ClWYm0d3bz1oFTZksZOp3tRu/zqTdCQJDZapyiuqGVXcfPsNSKcz/6IjDUaImwfwN0WfwhxU0MxoB0qlGNKw/4rao+DkS44No7gTQRGS8iQRhB8QtN/TrgPsf
7W4C3HFrWAXc4srTGA2nADhdo8n0ik40yGT6Q4z5j7CjiI4KtnY115B0j28cn3FeO7Curxz96k54HLafhxDazlXglgzEgjSLyfeAeYKOI+AFOBwMcMY1vAK8D+4EXVbVYRH4sIj3Vfv8ExIjIYeBB4CH
HscUYfdpLgNeAr6tql7Oahg3pK6GmEOrKzFbiFH5+wuLMRN4+eIqWdos+IZasNbJ9JlxrthKn2VxUzeSEcCbFh5stxXVMWgiBYT5RG8sdDMaA3A60AV9U1WoMd9F/ueLiqrpJVSer6kRVfdSx7kequs7
xvlVVb1XVSao6R1WP9Dr2UcdxU1R1syv0DBt8yI21JDOJ1o5u3jk4tAQJU+nqMLJ8piwxsn4sTG1jGzuO1ftG8Lw3QWGQthD2r4du+xn1QgZTC6taVX+hqu87Pp9QVVfEQGzMYuQYSJnlEz1C5oyPJjY
8yJq1sY6+Z2T5+ID76vXialSxfvpuX6TnQfMpOLHdbCVex0C1sD5wvDaKyLleS6OIuLdPoo37Sc+DqgKoP2q2Eqfw9xMWZSTy1oFTtHZY7AmxZK2R5TPRJYUdTGVzURUT4kYwOcGH3Fc9pC2CgBCfGLG
7moFKmVzpeI1Q1cheS4SqRnpOoo1bSHeEmXwgRXFpVhIt7V3WcmN1dRruq8mLjGwfC1PX1Mb2I/UsyUz0nWzF3gSHw6QFxr3S3W22Gq9iMPNAFvSx7r6+9rWxEKPGQVKuTzxVzR0fTfSIIGt1Kjz+IbT
UGQkNFuf14hq6utU33Vc9pK+ExiqosJM9ezOYIPqPROR3IjJCRBJEZD2w3N3CbDxAeh5U7oKzJ8xW4hQB/n4sykjgzf0WcmOVrDWyeyZ95vnMcmwuqmJcTBjpST7smJi8CPyDfeKBy5UMxoBcgzF5cC/
wAfCsqt7iVlU2nqEneOsDc0KWZCbR1NbJ+6WnzZZycbq7jKyetIVGlo+FqW9uZ1tZHUuzknzTfdVDSCRMmm8YENuN9QmDMSCjMAoVlmGk86aKT/+lDCNiJkJilk9kY82bGMPIsEBrZGMd32Zk9fiA+2p
LcbXvu696SM+Dc5XGqN0GGJwB2Q68pqqLgdlAMvChW1XZeI70PKjYCQ0VZitxikB/P25IT+CNkhraOr3cjVWyBgIcZTIszqaiasZGh5GR7MPuqx4mLwa/QJ944HIVgzEgC1T1zwCqel5Vv4ljRriND5B
+k/HqC26srCQa2zr5wJvdWN1dxr912kIIsnbBwbMt7Ww7fNr33Vc9hI40Uq5L1lq+KZurGMxEwhMiMkpE5ojI1SJi/ZKhNn8ndhIkZEHxq2YrcZorJsYSGRLARm92Y534yHBfZfiC+6qGzm5l2XBwX/W
QsRIayqFyt9lKvILBpPF+CXgPo2bVvzteH3GvLBuPkpFnpCda3I0VFODHDRmJbPVmN1bxGmNSWpovuK+qGD0qlMyUYeC+6mHKUsONVfyK2Uq8gsG4sP4JI/ZxXFWvA6YDZ92qysaz+JAba1l2Eo2tXur
G6u4yJqOlLTQmp1mYhpYOPjx8mmXDxX3Vg+3G+hSDMSCtqtoKICLBqnoAmOJeWTYeJXYSJGT6RHDQq91YJ7ZDU41vZF+VVNPRNUyyry4k4yaHG8vOxhqMAakQkZHAGmCriKwFjrtXlo3HSV8J5R9Dg1v
7crmdoAA/FmUksrXYC91YJQ731eTFZitxmg37qhgTHUr26CizpXieKUscbizrxw2dZTBB9JtU9ayqPgL8G0aPDus/Qtl8mp6grg/UxlqWbWRjvX/Ii9xY3d2Gi3DSAsu7r840tzvcV8nDy33VQ+jIv08
qHOZurMGMQD5BVd9V1XWq2u4uQTYmEZtmuLF84KnqikmxRIUGepcb68RH0FRtuD8szpaSajq7lRuzh6H7qgfbjQVcogFxFSISLSJbRaTU8Tqqj31yReQjESkWkX0icnuvbU+KyFER2etYcj37DXyUT9x
Y1s7GCnTUxnqjpMZ7amMVv+KYPOgb7qvUmGEyebA/piwB/yCfeOByBlMMCMZExDdVNQ14k74nJrYAn1PVDGAx8CtHLKaH76pqrmPZ637Jw4DMm41XH2jfuSw72XBjeUM2Vlen4e6YvMjy7qu6pja2ldU
Nv+yrCwmJgonzjXtlGNfGGsw8kH/sa4TgJHnAU473T9FHTEVVD6lqqeP9SeAUEOdiHTa9iZkIidk+keN+uaM21sZ9J82WAsc/gObavxtoC9NTun3ZcHZf9ZBxE5yrgMp8s5WYxmBGIAnAThF5UUQWu6i
QYoKq9jioqx3X6BcRmQMEYRR07OFRh2vrlyLSb0NpEXlARPJFJL+21kINh8wic5Xh1z1zzGwlThHo78eidGNSoelurKJXIHAEpN1grg4XsLHwJBNiR/h26fbBMmWJUeK96GWzlZjGYLKw/hVIw8i+uh8
oFZH/FJGJAx0nIm+ISFEfy6caQKuqAv2mMohIEvA34POq2jNW/D4wFWOCYzTwLwPof0JVZ6nqrLg4ewBzUXqCvEXWH4Usz0mmub2Ltw+cMk9EV4eR2TZlieU7D55uauOjsjqWZQ9z91UPIZEw+QYjDtL
tJbE2DzOoGIjjR77asXRilHhfLSI/G+CYBaqa2ceyFqhxGIYeA9HnHS4ikcBG4Iequr3XuavUoA34C0a5eRtXMCoVUmb5hBvrsgnRxIYHsd5MN9aRd+H8GZ9wX20uqqZbGZ6TB/sjc5UxOfT4NrOVmMJ
gYiD/JCK7gJ9hlHHPUtWvATOBVUO87jqgpy3ufcBn2nyJSBDwKvBXVV19wbYe4yMY8ZOiIeqw6YvMVVBdCKdLzVbiFAH+fizLSuLN/adoaus0R0TxKxAc6ROdB9fvPcmk+HCmJkaYLcV7SFtkuCeHqRt
rMCOQaOBmVV2kqi+pageAw5104xCv+xiwUERKgQWOz4jILBH5o2Of24Crgfv7SNd9RkQKgUIgFviPIeqw6YuMlYD4jBurrbObN0pqPH/xzjbYvwGmLoOAfsN0lqCq4Tw7jtWzImeYTh7sj6Awwz1ZstZ
wVw4zBhMDeVhV+yxdoqr7h3JRVa1T1fmqmuZwddU71uer6pcc759W1cBeqbqfpOuq6vWqmuVwid2jqk1D0WHTD5HJMHaeT7ixZowdRXJUCOsLTHBjlb0FbQ2QYX331YYCI+dlRU6yyUq8kMxVcL7ecFc
OM8yaB2Lj7WTeDLUHoKbYbCVO4ecn3JiTzHultZxt8XABhcKXIDQaJl7n2eu6gXUFJ8keHcW4WGs3wXILk+ZDcJRPPHBdKrYBsemb9JUg/lC4+uL7ejnLs5Pp6FJeL6723EXbmuDAJsMd6B/oueu6gaO
nmymsbLBHH/0REAzTboT96w235TDCNiA2fRMeBxOuNQyIxQvGZaZEMi4mjPUFHqyNdXAzdJ6HrFs9d003sW7vSUTgxmzbgPRL5s3Qdg5Kt5qtxKPYBsSmf7JuhYYTUL7DbCVOISIsz0lmW9lpahs99IR
Y+BJEpsCYyzxzPTehqqwrqGT2uGgSo0LMluO9jL8WwmKh8EWzlXgU24DY9M+0G43+FYUvma3EaVbkJNOtsMETc0Ja6qHsTSO46mftW2x/VSNltc22++pi+AcYo5CDr0Frg9lqPIa1/7pt3EtwhJGiWPy
K5VMU0xIiyEiOZM0eDzTMKlkD3Z2+4b4qOEmAn9iTBwdD1m3Q1WbEQoYJtgGxGZis26ClDo68Y7YSp1mZm0JBRQNHat2c9V24GmInQ2KWe6/jZrq7lXV7K7kyLZboEUFmy/F+Rs+CUeN9YsQ+WGwDYjM
wkxZAyEifuCmW5yQjAmv2utGN1VABxz80Rh8Wn3D38dF6Tja0ctP0FLOlWAMR4//96HvQ6MGMPxOxDYjNwAQEQXqeMaO6vdlsNU6RGBXC5RNjWLu3EnVXZllP2nPmUKv8eA9r9lQyIsifG9ITzZZiHbJ
vA+0eNqVNbANic3Gyb4OOZiM11eLk5aZwvK6FPeVnXX9yVdj3AoyebfRWsTCtHV1sKqxicWYSoUH+ZsuxDrFpkJQL+4ZHNpZtQGwuztjLjZTUfS+YrcRpFmcmEhTgx1p3BNOrC+FUCWTffvF9vZw395+
isa3Tdl8NhezboGqv5YuRDgbbgNhcHD8/40fx8JvQaEJRQhcSGRLIwmkJbNhXRUeXi1uRFjwPfoE+4b56dU8lCZHBzJsYY7YU65G5CsTPJx64LoZtQGwGR86doF0+EUzPy02mrrmd90td2KGyq9OYRDZ
5EYRFu+68JlDf3M47B0+Rl5uCv5+1EwFMISLRqOJQ8LzP90u3DYjN4IibDCkzoeA5s5U4zbVT4hkVFsjLu13oxip7y+h7nnOn685pEhv3naSzW1mZa7uvhkzOXdBQDsc/MFuJW7ENiM3gybkTaooMX7+
FCQrwIy83ha3FNTS0uGiC5L7nIXSUT/Q9f3VPJVMTI0hPtvueD5mpy4xGYnut/8A1EKYYEBGJFpGtIlLqeB3Vz35dvZpJreu1fryIfCwih0XkBUf3Qht3k7nK8PH7wE1xy8zRtHd1s84VpU1aG+DARuP
fJ8Daf4pltU3sPnHWDp47S1CYUYm5ZK1RmdlHMWsE8hDwpqqmAW86PvfF+V7NpFb0Wv9T4JeqOgk4A3zRvXJtAMO3P3mR4evvMqlFrIvISI5kamIEq/PLnT9ZyVrobPUJ99VL+RX4+wk3zbANiNPk3m2
kv+9fd/F9LYpZBiQPgLXpRAAAHflJREFUeMrx/imMvuaDwtEH/Xqgp1HFJR1v4yS5dxm+/rI3zVbiFCLCLTNHU1DRwKGaRudOtvc5iJlkxIgsTGdXN6/sruC6KXHER9iVd51mzFyIngB7nzVbidswy4A
kqGpPc4ZqIKGf/UJEJF9EtotIj5GIAc6qas8jcAVgPy55ikkLjS57PnBTrJyeQoCf8PKuiqGf5PRhOLHNeNq0eOmS90prOdXYxq2zxpgtxTcQMUalx96HsyfMVuMW3GZAROQNESnqY8nrvZ8aNSX6qyu
RqqqzgLuAX4nIJU/vFZEHHEYov7bWhWmbw5WAIGNOyMFN0FxnthqniA0P5top8byyp5LOoc4J2fM3o3Nj7l2uFWcCL+VXEDMiiOunxpstxXfIucN4LXjeXB1uwm0GRFUXqGpmH8taoEZEkgAcr6f6OUe
l4/UI8A4wHagDRopIgGO30UC/+Ziq+oSqzlLVWXFxcsS77fsOaGfdCV7tPTJS6ZeZoahvbeL/09KUf3NVppDVPXmTk/luY+uZ23thfw03TUwj0t5MzXcbIsTD+atj7jE/OCTHrL2UdcJ/j/X3A2gt3EJF
RIhLseB8LXAGUOEYsbwO3DHS8jRtJyICUWbD7Kcu3u71+ajzRI4J4YecQgumlW6CpBqbf63phHmbNnko6utR2X7mD6Z+DM8fg6LtmK3E5ZhmQx4CFIlIKLHB8RkRmicgfHftMA/JFpADDYDymqiWObf8
CPCgihzFiIn/yqHobmPE5qD0AFTvNVuIUQQF+rJqRwhv7azjV2HppB+/+K4QnWH7uh6ryYn45OaOjmJIYYbYc32PacmOO0O6nLr6vxTDFgKhqnarOV9U0h6ur3rE+X1W/5Hi/TVWzVDXH8fqnXcUdU
5qjpJVW9VVQ81urb5hMybIXCET9wUd8wZS2e3svpSgumN1cYIJOdOo52phSmsbOBAdSO32KMP9xAYYvyd7N8AzUNwlXoxtrPTZmgERxhGpOgVaD1nthqnmBgXztzx0Ty/o5zu7kG65PY+a9QG8wH31TP
bTxAa6E9ert333G3MuA+6O3wie7E3tgGxGToz74eOFqNnusW5a+5YTtS38GHZIJ4Qu7thz9OQegXETnK/ODfScL6DdQUnyctNJjIk0Gw5vkv8VBhzmU/EDXtjGxCboZMyE+LTYZf13ViLMhIZFRbIczs
Gka9/9F2oLzOeKi3Omj2VnO/o4u65qWZL8X1m3gd1h42Wxz6CbUBsho6I8SN6cjec3GO2GqcICfRn1YzRbCmuobbxIiG1nX+EsBij1pGFUVWe+fg42aOjyBodZbYc3yd9JQRHwa4nzVbiMmwDYuMcOXd
AYBjs+OPF9/Vy7pxrBNNf2jVASm9DhTGJcsbnICDYc+LcQP7xMxyqaeLuuWPNljI8CAozuhWWrPOZYLptQGycI3SkMTO9aDW01Jutxil6gunP7ThBV3/B9Py/GD7sWV/wrDg38Mz240QEB7A8xw6ee4z
ZX4SuNp/IXgTbgNi4gjlfNqrR7vmb2Uqc5nPzxlFef563DvRRHKGz3bjxJy82ZhhbmPrmdjYVVnPzjBTCgqydhmwp4qcZM9N3/tnyFa3BNiA2riAhA1KvNGID3V1mq3GKRRkJJEWF8JcPj3524/51RiX
i2V/yvDAX88LOctq7urnLDp57nrlfhXMVcHCj2UqcxjYgNq5hzpeNiqOlW8xW4hQB/n7cOy+VbWV1HKy+oMz7zj/CqPEw8XpzxLmIjq5untp2jCsmxdgzz82gZwT78f+ZrcRpbANi4xqmLoOIZNjxhNl
KnObO2WMJDvDjyW3H/r6yuhBOfGT4sP2sfdtsKqyi+lwrX7hivNlShid+/jD7y0Y6b3WR2Wqcwtp3go334B8Isz4PZW9B7SGz1TjFqBFB3DQ9hVf3VHC2pd1Yue23EBRu+ZnnqsqfPzjKhNgRXDfFLtt
uGtPvgYBQ2PH/27v3sKrK7IHj3wWCoGgoKhZewLQQRSjIS14iLE3Hqay8dZlKzcxL1pTlzPSbcWa62Tjl5DQ2qZU2lo2XtDGdGstSQw1QxAuZWioYGoMFoqJc1u+PfXAIIW7nnH2OvJ/n4Qn22WefdXb
COu/77r2Wd49CTAIxnCfuPvBtDFtfsTuSeruvbziFRaUsTc6EvKPWVWZX/8K66syLpR7+np1ZedzfNxwfH+9ugOXVmrS0LulNX+bVVy+aBGI4T1Abq7FS2jtw8rjd0dRLZNvm9OkUwuKQ5RunQdaai1
```

```
+ermFm7/hkkA/bo9rZ3coRq+JUHwGkr23mLhJIIZzXTvVajbl5UNzgHH9IsjPO0FJ8hvWXcQtvPuKpcwTp/lwzzHG9OxgLt31BKFR0GUwbJsH507bHU2dmARiOFfI5Vb/g+QFcPZk9ft7sMTINkwN3oJ
fcQGlfabaHU69vfH5IXxEuPda706EF5V+j8LpXKs4pxcyCcRwvr7ToDAPtnv3jYU+Wsw98gFbS7vycX6Y3eHUS27BWd754gg3x1zGpZcE2h2OUaZjH6tKb9JcKCmyO5pas2UcKyItgXeBcOAQMFJVv6+
wz/XAS+U2RQKjVXWViLwJXAfkOR67T1XT6hJLUVERWVlZFBbWshvdRSogIIB27drh51eP0t7t4q0bC7e8Yt0f4uulZcJ3r6TJmWO8FziWLzcc4IaubRDxzoXnBZu/obC4hEnXe3f5+YtSv0fhnVFWb52
YUXZHUyt2TYTOAD5W1edFZIbj5yfL76CqG4BYOJ9wDgDl71KbrqrL6xtIVlYWzZo1Izw83Gv/ODiLqpKbm0tWVhYREfW8R6Dvw/D2SNi1HGLHOCdAdyopho0vQJsooq8aybur97LlYC7Xdm5ld2S19sP
pcyxOOsTPoi+lc5sgu8MxKuoyyGqLsPkliB7hVfcZ2RXpLUBZNbFFQHV1se8A1qmq01eaCgsLCQkJafDJA0BECAkJcc5orPONEBoNG//knTV/di+3ejckzOCO+A60btaYv3160O6o6uT1zw9x6lwJUxL
N6MMj+fhA30cgJwP2f2h3NLViVwIJVdVsx/fHgNBq9h8NvFNh2zMiki4iL4lIlXW1RWSCiKSISEpOTk5V+9Q07oue086Fjw9c/2ur8VL6Uucc011KiuGzWRDaHSJ/ToCfL+P7RbD5wH/ZceT76p/vQfI
Li3jj828Y3C2UyLbN7Q7HqEr32yC4I2x41up46SVclkBEZL2I7K7k65by+6mqAlX2eBSRS4FooHxq/hXWmsg1QEsqTH9VOP5rqhqvqvGtW7euz1syauvKIXDZ1fDpLKuSrbfY9U848TUkzDg/nXBX746
ENPVn9kf7bA6udhYnHeJkYTFTE7vYHYrxU3z9rA9cx9IhY7Xd0dSYyxKIqt6gqt0r+VoNHHckhrIEUUnt7PNGAu+p6vlLFFQ1Wy1ngTeAnq56H+7wzDPP0K1bN3r06EFsbCzbtm1j/Pjx7N271+7Q6kc
EEn8DeUdgx2K7o6mZkmL47AVoGw2Rw85vDmrciCmJnfn8QC6b9lc+kvU0eaeLmL/pGxIj29A9zHQc9HjRI6B1V/jkGa+Z9rVrCut9oKyh9L3AT6XcMVSYviqXfARr/cRrK5Jt2bKFNWvWsH37dtLT01m
/fj3t27dnwYIFREVF2R1e/V0+0LpMceNsKDpjdzTVS18K338DCb+yEmA5d/bqQFhwIC/8ex+lVTWc8iCvfHqA/MIiHh90pd2hGDXh4wuJT0HufthZccbeM9l1FdbzwD9FZBxwGGuUgYjEAxNVdbzj53C
gPfBZhecvEZHWgABpgFNqTPz+X3vY+22+Mw51XtRlzfndz7tV+Xh2djatWrWicWNrGadVK+sqn4SEBGbPnk18fDxBQUFMmzaNNWvWEBgYyOrVqwkNDSUnJ4eJEydy5MgRAObMmUPfvn2dGn+9iVi/FIu
GQcrr0Gey3RFV7dxpaw760li4cugFDzdu5Msvb7yCx5btZO3ubIb18NxOfpknTvPm54e47ap2RF1m1j68RuTPHNO+z1u1sjy8bbItIxBVzVXVgaraxTHVdcKxPaUseTh+PqSqYapaWuH5iaoa7ZgSu1t
VC9z9Hpxl0KBBZGZmcsUVVzBp0iQ++6xiroRTp07Ru3dvdu7cyYABA5g/fz4A06ZN49FHHyU5OZkVK1YwfryHNjqK6A+dEqwrsjy5cFzSy5B/FG567oLRR5lbrwrjytBm/Pmjrygq8dzFzj9/tA8ReHz
wFXaHYtSGCAz8rdVwKuV1u6OplimIU85PjRRcJSgoiNTUVDZt2sSGDRsYNWoUzz///I/28ff3Z9gwaz4+Li6O//znPwCsX7/+R+sk+fn5FBQUEBTkgdf6D34WXu0PG56Bn/3Z7mgulJcFm+dYNa86Xlv
lbr4+wvTBVzJ+cQpLkzO5p7fnlQXZlZXHqrRvmZRwubnr3Bt1SoCI66xRSPRIaBpid0RVMgnEA/j6+pKQkEBCQgLR0dEsWrToR4/7+fmdv7zW19eX4mJrga20tJStW7cSEBDg9phrLbSb1Qo2eb5V9r1
ttN0R/dj6mVbF3Rv/UO2uA7u2oXenlsz+cB9Du7clJMhzphlUlWfXZtCyqT8TEy63OxyjLkRgyCyY1xc+ngk3z7U7oip5zy2PF6l9+/axf//+8z+npaXRsWPNPtUOGjSIuXP/948rLa1O1Vzc5/pfQWA
LWPsEqActQmd+AbuWwbVTalRxV0T44y3dOXW2mOfWfemGAGtuTXo2W77OZdrALjQP8NISMga06Qq9H4LtiyEz2e5oqmQSiM0KCgq49957iYqKokePHuzdu5eZM2feZK2fW6Lkvv/wyKSkp9OjRg6ioKF599VX
XBltfgS2s+d0jSbB7hd3RWEpLYN2TENQW+v2yxk/rEtqMCQM6sTw1i21f57owwJr7/tQ5Zr6/hx7tLuFuD5xaM2opYQY0uxTWPmb9O/VAop70SdDF4uPjNSUl5UfbMjIy6Nq1q00ReSaXnpPSEph/PRR
8B5O22t/hb/McWP87uH0hRN9Rq6eeOVfCjS99RqCfLx883B//RvZ+Hnt82U7e23GUf03pZ668uljsXgHLx8LQ2VZhUpuISKqqxlfcbkYghnv5+MKwOXAqB9ZOtzeW7760FvUjh0H322v99EB/X35/czf
2f1fAgs1fuyDAmtu8/78sT83iwQGdTPK4mHS7zVpQ//iP8EOm3dFcwCQQw/3CroYBT1glQ3avtCeGkmJYNREaN7MSWh1rgA3sGsqQ7m2Z85/97D6aV/0TXODMuRJ+/d4uIlo15eGBpmTJRUUEfv4X0BJ
YOcHjprJMAjHs0f8xCIuDNY9C/rfuf/3P58C3O6xLioPqVyPt2eHRtGzqz9R3dlBw1v0lKGa+v4cjJ07z7PBoAvx83f76hou1jLD+nR5Jgk0v2h3Nj5gEYtjDtxEMf83qn756snuvyspMtq6x73YbdBt
e78O1aOrPnNGxHM49xW9Xu7eqzrvJR3g3JZOpiZ3pc7nn3i9g1FOPUdD9Dvj0OeuqQQ9hEohhn1adYdDTcPAT6xfDHfKOwtI74ZIwp97Q2LtTCA8P7MLK7UdZkZrltOP+lN1H8/i/1Xvo17kVj9xg7ji
/qInAsBeheRisGA9nPKOtgEkghr3ix8JVd1v9N9JcXEDu3GlYOsYq6jhmKTRp6dTDT03sQq+Iljy1arfL+4bknS5i0pLthDT15y+jY/H1MT1tLnoBl8DtC6wp36V3QZH9bbhNAvEAx44dY/To0Vx++eX
ExcUxdOhQvvrqq1odY+jQofzwww8uitCFRKxF7IgB8P5UOLTZNa+jak2VZafDHQutG7WczNdHmHvnVbRu1pj730zmq+Mnnf4aAKfPFTPhrRSy887wyl1Xe9Sd8IaLdegFw1+Fw5/DygdsX1Q3CcRmqsr
w4cNJSEjg4MGDpKam8txzz3H8+PFaHWft2rUEB9t8T0Vd+frByLegZSfrk9V3Gc49fmkprHsC9qyEG2bCFYOde/xy2jQL4B/jeuHv68M9C7eRecK5XZjPnCth3JspJB86wZ9HxnJ1hxZOPb7hBaLvgEH
PQMb78O8ZtlZ1MLWwyls3A47tcu4x20bDkOerfHjDhg34+fkxceL/KtLHxMSgqkyfPp1169YhIjz11FOMGjWK7OxsRo0aRX5+PsXFxcybN4/+/fsTHh5OSkoKBQUFDBkyhH79+pGUlERYWBirV68mMDC
QgwcPMnnyZHJycmjSpAnz588nMjLSue+3rgKD4a5/wsJB8PpNMPptCHdCafqSIlj1kFWqpM8U6Dut/sesRoeQJiwe15ORr27hnoXbWPJAb8KC61/UsLCohPGLk9n2TS4vjozl15hjPLSdvuNi1U+BkNmz
5K/g0stYSfdx/BZ4Zgdhs9+7dxMXFXbB95cqVpKWlsXPnTtavX8/06dPJzs7m7bffZvDgwecfi42NveC5+/fvZ/LkyezZs4fg4GBWrLDKhkyYMIG5c+eSmprK7NmzmTRpksvfX620CIdxH0HT1vDWrbB
ref2Od+60tWC+axkM/J31S+asnu/ViGzbnDfu70luwTl+PnczSQf+W6/jHcsr5BcLvyDpYC6zR8Rw61VhTorU8Fo3/hF6Pghb/wZvj4JC99+HZEYg5f3ESMHdNm/ezJgxY/D19SU0NJTrrruO5ORkrrn
mGsaOHUtRURG33nprpQkkIiLi/Pa4uDgOHTpEQUEBSUlJjBgx4vx+Z8+eddv7qbGyJLL0TlgxzprOGvA4+NXyE/yRbfCvaZDzpbXGEn+/S8L9KXEdW7B6Sl8efCuVuxduY8aQSB7o3+l8ZeWa+nDPMZ5
ckc654lLmjIrllliTPAzAxweGvgBtIq2qDgtuhNFLoJX7bia1ZQQiIiNEZI+IlDq6EFa1300isk9EDojIjHLbI0Rkm2P7uyLi757Ina9bt26kpqbWeP8BAwawceNGwsLCuO+++1i8+MJe42XdDeF/5d9
LS0sJDg4mLS3t/FdGhpPXGpylSUu4ZxXEjIFNs+GvPWHPqprN9RbmwZpfwuuD4exJuGu5LcmjTKfWQbw3uS83dW/Ls2u/ZMSrW9i0P4ea1KA7knuaJ5en8+BbqbRv0YQ1U/uZ5GFcKH6s9fty6jv4W2/
44DE4Wbs11LqyawprN3AbsLGqHUTEF3gFGAJEAWNEpKxJ+CzgJVXtDHwPjHNtuK6TmJjI2bNnee21185vS09PJzg4mHfffZeSkhJycnLYuHEjPXv25PDhw4SGhvLAAw8wfvx4tm/fXqPXad68ORERESx
btgywFu937tzpkvfkFH4B1tUm966BgOaw7F6Yn2j1Vv92h7UwXqboDGT8y7o+/sVukPoG9J4Ek7dBlxvsew8OQY0b8cqdV/Ps8GiO/nCGexZ+wW3zkliWkklGdv75zoalpcrx/EI+zjjO2DeTuW72BpZ
vz+LB6zqx4qFr6dTaAxuFGZ4hor9VnPTqX0Dqm/ByrNU2IWONVbjURWyZwlLVDKC6oXxP4ICqfu3Ydylwi4hkAInAnY79FgEzgXmuiteVRIT33nuPRx55hFmzZhEQEEB4eDhz5syhoKCAmJgYRIQXXni
Btm3bsmjRIv70pz/h5+dHUFBQpSOQqixZsoSHHnqIp59+mqKiIkaPHk1MTIwL350TRPSHBzfC9kXWL8Ynf7S+/INAfKD4LJQ4puICW0L34RA/Di67cGrPTiLCnb06cHtcGMtTs/jbhoNMX54OgH8jH1o
HNea7k4UUlVgjk1ZBjZl6fWfG9OpgugoaNdOsLQx7ybpYZMOz1gepL/5uPdYiAsa84/TL120t5y4inwKPq2pKJY/dAdxU1iNdRO4BemEli62O0Qci0h5Yp6rdq3iNCcAEgA4dOsQdPnz4R4+bcu4X8uh
zUvAdHNwAR1OtBNLIHxoFQofeEN7PuiTYC5SUKt/8t4A93+az99t8ck6epe0lAVwaHEiHlk3o0ynE9vLwhpcrKoTsnZC51Sp/cus8azRfB1WVc3fZCERE1gNtK3noN6q62lWvW5Gqvga8BlY/EHe9ruE
iQW0gZpT15cV8fYTObZrRuU0zs65huIZfgHXjYYdeLnsJlyUQVa3v5PNRoH25n9s5tuUCwSLSSFWLy203DMMw3MiTx8jJQBfHFVf+wGjgfbXm3DYAZe3j7gXqNaJpSF0Zq2POhWEYNWXXZbzDRSQL6AN
8ICIfOrZfJiJrARyjiynAh0AG8E9V3eM4xJPAL0XkABACLKxrLAEBAeTm5po/nFjJIzc3l4CAALtDMQzDCzT4nuhFRUVkZWVRWGh/ZUtPEBAQQLt27fDz847FaMMwXM/ti+jews/Pj4iICLvDMAzD8Dq
evAZiGIZheDCTQAzDMIw6MQnEMAzDqJMGtYguIjnA4Wp3rFwroH41ub2fOQfmHDT09w8N8xx0VNXWFTc2qARSHyKSUtlVCA2JOQfmHDT09w/mHJRnprAMwzCMOjEJxDAMw6gTk0Bq7rXqd7nomXNgzkF
Df/9gzsF5Zg3EMAzDqBMzAjEMwzDqxCQQwzAMo05MAqkBEblJRPaJyAERmWF3PO4kIu1FZIOI7BWRPSIyze6Y7CIiviKyQ0TW2B2LHUQkWESWi8iXIpIhIn3sjsndRORRx+/BbhF5R0QadOlqk0CqISK
+wCvAECAKGCMiUfZG5VbFwGOqGgX0BiY3sPdf3jSs1gIN1V+Af6tqJBBDAzsXIhIGPAzEO1po+2L1KWqwTAKpXk/ggKp+rarngKXALTbH5Daqmq2q2x3fn8T6o9HgerCKSDvgZ8ACu2Oxg4hcAgzA0Xt
HVc+p6g/2RmWLRkCgiDQCmgDf2hyPrUwCqV4YkFnu5ywa4B9QABEJB64CttkbiS3mAE8ApXYHYpMIIAd4wzGNt0BEmtodlDup6lFgNnAEyAbyVPUje6Oyl0kgRo2ISBCwAnhEVfPtjsedRGQY8J2qpto
di40aAVcD81T1KuAU0NDWA1tgzT5EAJcBTUXkbnujspdJINU7CrQv93M7x7YGQ0T8sJLHElVdaXc8NugL3Cwih7CmMBNF5B/2huR2WUCWqpaNPpdjJZSG5AbgG1XNUdUiYCVwrc0x2cokkOolA11EJEJ
E/LEWzd63OSa3ERHBmvfOUNUX7Y7HDqr6K1Vtp6rhWP//P1HVBvXJU1WPAZkicqVj00Bgr40h2eEI0FtEmjh+LwbSwC4kqKjBt7StjqoWi8gU4EOsqy5eV9U9NoflTn2Be4BdIpLm2PZrVV1rY0yGPaY
CSxwfpL4G7rc5HrdS1W0ishzYjnV14g4aeFkTU8rEMAzDqBMzhWUYhmHUiUkghmEYRp2YBGIYhmHUiUkghuFGIpJUi30/FZH4avY5JCKtanHM+0TkrzXd3zB+ikkghuFGqtq
g71w2Li4mgRhGJUTkGhFJF5EAEWnq6AHRvZL9VolIquPxCY5tHUVkv4i0EhEfEdkkIoMcjxU4/nupiGwUkTRHb4n+1cQzT0RSHK/z+woPPyEiu0TkCxHp7Ni/tYisEJFkx1dfp5wYwyjH3IluGJVQ1WQ
ReR94GggE/qGquyvZdayqnhCRQCBZRFao6mERmQXMA74A9lZStfVO4ENVfcbRc6ZJNSH9xvE6vsDHItJDVdMdj+WparSI/AKravAwrN4dL6nqZhHpgFVJoWvtz4RhVM0kEMOo2h+waqEVYjUSqszDIjL
c8X17oAuQq6oLRGQEMBGIreR5ycDrjkKVq1Q1rZJ9yhvpGOE0Ai7Fam5WlkDeKffflxzf3wBEWSWbAGjuqKhsGE5jprAMo2ohQBDQDLigdamIJGD9oe6jqjFYtZECHI81warcjOMYP6KqG7EaNB0F3nS
MHiolIhHA48BAVe0BfFAhHq3kex+gt6rGOr7CVLWg2ndsGLVgEohhVO3vwP8BS4BZlTx+CfC9qp4WkUislr9lZjme91tgfsUnikhH4LiqzsfqcvhTpdGbY/XfyBORUKz2yuWNKvffLY7vP8Iqflj2epW
NggyjXswUlmFUwjEiKFLVtx3rDkkikqiqn5Tb7d/ARBHJAPYBWx3PvQ64BuirqiUicruI3K+qb5R7bgIwXUSKgAKgyhGIqu4UkR3Al1jdMT+vsEsLEUkHzgJjHNseBl5xbG8EbMSaTjMMpzHVeA3DMIw
6MVNYhmEYRp2YBGIYhmHUiUkghmEYRp2YBGIYhmHUiUkghmEYRp2YBGIYhmHUiUkghmEYRp38P60khJOkqACmAAAAAElFTkSuQmCC\n",
            "text/plain": [
              "<Figure size 432x288 with 1 Axes>"
```

```
          ]
        },
        "metadata": {
          "tags": [],
          "needs_background": "light"
        }
      }
    ]
  },
  {
    "cell_type": "markdown",
    "metadata": {
      "id": "R5IeAY03L9ja"
    },
    "source": [
      "###Subplots "
    ]
  },
  {
    "cell_type": "markdown",
    "metadata": {
      "id": "CfUzwJg0L9ja"
    },
    "source": [
      "You can plot different things in the same figure using the subplot function. Here is an example:"
    ]
  },
  {
    "cell_type": "code",
    "metadata": {
      "colab": {
        "base_uri": "https://localhost:8080/",
        "height": 281
      },
      "id": "dM23yGH9L9ja",
      "outputId": "a7eefc31-5df8-4174-c10e-471cc3f012a4"
    },
    "source": [
      "# Compute the x and y coordinates for points on sine and cosine curves\n",
      "x = np.arange(0, 3 * np.pi, 0.1)\n",
      "y_sin = np.sin(x)\n",
      "y_cos = np.cos(x)\n",
      "\n",
      "# Set up a subplot grid that has height 2 and width 1,\n",
      "# and set the first such subplot as active.\n",
      "plt.subplot(2, 1, 1)\n",
      "\n",
      "# Make the first plot\n",
      "plt.plot(x, y_sin)\n",
      "plt.title('Sine')\n",
      "\n",
      "# Set the second subplot as active, and make the second plot.\n",
      "plt.subplot(2, 1, 2)\n",
      "plt.plot(x, y_cos)\n",
      "plt.title('Cosine')\n",
```

```
        "\n",
        "# Show the figure.\n",
        "plt.show()"
      ],
      "execution_count": 101,
      "outputs": [
        {
          "output_type": "display_data",
          "data": {
            "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAXIAAAEICAYAAABCnX+uAAAABHNCSVQICAgIfAhkiAAAAAlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9uMy4yLjIsIGh
0dHA6Ly9tYXRwbG90bGliLm9yZy+WH4yJAAAgAElEQVR4nO3deVzU1f7H8ddh2HcFFAVZVBQ3ZHPNbLG6mpZmmkuall3LVtv35bbdbrua3XJLM9MsM8tKK7PScgdDRVFBcEFAUQARkRVBcEFUQARk387vD/D+bLFchvn
O8nk+Hj4eMsLMe0bn7fme+Z7zVVprhBBC2C4nowMIIYS4MFLkQghh46TIhRDCxkmRCyGEjZMiF0IIGydFLoQQNk6KXDgspdSNSqlvjc4hxIVSch65sHdKKn/fktSql0pdQJJpdRqVS4EU9IiN+THf2bi9Qp5RaoJQapJRq9jffPwAcQANQE
/AFBKDQUeB4YDQcA6YHGTpRbiHEiRC7umtS4B+gEamA3kK6W+UEq1PMOPvKy1LtJaHwTWArGNt98Ftrna61rgVeAmJlVC6sgRS5sHuN5TtRax0KdAVaA2+d4duPnvb7csC78ffhwLTGKZcioBBQQEg
TxRbirEmRC4eitd4NzKeh0M/FIeA2rbX/ab88tNa/mj2kEOdIilzYNaVUtFLqAaVUaOPXbYAxwMZzvKt3gceUUl0a78dPKTXSvGmFOD9S5MLenQR6AZUUmU0FHga8MC53InWejjnwH2CJUqqk8T4GmTm
rEOdFFgQJIYSNkxG5EELYOClyIYSwcVLkQVLkQghh46TIhRDCxhmyaVZgYKCOiGItw4qGFEMJmJScnF2itg35/uyFFHhERNQRVJSkhEPLYQQNkspdeDPbjfL1IpSap5SKk8plWaO+xNCCHH2zDVHPh8zbH8
IcQ7MMrWitf5ZKRVhjvuyNxXVdSQdKGTP0ZNkHCslM7+Ukooaquvqqa6tx8PFRLCfGk0F8F+7kQEeNEjojlxYf64u5iMji6ExR04XsaGfcfJKigju6CMg8fLqaytQ2vQaHzcXAhr7klYgCftg7zp2z6z0Ga
eRsc2nMXmyJVSk4HJAGFhYZZ6WEMUV9SwKu0I3+3KY31mPpU19QAEeLnSvoU37YK8cXNxwtXkRHl1HUeKK9i47zjLUw6jNbg6OxEf5s/Q2BCGxLTCx93F4GckRNPQWpN84ARfpuby09589h89v89h8vBxreA+H
NPQkP8MLLzYQClFIUlVeTkXeSH/bkUV3b8L5qG+hF/w5BjEgIpWuIn4HPxjhmW6LfOCJfqbX+213lEhMTtT1+2dYcyqvrCPH34CfZ+2efDL/l1bV8npLLSH/w5BjEgIpWuIn4HPxjhmW6LfOCJfqbX+213lEhMTtT1+2Lkvv5T5v+xn2dYcyqvrCPH34IpOLbgsugXdQvwI8Hb7y58vLq9hy/5CNmUfZ+2efDL
zSnF3ceLqbq2Y1C+SLq0d8x+psD/l1bV8npLLwo0HSD9SgoeLiT7tArikQxD9ogKJCPDC5KTO+PP19ZrM/FLWZRSwPiOfX/cdp6q2nu6hfoztFcbQ2BC7PKpVSiVrrRP/cLsU+YXbX1DGy9/tYWXaEMxDGy9/sZtXOo7i
anLg2tjUT+kTQNcSX064Udk601qTmFLM06RBfbMultKqWq7sFc98VHYhq6WPmZyCEZdTU1bNk80GmcrcmgoLSaTq18ualPOENjW+Ppev4TBMUVNSzfmsNHmw+y91gprfzcue+KDgyPD8HZD/LZaTIm0B
xRQ0z1mSwYMN+XE1OTOoXyfg+EQT5/PXI+3weZ+66LOb9sp+y6lpGJoTy+NWd8Pd0Nevjc+CNGUVu88ysvf7Ca7oIyekc158KqO9Ihodt6DnT+jtebXfcd5ZfUeUg8Ug8V0b6FN08O7sSlHVuY7TGM1KRFrpR
aDFwKBALHgGe01nPP9P32UOTf7TrGY59t53hZNSMQnnwqo608HVv0sc8UVbNOz9mMu+X/TTzdOHZa7swuFsrs74RhDC346VVPLUija93HCWqhTePDorm8ugWTFrvVmvNqrSjvLp6D1kFZVwfH8rTQzrz
j52nbnzc1+Yj8XNhykZ+srOH5lbtYmpRD51a+vDIixuIfsOzMLebRZTvYcbiYKzu35JJaNzY2jmJaNzYX2+2XGEJz9P42RlLVOvjGLyxW0tOtOtVRVVvHjDWZ/PenfQR4ufLv4ajZrAjp5g
pi5LJLapgyqXtuHdAB1ydjZ1/q62rZ94v2by6eg8W/d4F4ISyjurae51bu5MONB+kW4sfrN3Sng4Gf7aQdLubBT1Z3ZffQkt13Sloeu6miTc+dS5BdoeUoOjy7bQaC3G9PHxJIQ3tzoSAC
kHirijkVbyTtZyeNXd2Ji3wiZahGGOlpcyZRFyaQcLLKq0qyqreO5L3exaNNB+rQNYPqYOLN/ntXUpMjPU21dPS9/s5s567Pp3bY5M8fG/+1phJZWXF7DA59jWVtgMDxq6WyAOthOR4/vhjCPA6jaNMVSzT
SFdV1vDqyO1d3a2V0pD9YlpzD48t30t30MzTlbkTE23qtN4zFbm82/9CZU0dt3+YzJz12UzsG8HCSb3OqzQ5TZdUkDwjbaW+qFTXtzcX18RsvhJw2QK/8Dqzj5z9rhvuTnG9kN1PQ1w8Su/mKsQyMTumjqF0aD
GvbeRXzILjI50waTIz6C4oobxczexZncezew/twrPXdsHFike5Sike/EdH/j28G+szCxj57gbySiqNjiUcxOLNB7ltYRLRwT4sm9LX6tc6dGntx2d39Gntx2d39CXE34OJ72/m85TDRke6INbbTAbKK6lk1Hsb2Ha
oiBlj4hjfJ8LoSGdtTM8w5k5I5MDxMkbN2siR4gqjIwk79/YPGTz22Q76dwhi8eTeVnnU+mda+Xmw9PY+JIQ3Y+rH21jw636jI503KfLfOVZSyahZGzlYWM68iT0YEtPa6Ejn7NKOLVg4qScFJ6u44b0
NHCosNzqSsFNvfb+X177dy/C4EGbflHhBqzON4OfhwoJbenVV55Y888VO5q3PNjr1fiPKo1yIw2YkhDfnw1t7UVxew+hZGzlwvMzoSMLOvPX9Xt76oPoMRCaG8OrK7VU8
9/hU3ZxMzb4xnYJdgnlu5iznrsoyOdM5s85VvAvknqxg7exNHiwxNHiyuZf0tPqzm98EJ0b+PPR//sTXl1LWNnb5JpFmE2077P+F+J/+f6mL/c4MoWuJicmDE2jqu7BfCV+k2V+ZS5DScvjduziYOn6hg3sQ
e9Iiw/RI/pWuIHwsn9aKkouE5Hi+tMjqSsHHz1mfz5vd7uT7ePkr8FBeTE9NG/3+TE9NG/3+Zf5J0yOhIZ83hi7yypo5JC7aQXVDGnAmJ9G4bYHQks+sa4secCYkcCYnknKhgwvubKamMsMTqSsFGfpxzmuZW7GNglmFd
G2E+Jn+JicuLNUbFcHBXIo5/t4NudR42OdFYcushr6+q566Mkv6q566MUkg+e4I1R3bmofaDRkZpMr7YBvDsugd1HTvLPBUlU1dYZHUnYmLV78njwk1T78njwk1T6tA3grdGxdlfip7g5m3hh3XAJdQ/y4a3EKG/YdNzrzS33L
YItda8+TnaXyffoxnr+lik2ennKvLolvw+g3d2ZRdyMOfbqe+3vKreoVtSj1UxJQPk4lu5cOsmxLs8qINp/Nyc2b+xB6EN/dk8gdJ7D120uhIf8lhi/ydH/exZMsh7rqsPRP6Rhgdx2KGxobw8MCOrNi
Wyxvf7TU6jrABOSfKmbQgiSAfN+bf3NNhLj3YzMuV+bf0xN3VxM3vbyH/pPV+vuSQRf7V9iO8unoPw2Jb88BVHYyOY3FTLmnHmJ5teHttJks2JBiJZU1TJrfMBX3/sQeBNrIYh9zCfH3YO6ERI6
XVXHrBw17yFgjhyvy1ENF3L90GwnhzXj5+hiH3ClQKcVzQ7vSv0MgTvSv0MQT3yexq92sNeEML9TnyHtyy/l3XEJtG9h3cvum0pMqD/TRsexPaeI+z7eZpVTkg5V5LFdz6QcMh4nvj7X+e76+4mJyYOTaOtoF
e3PHRVg4el9Wf4rde+Cqdn/fm88KwrnZ9IsDZ+EeXYJ64uhOrdh7lrTUZRsf5A4cp8sqaOm5bmExExFdR3zHPAQ8c/4uDfsmqg1/PMD2TFR/L+lSYeY/+t+JvWLZHTPMKPjWIVJ/SIZkRDK9DUZrEo7SYnS
c33CIItda88TyNHYcLubNUbGGXqnE2kQEevH22Dgy8k5a7WGjsKyUgyd4cnkaF7UP4LFB0UbHsRpKKV4Y1pXYNv7cvzSV3UdLjI70Pw5R5At+3c+yrTncOyCKzvb6vX6msrFUUE8Mbgz3+06xvQfrO+
wUVhOXkklt3+YTEs/N94eEy8XKPkddxcT741PwPvNvNmX9+kMSJsmqjIwEOUOSbso7z/FfpXNGpJfcOiDI6jtW65aIIrosLYdqaDH7ck2d0cHGGAmrp67li0lZKWmaNT5QLep9BS1933h2fwLHiKqZayVG
sXRd5Xkkld36UQniAJ2+M6o6Tna5EMwelFC9d9l42OLX24d8k22frWAb38zW6SDpzgPyNi6NTK1+g4Vi0+rBlPX9OZn/bmW8VRrN0W+alTp8qqanl3XAK+DrKI4UJ4uDYsTa7XmimLkqmssc5zZoX5fbX
9CHMbL2l4bXf7X+VsDjf2CmN4vHUcxdptkb/67R427y/k38O7yYeb5yAi0Is3b4gl7XAJz36x0+g4wgL25Zfy8KepxIX58/jVnYyOYzOUUrw4rOEodurH28g5YdxRrF0W+bc7j/LeT1mM6x3GsLgQo+P
YnCs6t+SOS9uxZMshlqfkGB1HNKGK6jqmfJiMm4uJd26mx9XZLiuhyZw6iq2r09z5UQrVtfWG5LC7v7VDheU8U88Ukb5MCxAf+PDWks9FxbNb9V3agZ2RznlieRmZeqdFxRBN55os0MvJKeWtULK38PIyOY5M
iAr14ZUQMqYeK+M+q3YZksir66t567FKQDMHBuPm7Pjrty8UM4mJ2aMicPDxcSdi7Za7R4T4vx9tjWHpUk53HVZe/p3sN3LGlqDQd1aMbFvBHPXZxuyh7ldFfkrq3aTeqiIV0fE0Ka5p9FxbF5LX3f
eHBXL3ryTPPNFmtFxhBll5p3kieVp9IxsLqflmsljV0fTLcSPBz9JtfhZX3ZT5N/vOsac9dlM6BPOwK6tjI5jN/p3COLOS9uzNCmHFdsOGx1HmEFlTR13LkrBw9XE9NFxsujHTNycTcwcG4/WcPfiFGr
qLDdfbhd/g7lFFTz4aSpdWvvymHzqbnZTr4giMbwZTyxPY39Bmd6buUu9hw7yRs3dCfYz93oOHYlLMCTl6/LkMCTl6+PYduhIl7/1nL7/dt8kdfVa6Yu2UNbT1vj4136B0Nm4qzyYlpY+IwOSnuXmzcJ/P
iwn2z4wgfbTrIbf3bcmnHFkbHsUuDY1oxpmcY7/60j5/35lvkMW2+yGf8kMHjH0lYfRsexWyH+HrwyIoYdh4sN+2ReXJicE+U8smw73UP9eOCqjkbHsWtPD+lMh5be3L801/35lvkMW2+yGf8kMHm/YU8P6wrkYFeRsexWyH+HrwyIoYdh4sN+2ReXJicE+U8smw73UP9eOCqjkbHsWtPD+
0NRkMjwtheHyo0XHs3j+6BDOhTzhz12ezVvZjsSm1dfVMXbKNeg0zxsj54k3Nw9XEjDHxnKys4f6lTb8fi83+bRaVVzN1SQphzT15blhXo+M4jMeu7kR0sA8PLk0l72Sl0XHEWZq+JoOkAyd48bquhAX
IGV2W0DHYh6ev6cy6jALmrM9q0seyySLXWvPosh3kl1YxY0w83m7ORkdyGO4uJmaMiaOsupYHlqZaxc5v4q9tzDrO22szGZkQytBYWelsSWN7hjGwSzCvrt7DjpziJnscmyzyzsPsWrnUR7+RzTdQv2
MjuNwolr68NQQy4w0xIUpKq/mvo+3ER7gxbPXdjE6jsNRSvHy9BkdRTo9d0I9HbjniUNm/g1BbMUuVJqoFJqj1IqUyn1qDnu80wyjp3ku9krbhF8IpvyH0FPOjzPgkDXQFMVqM5wQVkhzAw2/IpWOTwZ8ySl8
MjuNwolr68NQQy4w0xIUpKq/mvo+3ER7gxbPXdjE6jsNRSvHy9d0I9HbjniUNm/g1BbMUuVJqoFJqj1IqUyn1qDnu80wyjp3kuZU7uTgqkEn9IpvyocRfODXSeGXVHrbnFBkdR/wJrTWPLNtOQWkV00f
H4SVHrobw93TlzVGx7D9exjNNtBHdBRe5UsoEzAQGAZ2BMUqpJtnkLmjrsXp+Dl6szrN8j+4kY6NdII8nHjnsUpcr1PK/TR5oOs3nmMh/7RUY5cDda7bQB3XdaeT5NzWN0ES/jNMSLvCWRqrbO01tX
AEmCoGe73D15ZtYfdR0/y2sjutPCRhQxGOzXSOFBYLlveWpmMYyd5fuUuLo4K5NZ+bY2OI4B7B0TxyMBoLo4KNPt9m6PIQ4BDp32d03jbbyilJiulkpRSSfn553eS8eS/NXdgnnoHx25LFoWMliL00caX6T
mGh1H8P9Hrp6uzrw+Uo5crYWzyYkpl7bD09X8U1wW+7BTaz1La52otU4MCjq/NdYSI5pz52XtzZxMXKh7BkQRF+bPE5/tkEvWYH/rNrdeOQaQwtfOXJ1BOYo8sNAm9O+Dm28TTgIF5MT00fHAXDvkhR
qLbhZkPittbvzeP+X/UzsG8Hl0S2NjiMsxBxFvgWIDAUkpFKqVcgdHAF2a4X2FD2jT35IXrurL1YBHT1hh/MVpHlFdSyYOfpBId7MOjg6KNjiMs6IKLXGtdC9wFrAbSgaVaa/nkywEnjQ1hREIob6/NZENW
+40bHcSj19Zr7l6ZSVl3LjDFxsnmcgzHLHLnW+mutdQetdTut9YvmuE9hm/51bRfCA3ogKJVLjTscm1zZKayXl5szM8bEcbysioeAXbUdrWcLf1FIOnuD1b/cwuFs
```

rRvdo8/c/IOyOFLkwu64hfjwyMJrvdh3jgw0HjI5j10oqa7hnSQotfd15aXg3lJJTDR2RFLloEpP6RXJ5dAte/CqdtMNNt1mQI9Na89hnO8gtqmT6mFj8PFyMjiQMIkUumoRSitdGdqeZlwt3yxL+JvH
R5oN8tf0ID1zVgYTw5kbHEQaSIhdNprmXK9NGx3HgeBlPfZ4m8+VmlH6khOe+bFiCf3v/dkbHEQaTIhdNqnfbAO4ZEMXylMN8kpRjdBy7UF5dy10fbcXXw4U3R8XKEnwhRS6a3t2XR3FR+wCeWpFG+pE
So+PYNK01TyxPI6ugjGmjYgn0djM6krACUuSiyZmcFG+NisPPw4U7F22V+fILsHjzIZanHGbqgA70bW/+XfSEbZIiFxYR5OPG9DFx7D9exmOf7ZD58vOQdriYZ7/YSf8OQdx9uWweJ/6fFLmwmN5tA3j
gqo58mZrLgl/3Gx3HphRX1DBlUTIB3q68JfPi4nekyIVFTbmkHVd0asELX6WzZX+h0XFsQn295oGl2zhSVMnbY+Np7uVqdCRhZaTIhUU5OSlevyGW0GYe3LFoK3kllUZHsnozfsjk+/Q8nhzciYTwZkb
HEVZIilxYnJ+HC++OT6C0spY7P9pKjexffkZr0o/x5vd7GR4fwoS+EUbHEVZKilwYIjrYl5ev78aW/Sd4fuUuo+NYpeyCMqZ+vI2uIb68dJ3soyLOzPwXjxPiLA2NDSHtcDGz12XTMdiHG3uFGx3JapR
U1jD5gyScnRTvjkuQ/cXFX5IRuTDUo4M6cWnHIJ5ZsVMuRtGotq6euz9KIbugjHduTCC0mafRkYSVkyIXhjI5KaaPiSM8wJM7FiVz8LhcvPnFr9P5aW8+LwzrSp92AUbHETZAilwYztfdhTkTelCv4ZY
FWygurzE6kmEWbTrA+7/sZ1K/SEb3DDM6jrARUuTCKkQGevHuuAQOHi9n8sIkqmrrjI5kcWv35PH0ip1c1jGIx6/uZHQcYUOkyIXV6NMugFdHxrApu5AHP9lOfb3jLONPPVTEHR9uJTrYh+lj4jDJyk1
xDuSsFWFVhsaGkFtUyX9W7aa1nzuPOcDIdH9BGbfM30Kgjyvv39wDH3e50o84N1LkwurcfklbcosqeO/nLPw9XZlyqf1eOCHvZCU3zduMBhbc3JMWPu5GRxI2SIpcWB2lFM9e24WSyhr+s2o3Xm4mbuo
TYXQsszteWsWNszdRUFrFolt70TbI2+hIwkZJkQurZHJquOZneXUdT6/YiaerMyMSQo2OZTZF5dWMm7uZg4XlzL+5J3FhsoeKOH/yYaewWi4mJ2aMiaNf+0Ae/jSV5Sn2cam4ksoabpq3mX15pcy+KVH
OFRcXTIpcWDV3FxOzbkqgd9sA7l+aykebDhod6YIcL61i7OyNpB8p4b/j4unfIcjoSMIOSJELq+fp6sy8iT24tEMQjy/fwdz12UZHOi9Hiiu44b0NZBwr5b3xCQzo1NLoSMJOSJELm+DuYuK98YkM6hr
M8yt38drqPTZ1nnl2QRkj/ruBvJIqFk7qxeXRUuLCfKTIhc1wdW6YMx+V2Ia312Zyz5IUKmusfwXohn3HGf7OL1TU1LF4cm96RjY3OpKwM3LWirApziYnXr6+G5FBXrz8zW4OF1Uw+6ZEAr3djI72pxZ
tOsAzK3YSHuDJ3Ak9iAj0MjqSsEMyIhc2RynF7Ze04783xpN+pIQh09ezKcu6tsCtrKnjqc/TeGJ5Gv2iAll+50VS4qLJSJELmzWoWys+vb0vHq4mxszeyIw1GdRZwbz5nqMnGTbzFxZuPMDk/m2ZO6E
HvrLsXjQhKXJh07qG+PHl3f24pntrXv9uLzfO2UhWfqkhWerrNfN/yeaat9dTUFrF+xN78PjVnWQDLNHklNaWH8EkJibqpKQkiz+usF9aaz5JzuH5lbuoqqlnyqXtmHJpO4tdIm3boSKe+WInqYeKuKx
jEK+M6E6Qj3XO2wvbpZRK1lon/v52+bBT2AWlFDcktuHSjkG8sDKdaWsyWLHtMFOv6MCQmFY4m5rm4PNIcQVvfLuXT5JzCPJx4/WR3RkeHyIXShYWJSNyYZfWZeTzwsp09hw7SWSgF3dc2o5rY1vj5my
eEfruoyXM+jmLL7blohTcclEkdw+IwttNxkai6ZxpRH5BRa6UGgk8C3QCemqtz6qdpciFJdTXa77ddYzpazLYdaQEX3dnBse0Znh8CAlhzXA6x7nrYyWVrEo7ytc7jrApuxAPFxOjerRhUr9I2jSXCyS
LptdURd4JqAfeAx6UIhfWSGvNuowClqccZlXaUSpq6vDzcCEuzJ+EsGZ0auVLoI8bAV6u+Lq7UF5TS1lVLUXlNew+epJdR0pIO1zM9pxiAKJaeDM0tjU39gqnmZerwc9OOJImmSPXWqc33vmF3I0QTUo
pRf8OQfTvEMQLw2r5Pv0YG/YdJ/nACX7ck/+3P+/r7kzn1r7cf2UHBnUNJqqljwVSC3H2LDahp5SaDEwGCAuTq4MLY3i5OTM0NoShsSEAFJfXkH28jMKyKgpKqzlZWYunqwkvN2d83J1pH+RNaDMPGaw
Iq/a3Ra6U+h4I/pM/ekJrveJsH0hrPQuYBQ1TK2edUIgm5OfpQqynv9ExhLggf1vkWusrLBFECCHE+ZGVnUIIYeMu9KyV64AZQBBQBGzTWv/jLH4uHzhwng8bCBSc58/aC3kN5DVw9OcPjvkahGut/3B
ZKUMWBF0IpVTSn51+40jkNZDXwNGfP8hrcDqZWhFCCBsnRS6EEDbOFot8ltEBrIC8BvIaOPrzB3kN/sfm5siFsBSl1E7gTq31j0ZnEeKv2OKIXIg/pZQaq5RKUkqVKqWOKKW+UUr1O9/701p3kRIXtkC
KXNgFpdT9wFvAS0BLIAx4BxhqZC4hLMGmilwpNVAptUcplamUetToPJaklGqjlFqrlNqllNqplLrX6ExGUUqZlFIpSqmVjV/7Ac/RMA3ymda6TGtdo7X+Umv9kFLKTSn1llIqt/HXW0opt8afDVRKrVR
KFSmlCpVS65RSTo1/tl8pdUXj759VSi1VSn2glDrZ+HeQeFqm1kqpZUqpfKVUtlLqniZ8/v5KqU+VUruVUulKqT5N9VjWSil1X+PfQZpSarFSyt3oTEaymSJXSpmAmcAgoDMwRinV2dhUFlULPKC17gz
0Bu50sOd/unuB9NO+7gO4A8vP8P1P0PCaxQLdgZ7Ak41/9gCQQ8OitpbA48CZPji6FlgC+ANfAG8DNBb/l0AqEAIMAKYqpf52cdx5mgas0lpH0/B80v/m++2KUioEuAdI1Fp3BUzAaGNTGctmipyGN1+
m1jpLa11NwxvKYQ6btdZHtNZbG39/koY3b4ixqSxPKRUKDAbmnHZzAFCgta49w4/dCDyntc7TWucD/wLGN/5ZDdCKhhVzNVrrdfrMZwCs11p/rbWuAxbSUKIAPYAgrfVzWutqrXUWMJsmKJfGo4/+wFy
AxscrMvfj2ABnwEMp5Qx4ArkG5zGULRV5CHDotK9zcMAiA1BKRQBxwCZjkxjiLeBhGi5ocspxILDxTf1nWvPbLSEONN4G8CqQCXyrlMr6mym7o6f9vhxwb3zMcKB14/RMkVKqiIaRfcuzfVLnIBLIB95
vnF6ao5TyaoLHsVpa68PAa8BB4AhQrLX+1thUxrKlIheAUsobWAZM1VqXGJ3HkpRSQ4A8rXXy7/5oA1AFDDvDj+bSULanhDXehtb6pNb6Aa11WxqmTu5XSg04x2iHgGyttf9pv3y01lef4/2cDWcgHvi
v1joOKAMc7fOiZjQcjUfS8B+yl1JqnLGpjGVLRX4YaHPa16GNtzkMpZQLDSW+SGv9mdF5DHARcK1Saj8NU2uXK6U+1FoXA08DM5VSw5RSnkopF6XUIKXUK8Bi4EmlVJBSKrDxez+Ehv8clFLtVcOVI4q
BOn472j8bm4GTSqlHlFIejR/GdlVK9TDLs/6tHCBHa33qaOxTGordkVxBw3+c+VrrGuAzoK/BmQxlS0W+BYhSSkUqpVxpmH/8wuBMFtNYNHOBdK31G0bnMYLW+jGtdajWOoKGv/8ftNbjGv/sdeB+Gj7
EzKdhlHwX8DnwApAEbAd2AFsbbwOIAr4HSmkY2b+jtV57jrnqgCE0fJiaTcOOfHMAv/N9rn/xWEeBQ0qpjo03DQB2mftxrNxBoHfjf9iKhtfAoT7w/T2bWtmplLqahjlSEzBPa/2iwZEspnFhyzoaiuj
UiPFxrfXXxqUyjlLqUhou+MbigIgAABwhSURBVD3E6CyWppSKpeE/ClcgC7hZa33C2FSWpZT6FzCKhrO5UoBbtdZVxqYyjk0VuRBCiD+ypakVIYQQf0KKXAghbJwUuRBC2LgzLaBoUoGBgToiIsKIhxZ
CCJuVnJxc8GfX7DRLkSul5tFw+lVe494HfykiIoKkpCRzPLQQQjgMpdSfXrTeXFMr84GBZrovIYQQ58AsI3Kt9c+N+380qfQjJeSfrMLf0wU/Dxeae7ni4+7S1A8rhM04UVZNfmkVZVW1VFTXARDo40a
Qtxt+Hi44OSmDE4qmYLE5cqXUZGAyQFhY2Hndx4cbD7Bo08Hf3BbW3JOYUD9i2/hzWXQL2gV5X3BWIWyB1podh4v5ftcxth8uJv1ICcdKzrwmxs3Zie6h/iRENKNHRDP6tgvE3cVkwcSiqZhtQVDjiHz
l2cyRJyYm6vOZI88tqiC3qIKi8hqKK2o4WlLJjpxitucUkVtcCUCX1r5c070118WF0NLXofeaF3YqM+8kizYdZHXaUXKLKzE5KaJaeNOplS+dWvnQys8DbzdnPF1N1GsoKK2ioLSKg4XlbD1wgp25JdT
Wa3zcnLm6Wyuuiw+hZ0RzGa3bAKVUstY68Q+321KR/5Xcogq+STvKl6m5bDtUhKvJiesTQritfzsiAh1ql09hh7TWbMouZPbPWazZnYersxP9o4IY2DWYAdEtaObletb3VVFdx5b9hazYlss3aUcor66
jQ0tv7hkQxdVdW0mhWzG7L/LT7S8oY876LJYm5VBbV8+w2BAeHRRNCxmhCxuUfqSE577cxYas4zT3cuWmPuGM7x1OgLfbBd93eXUt3+w4yn9/2kdmXilRLby5/8oODOwaTMN+VMKaNGmRK6UWA5cCgcA
x4Bmt9dwzfX9TF/kpeSWVzFmfzfxf9+NqcuK+KzswoU84ziZZByWs34myat74bi+LNh3A18OFqQOiGN0zrEnmtevqNV/vOML0NRlk5JVySYcgXhjWlTbNPc3+WOL8NfmI/FxYqshP2V9QxjNf7OSnvfl
EB/vw1uhYooN9Lfb4QpyrtbvzeOjTVE6U1zCuVxj3XdkBf8+znz45X3X1mg827Oe11Xuo05p7B3Rgcv+2mGS6xSo4dJFDwxzj6p3HeGpFGsUVNTw5uBPje4fL4aOwKhXVdbz0dToLNx4gOtiHN0fF0qm
V5QcdR4oreGbFTr7ddYzebZszbXScnDxgBRy+yE8pKK3iwU9S+XFPPld0aslrI2MsMtIR4u9k5ZcyeWEymXml3NovkocGdsTN2bjTA7XWfJqcw9MrduLpauLNUbH07/CH1eHCgs5U5A43WRzo7ca8CT1
4akhnftqbx3Xv/EpWfqnRsYSD+2lvPkNn/kJhWTUfTurFk0M6G1riAEopRia24Yu7LiLA25UJ729m5tpM5BoG1sfhihzAyUkxqV8kH/2zN8UVNVz3zq/8mllgdCzhgLTWzFmXxc3vbybE34MVd15Ev6h
Ao2P9RlRLH1bc2Y9rYlrz6uo9PLJsOzV153pZU9GUHLLIT+kR0ZwVd15ES183bpq3maVbDhkdSTiQ+nrNv77cxQtfpXNV52CWTelrtWeJeLiamDY6lnsub8/SpBwmvr+Z4ooao2OJRg5d5ABtmnuybEp
f+rQL4OFl25m7PtvoSMIB1NTV88Anqcz/dT+T+kXyzo3xeLkZsqv0WVNKcf9VHXltZHc2ZxcyetZGjpc67GUyrYrDFzmAj7sLcyYkMqhrMM+v3MX0NRkyDyiaTGVNHVM+TGZ5ymEevKoDTw7uZFOrKUc
khDJnQg+y8ksZNWsjeSWVRkdyeFLkjdycTcwYE8f18aG88d1eXv5mt5S5MLvKmjr++UESa3bn8fywrtx1eZRNngJ7SYcg5t/ck9yiCm54bwOHiyqMjuTQpMhP42xy4tURMYzvHc57P2fxxnd7jY4k7Eh
1bT13LNrKuowC/nN9w78zW9anXQALJ/XieGk1o2dt4GixjMyNIkX+O05Oin9d24VRiW2Y8UMmM9dmGh1J2IGaunru+mgrP+zO46XrunFDYhujI5lFQngzFt7ai8LSasbN3SRz5gaRIv8TTk6Kl4Z3Y2h
sw+lW8+QDUHEB6us1DyxN5dtdx/jXtV0Y2+v89uO3VrFt/Jk7sQeHCsu5ad5mSirlbBZLkyI/A5OT4vWR3RnYJZjnVu5ixbbDRkcSNkhrzfNf7eKL1FweGRjNhL4RRkdqEr3bBvDu+AT2HjvJpPlbqKy
pMzqSQ5Ei/wvOJiemjYmld9vmPPhJKr/IoiFxjmb9nMX7v+znlosiuf2StkbHaVKXdWzBW6PiSDpwgqlLtlFXLycLWIoU+d9wczbx3vhE2gZ6c/vCZNKPlBgdSdiI5Sk5/Pub3QyJacWTgzvZ5Nkp52p
wTCueGtyZVTuP8uJX6UbHcRhS5GfBz8OF92/ugZebMxPf30yunGol/sbGrOM89Ml2+rQN4PUbutvUeeIX6pZ+kdxyUSTzfsmWBXYWIkV+llr7ezD/lh6UVdVx64IkyqtrjY4krNT+gjJu/zCZ8ABP3h2
fYPjmV0Z4YnAnBnYJ5oWvdvHtzqNGx7F7UuTnIDrYl+ljYkk/WsIDS1OplzlA8TvFFTVMWrAFgLkTeuDn4WJwImOYnBRvjY4lJtSfqR9vY/dRmZJsSlLk5+jy6JY8PqgT36Qd5a3vZcGQ+H+1jeeKHyw
s591xCQ5/0W93FxOzxifg7ebMrQuS5BzzJiRFfh5uvTiSkQmhTP8hky9Tc42OI6zEy9/sZl1GAS8M60rvtgFGx7EKLX3dmXVTInknq5iyaCvVtbL9bVOQIj8PSilevK4bieHNePjT7XLYKFix7TBz1mc
zoU84o3rY14KfCxXbxp9Xro9hc3Yhz6/cZXQcuyRFfp5cnZ1458Z4fNyduW1hsuzN7MB25ZbwyLLt9IxozpNDOhsdxyoNiwthcv+2LNx4gGXJOUbHsTtS5Begha87/x0XT25RBfd9vE0+/HRAReXV3PZ
hEv4ersy8MR4Xk7ylzuThf3SkT9sAHl++g525xUbHsSvyr+4CJYQ35+khnflhdx7T1mQYHUdYUH29ZurH2zhWXMV/x8UT5ONmdCSr5mxyYsbYOJp5unL7h8kUlVcbHcluSJGbwbje4QyPD2H6Dxn8vDf
f6DjCQt75MZMf9+Tz1DWdiQtrZnQcmxDo7cY74+I5WlwpR7FmJEVuBkopXhzWjQ4tfJj68TaOFMvKT3v3a2YBb3y3l6GxrRlnZ7sZNrX4sGY8fU0X1u7J592f9xkdxy5IkZuJh6uJmTfGU1VTx90fpch
Vxu3YsZJK7lmSQtsgb166rptD7KFibuN6hTEkphWvrd7DpqzjRsexeVLkZtS+hTcvDe9G0oETvLp6j9FxRBOoravn7sUplFXV8V8buGCytVJK8e/h3QgP8OLuxSkUyGKhCyJFbmZDY0MY1zuMWT9n8cP
uY0bHEWY2fU0Gm7MLefG6rkS19DE6jk3zcXdh5th4iitqZL78AkmRN4EnB3emcytfHliaKvPlduSXzAJmrM1kZEIow+NDjY5jFzq39uVf13ZhXUYB//1J5svPlxR5E3B3MfH22Diqauu5d/E2amW+3Ob
ln6zi3iXbaBfkzb+GdjE6jl0Z1aMN13RvzRvf7SVpf6HRcWySFHkTaRvkzQvDurJ5fyHT5fxym1Zfr7l/6TZOVtbw9tg4PF1lXtyclFK8dF1XQvw9uGdxipxffh6kyJvQ8PhQRiSEMmNtJr/uk8vE2ap
Z67JYl1HAM9d0ITrY1+g4dsnH3YW3x8aRX1rFw59uR2uZLz8XUuRN7LmhXYgM9OK+j7dRWCYjDVuTcvAEr63ew+BurRjTs43RcexaTKg/jwyM5ttdx1i48YDRcWyKFHkT83R1ZvroOE6U1fDwp6ky0rA
hJZU13LMkhZa+7rw0XM4Xt4RbLork0o5BvPBVuuwqeg6kyC2ga4gfjw6K5vv0PD7YICMNW6C15snlaeQWVTJ9TKzDXunH0pycFK+N7I6vuwv3LE6horrO6Eg2QYrcQm6+KILLo1vw4tfppB+RkYa1W7b

```
1MF+k5nLvgCgSwpsbHcehBHq78cYN3dl7rJQXvpL9y8+GWYpcKTVQKbVHKZWplHrUHPdpb5RSvDoiBj8PF+6WkYZVyy4o4+kVafSKbM6dl7U3Oo5D6t8hiNv6t2XRpoOsSpOLN/+dCy5ypZQJmAkMAjo
DY5RSsrv+nwhoHGlk5slIw1pV19Zz75IUXExOvDkqFpOTzIsb5YGrOhIT6sejn22XhXV/wxwj8p5AptY6S2tdDSwBhprhfu3SxVEy0rBmr3+3h+05xfzn+hha+3sYHcehuTo7MW10HNW19Uxdso06WcJ
/RuYo8hDg0Glf5zTe9htKqclKqSSlVFJ+vmPv2f3AVR3pFiIjDWuzPqOA937KYmyvMAZ2DTY6jgAiA73417Vd2JRdyLuyhP+MLPZhp9Z6ltY6UWudGBQUZKmHtUquzk5MH9Mw0rjvYxlpWIPjpVXct3Q
b7Vt489RgmRm0JiMSQv+3hH/rwRNGx7FK5ijyw8DpKyVCG28Tf+HUSGNjlow0jKa15qFPt1NcUcOMMXF4uJqMjiROo5TihWFdCfZ1594lKZRUyoXOf88cRb4FiFJKRSqlXIHRwBdmuF+7JyMN67Dg1/3
8sDuPxwZF06mVLMG3Rn4eLkwfE0tuUSVPLk+ThXW/c8FFrrWuBe4CVgPpwFKt9c4LvV9HoJTixeu60srPnXsWy0jDCOlHSnjpm91cHt2CiX0jjI4j/kJCeHOmDojii9Rclm2Vg/7TmWWOXGv9tda6g9a
6ndb6RXPcp6PwdXdh2ug4jhTLSMPSyqtruXtxCn4eLrw6IkaW4NuAOy5rT6/I5jy9Io2s/FKj41gNWdlpBRLCm/1vpPFJco7RcRzGc1/uYl9+KW/eEEuAt5vRccRZMDkp3hwVi4vJiXuWpFBdK3v9gxS
51bjjsvb0btucZ1bsJDNPRhpN7cvUXJZsOcTtl7SjX1Sg0XHEOWjt78ErI2JIO1zCK6t2Gx3HKkiRWwmTk2La6IYzJu76aCuVNbKEv6kcKizn8c92EBfmz/1XdjA6jjgP/+gSzE19wpmzPluujYsUuVV
p6evOayNj2H30JP/+Ot3oOHappq6euxengILpo+NwMclbwFY9fnUnOjVeG/docaXRcQwl/4qtzOXRLZnUL5IFGw6wKu2I0XHszqur97DtUBEvD4+hTXNPo+OIC/Cba+MuSXHohXVS5FbokYHRdA/146F
Pt3OosNzoOHbjh93HmPVzFuN6hzE4ppXRcYQZtAvy5rmhXdmUXcg0B742rhS5FXJ1duLtsfEA3PXRVvlk3gxyiyq4f2kqnVv58qQswbcrIxJCuT4+lBk/ZLAuwzH3cZIit1Jtmnvy6ojupOYU8+9vZL7
8QtTU1XPP4hRqauuZeWM87i6yBN/ePD+sC+2DvJm6ZBvHShxvvlyK3IoN7BrMzRdF8P4v+/lmh8yXn69XVu0m6cAJXhrejchAL6PjiCbg6erMOzfGU15dx92LU6itc6yjWClyK/fYoE50b+PPQ59ul5V
s52FV2hFmr8tmfO9whsb+YXdlYUeiWvrw4nVd2ZxdyGvf7jU6jkVJkVs5V2cn3rkxHheTYsqHWymvrjU6ks3Iyi/lwU+2072NP08O6WR0HGEBw+NDGdMzjHd/2udQF26RIrcBIf4eTB8Tx968kzz22Q7
Zj+UsVFTXcceirbiYFO/cGI+bs8yLO4pnr+1M91A/Hvwk1WGOYqXIbcTFUUE8cGUHVmzLZcGv+42OY9W01jyybDt7jp3krdFxhMgl2xyKm7OJd8Yl4GJS3P5hMmVV9n8UK0VuQ+64tD1XdGrBC1+ls2H
fcaPjWK1ZP2fxRWouD17VkUs6OPbVqBxViL8HM8bEk5lXysPLttv9UawUuQ1xclK8MSqW8ABP7vxoqywW+hM/7c3nP6t2c3W3YO64tJ3RcYSB+kUF8vDAaL7afoSZazONjtOkpMhtjK+7C7NvSqSmrp7
JC5Plw8/T7C8o4+6PttKhpQ+vjugu+4sLbuvflmGxrXnt2718u9N+P/yUIrdBbYO8mTEmjj1HS3jok+3UO/AeE6cUV9QwacEWnJwUs8Yn4uXmbHQkYQWUUrx8fQwxoX7c9/E29hw9aXSkJiFFbqMu7di
CRwdF89WOI7z+3R6j4xiqpq6eOxYlc7CwnHfHJRAWIJthif/n7mJi1vhEPN2cmbRgC/knq4yOZHZS5Dbsnxe3ZUzPMGau3cfSLYeMjmMIrTVPfZ7GL5nH+ffwGHq3DTA6krBCwX7uzLkpkYLSKm79IIm
Kavva71+K3IYppXhuaBcujgrk8eU7WJ9RYHQki3vv5yyWbDnEXZe1Z0RCqNFxhBXr3saf6aPj2J5TxD12tu2tFLmNczE1rPxs38KbKR8msyu3xOhIFvPZ1hxe/mY3g2NayZV+xFm5qkswzwzpzHe7jvH
8yl12c1qiFLkd8HF3Yd7EHni7O3PTvM3sLygzOlKT+2H3MR76dDt92wXwxg3dcXKSM1TE2Zl4USS39otk/q/7eefHfUbHMQspcjvR2t+DhZN6Uldfz7i5m+z60lfJBwq5Y9FWOrXy4b3xCbL8Xpyzx6/
uxHVxIby6eo9drJSWIrcj7Vv4sOCWnpwoq+ameZs4UVZtdCSzSztczC3zk2jl58H8m3vi4+5idCRhg5ycFK+OiOHKzi155oudLEvOMTrSBZEitzMxof7MnpDI/uPl3DhnE4V2VOZph4u5cc4mvN2c+eC
WngR6uxkdSdgwZ5MTM8bEcVH7AB5etp2vttvunv9S5Haob7tAZt+UyL78UsbO3sjxUts/b3ZnbjHj5jaU+JLJveXCycIsTp1jHh/mz92Lt7I8xTZH5lLkduqSDkHMndCD7IIyxszeaNOLIHbkNIzEPV1
MLP6nlLgwLy83Z+bf3JNekQHcvzTVJtdkSJHbsX5Rgbw/sQcHC8u54b0NHDhue2ez/Lw3n1GzNuDl6sySyX1k1aZoEl5uzsyb2IN+7QN5eNl25v+SbXSkcyJFbuf6tg9k0a29OFFezfB3fiX1UJHRkc7
a5ymHuWX+FsIDvFh+R18pcdGkPFxNzL4pkSs7t+TZL3fxwspdNrOPkRS5A0gIb86yKX3xcDUxetZG1qQfMzrSX9JaM3NtJlM/3kZiRDM+vq03LXzdjY4lHIC7i4l3xyUwsW8Ec9Znc8eirTaxnF+K3EG
0C/Lmszv60r6FN7d+kMS07zOscrRRWlXLlA+38urqPVzbvTXzb+6Jr5xiKCzI5KR49touPDWkM6t3HWXUrA1Wv/e/FLkDaeHjzse39WZYbAhvfr+XWxZsoajcek5P3JdfyrCZv/Bd+jGeHNyJaaNjcXe
RxT7CGJP6RfLeuASy88sYPH2dVe9nLkXuYDxdnXnjhu68MKwrv2YeZ/D09fySaexmW/X1mgW/7mfI9PUUllWzcFJPbr24rVwYQhjuqi7BrLynH2EBnkxemMzzK3dRWWN9Uy3KiE1jEhMTdVJSksUfV/z
WtkNF3PfxtoZTFHu24bGrOl18GuNQYTmPLNvOr/uO079DEP+5vhut/ORiycK6VNXW8eJX6Xyw4QCRgV68dF03+rSz/JbJSqlkrXXiH26XIndslTV1vPndXmavy6KFjzuPDOrItd1DMDXxJlRlVbXM+jm
L2euyUMCTQzozukcbGYULq7Y+o4DHl+9oOKU3MZSH/hFNkI/lVhhLkYu/lHqoiMeX72BnbgnRwT48eFVHBnRqYfZiraqtY1nyYd78fi/5J6sY3K0Vjw6KlkU+wmZUVNcxbU0Gs9dl4WJSTOgTweT+bQm
wwJYRTVLkSqmRwLNAJ6Cn1vqs2lmK3DrV12u+2nGEN77bS3ZBGdHBPoztFcbQ2BD8PC5syuVIcQUfbTrI4s0HKSitJjG8GY8P7kR8WDMzpRfCsrILypixJoPPtx3G3cXEiIRQRia0oWuIb5MdWTZVkXc
C6oH3gAelyO1DTV09y7ceZsGG/ezMLcHdxYkrOwdzcftA+rYPILTZ34+etdZk5pXy45581u7JY1N2IfVaMyC6BTf1ieDiqECZRhF2ITOvlJlrM/lqxxGqa+uJDvZhSEwrerUNICbUz6zbLDfp1IpS6ke
kyO3SjpxiPtp8kO92HaOgcfOtEH8PwgM8adPMk1b+7piUol5DXX09h4sqyS4oJaugjKLyGgA6tPTmik4tGdMzTKZQhN0qLq/hy+25fJKc878V1K7OTnRp7Utrfw+Cfd1p6evGoK6tzvt9YHiRK6UmA5M
BwsLCEg4cOHDBjyssR2vN3mOl/JJZQMqhIg4VlpNzouJ/5X5KsK87kYFeRAZ50aW1L5d2bEGIv5yFIhxLYVk1SfsL2ZxdSFpuMcdKqjhaXElFTR0fTupFv6jA87rf8y5ypdT3QPCf/NETWusVjd/zIzI
id0g1dfUAmJRCKWS6RIgz0FpTWlWLq7PTeU+3nKnInc/iwa84r0cUDsHFJGvKhDgbSqkmu6KVvAuFEMLGXVCRK6WuU0rlAH2Ar5RSq80TSwghxNkyZEGQUiofON9POwMBYzcHMZ68BvIaOPrzB8d8DcK
11kG/v9GQIr8QSqmkP5vsdyTyGshr4OjPH+Q1OJ3MkQshhI2TIhdCCBtni0U+y+gAVkBeA3kNHP35g7wG/2Nzc+RCCCF+yxZH5EIIIU4jRS6EEDbOpopcKTVQKbVHKZWplHrU6DyWpJRqo5Raq5TapZT
aqZS61+hMRlFKmZRSKUqplUZnMYJSyl8p9alSardSKl0p1cfoTJamlLqv8X2QppRarJRyNzqTkWymyJVSJmAmMAjoDIxRSnU2NpVF1QIPaK07A72BOx3s+Z/uXiDd6BAGmgas0lpHA91xsNdCKRUC3AM
kaq27AiZgtLGpjGUzRQ70BDK11lla62pgCTDU4EwWo7U+orXe2vj7kzS8eUOMTWV5SqlQYDAwx+gsRlBK+QH9gbkAWutqrXWRsakM4Qx4KKWcAU8g1+A8hrKlIg8BDp32dQ4OWGQASqkIIA7YZGwSQ7w
FPEzDlakcUSSQD7zfOL00RynlZXQoS9JaHwZeAw4CR4BirfW3xqYyli0VuQCUUt7AMmCq1rrE6DyWpJQaAuRprZONzmIgZyAe+K/WOg4oAxzt86JmNByRRwKtAS+l1Dhj
pFxpKfJHW+jOj8xjgIuBapdR+GqbWLldKfWhsJIvLAXK01qeOxj6lodgdybXAfq1107XWNrhZiBEZFKqUillCsNH258YXAmi1ENl96ZC6Rrrd8wOo8RtNaPaa1DtdYRNPz9/6C1dqiRmNb
6KHBIKdWx8aYBwC4DIxnhINBbKeXZ+L4YgIN94FrKbhU+p5WuudBseypIuA8cAOpdS2xtse11p/bWAmYYy7gUWNA5os4GaD81iU1nqTUupTYCsNZ3Ol4ODL9WWJvhBC2Dhbm1oRQgjxO1LkQghh46Gb
RQgjxJ6TIhRDCxkmRCyGEjZMiF0IIGydFLoQQNk6KXAghbJwUuRBC2Lj/AyMO7/scLs1PAAAAAElFTkSuQmCC\n",
      "text/plain": [
       "<Figure size 432x288 with 2 Axes>"
      ]
     },
     "metadata": {
      "tags": [],
      "needs_background": "light"
     }
    }
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "gLtsST5SL9jc"
   },
   "source": [
    "You can read much more about the `subplot` function in the [documentation]
```

```
(http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.subplot)."
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "CurjWdZgKA-x"
      },
      "source": [
        "# Torch tensor\n",
        "\n",
        "The pytorch tensor class involves two attribute variables (*i.e*., two data containers), including one with the elements of the tensor,
analogous to a numpy multidimensional array, and the other container with the gradients of an input function with respect to the tensor elements. In
addition, the tensor class involves the attribute 'backward' method for the so-called *backward propagation* that computes the gradients of input
function with respect to the elements of the tensor."
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "id": "zjZCZgpHAfk9"
      },
      "source": [
        "import torch\n",
        "import torch.nn as nn\n",
        "import torch.nn.functional as F\n",
        "import torch.optim as optim"
      ],
      "execution_count": 102,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "iZaXW6Tqgcdi"
      },
      "source": [
        "As an example, we consider the following numpy multiarray "
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "id": "3LTZ9rV_ftmJ"
      },
      "source": [
        "nt=np.ones((2,2))"
      ],
      "execution_count": 103,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "Wh50R26-v-JU"
```

```
      },
      "source": [
        "which can be used to build a corresponding torch tensor, with associated gradients, as follows:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "id": "BcYSoJuiwQbn"
      },
      "source": [
        "r = torch.tensor(nt, requires_grad=True)"
      ],
      "execution_count": 104,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "YO6yt3CjwkW2"
      },
      "source": [
        "The resulting torch tensor can be combined with other torch tensors to define, for example, a function f as follows:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "d1lJ5ZLSYa9M",
        "outputId": "3ac98a09-1397-433a-a0dd-fd2b23bcb65f"
      },
      "source": [
        "p = torch.ones((2,2), requires_grad=True)\n",
        "p2 = p+p\n",
        "y=(r+2)+p2\n",
        "z=y*y*3\n",
        "f = z.mean()\n",
        "print(\"r=\",r)\n",
        "print(\"p=\",p)\n",
        "print(\"f=\",f)\n",
        "\n",
        "print('before backward: r.grad=', r.grad)"
      ],
      "execution_count": 105,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "r= tensor([[1., 1.],\n",
            "        [1., 1.]], dtype=torch.float64, requires_grad=True)\n",
            "p= tensor([[1., 1.],\n",
            "        [1., 1.]], requires_grad=True)\n",
```

```
      "f= tensor(75., dtype=torch.float64, grad_fn=<MeanBackward0>)\n",
      "before backward: r.grad= None\n"
     ],
     "name": "stdout"
    }
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "dJpuIfuOf0RO"
   },
   "source": [
    "Note that the torch tensor r does not have any gradients (i.e.,``before backward: r.grad= None``) since so far we have not invoked the method
``backward`` for any function of r. "
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "eZbzKI5UzaUJ"
   },
   "source": [
    "Now we can compute the gradient of f with respect to the elments of r by instantiating the ``backward`` attribute of f, as follows:"
   ]
  },
  {
   "cell_type": "code",
   "metadata": {
    "id": "xKsCdcdZZGHU"
   },
   "source": [
    "f.backward()"
   ],
   "execution_count": 106,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "iX5w4G64zhIS"
   },
   "source": [
    "We can now check that the tensor r has the correct gradients of f with respect to the 4 elements of r, as follows:"
   ]
  },
  {
   "cell_type": "code",
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "pY16cKzusTnx",
    "outputId": "4d7516c9-27f0-4ced-f51b-53f37ff9862d"
   },
```

```
      "source": [
        "print('after backward: r.grad=', r.grad)"
      ],
      "execution_count": 107,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "after backward: r.grad= tensor([[7.5000, 7.5000],\n",
            "        [7.5000, 7.5000]], dtype=torch.float64)\n"
          ],
          "name": "stdout"
        }
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "4kR3UhE4z8fs"
      },
      "source": [
        "We can also zero the gradients, as follows:"
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "UmBMTcoI0Bsa",
        "outputId": "5b4373d7-4716-43f1-d704-b918613f211a"
      },
      "source": [
        "r.grad.data.zero_()\n",
        "print('after zero: r.grad=', r.grad)"
      ],
      "execution_count": 108,
      "outputs": [
        {
          "output_type": "stream",
          "text": [
            "after zero: r.grad= tensor([[0., 0.],\n",
            "        [0., 0.]], dtype=torch.float64)\n"
          ],
          "name": "stdout"
        }
      ]
    }
  ]
}
```