```
{
 "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)\u001b[0m \u001b[0;31m# KeyError: 'monkey' not a key of d\u001b[0m\n",
          "\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": "E3CDH3bg2KLl"
   },
   "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": "l3FnmoSeL9iu",
      "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":
"iVBORw0KGgoAAAANSUhEUgAAAYIAAAD4CAYAAADhNOGaAAAABHNCSVQICAgIfAhkiAA
AAAlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW
9uMy4yLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy+WH4yJAAAgAElEQVR4nO3dd3hU95X4//cZ
VVRBSEKNJnoRCBDNdhIXjCk2uAdXkjixs4m9qd442d0461+cZLPZ1HWKY8d2bMfYwdh0Y9wL
YBBFEh0hiioSAoSEunR+f2jIV8GiajR3ynk9zzzyauWXukRjm3Hvup4iqYowxJni5nA7AGGOMsyw
RGGNMkLNEYIwxQc4SgTHGBDILBMYYE+RCnQ7gUiQmJuqqgQYOcDsMYY/zK5s2bj6pq0pnL
/TIRDBo0iNzcXKfDMMYYvyIh7pabUhYDEmyFkiMMaYIGeJwBhjgpwIAmOMCXIeSQ
Qi8hcRqRSR7WdZLYyWxEpFJF8EZnYad1CEdnnfiz0RDzGGGMunKeuCJ4FZp1j/WxgmPtxP/
AHABFJAB4FpgJTGEdFpI+HYjLGGGHMBPNKPQFU/EJFB59hkPvBX7hIL0BaRVOBKYK2q
HgMQkbV0JJSXPXPBXoGlubWfjgWNU1jZysqGF2sZWUnv3YurgBDL69EJEnA7RGJ9x5GQjH+4
7yqmmVtraIXZVhiTHMD2zL5FhIU6H51O81aEsHSju9LrEvexsyz9FRO6n42qCAQMG9EyUxh
VyS+pYcmWEpbIIXG8vqXL7VLiIpkxOmHrh5Gv7hIL0dpjG+oqW/hpU2HeWN7BduKT3S5TWS
Yi8uHJDIvO40bxqXhctkJIN/0LFbVJ4EnAXJycoJiNp3K2kb+e=47XtvLnzCBGhLq4d3Y8bs9MZkhx
DXGQoMZGhHDxaz8aDx9hQVM3Lm4pZvLmEL1+RyQOfyyQ2MszpX8EYr1BVXttayuMrd1F9q

pms9Hi+O3M4M0b3IzEmghD31XJeyQne3V3J27sr+caibfx1/SEemz+GMWnxDv8GzhJPzVDmL
g2tUNWxXaz7E/Ceqr7kfr2HjrLQlcCVqvpAV9udTU5OjgbyEBOqyrK8Mh5dtoP65ja+OWMYd08b
SNx5vtgPV9fzizf3sCyvjMSYcP5w9yQmD0rwUtTGOOPg0VN8f0kB64uqmTCgNz++cex5v9jb25
VXt5Tws9W7OV7fzD3TBvKDuaOICA3skpGIbFbVnE8t91IimAs8CMyh48bwb1V1ivtm8WbgdCui
LcCk0/cMziaQE0Fbu/KDJQW8nFtMdv/e/OK28QxNjrmo9ygoqeEbi7ZScryBn96cxS2TMnooWm
Octa34BF98ZiNt7cr3Zo/kjskDLqrUU9PQwq/W7uXZdQeZlpnAn+7JIb5X4F5J92giEJGX6Di7TwS
O0NESKAxAVf8oHXcx/4+OG8H1wBdVNde975eAH7jf6nFVfeZ8xwvURNDS1s63Xt7Givxyvn7V
EL41YzihIZfWsKumvoV/eXEz6/ZX87Urh/DdmSOsFmoCygd7q/jqC5tJjIng+fumMLBv9CW/1+tb
S3l4cR6ZiTE8+6XJpMb38mCkvqPHrwi8KRATQVNrGw/+bStrdx7h+7NH8sDnhnT7PVva2vnh0h
28tPEwX7p8MD+8YbQHIjXGecvzyvj2K9sYmhzLc1+cTLIHGkh8tO8oX31hM7GRobz0lWkMSrz
0xOKrzpYIrGexD2hrV772whbW7jzCf80b45EkABAW4uInN43li5cP4i8fH+CpD4s88r7GOGnd/qN
86+VtTOjfh0X3T/NIEgC4YlgiLz8wjcaWNr707CZqztJCLxBZIvABP1+zm7d3V/LY/DEsvGyQR99b
RPiPuaOZPTaFH6/cxYr8Mo++vzHedODoKf7lhS0MSozmqS94vp4/Ji2eP92TQ/Hxer76wmaaW9
s9+v6+yhKBw5bnlfGn94u4a+oA7p0+qEeOEeISfvX5bCYP6sO3X85j08Fz3os3xifVNLRw33Obc
Ak8vTDnvK3oLtWUwQn89y3jWF9UzX++vh1/LJ9fLEsEDtpZdpJ/W5xPzsA+PHrDmB49VmRYC
H++N4f0Pr146G9bOVHf3KPHM8aT2tqVh17ayuHqev5w96Ru3Ri+EDdPzOChq4fycm4xz6472K
PH8gWWCBxysrGFB17IJa5XKL+/eyLhoT3/T9E7KpzfLpjA0bomvr+kICjOdExgePqjIj7YW8Vj88c
yLbOvV475rRnDmTEqmZ+u3s2+I7VeOaZTLBE45KerdlN6vIHf3zWJ5FjvDQmRlRHPd2aOYPX
2Cv6+ucRrxzXmUu09Ussv1uxl5uh+3DGlv9eO63IJP7tlHLERoXzrlW20tAXu/QJLBA74uPAoL20
8zFc+k8mkgd4fbPX+z2YyLTOBHy3bwcGjp7x+fGMuVEtbO995JY+YyFAevynL6wMrJsZE8PhN
WWwvPcnv3t7n1WN7kyUCLzvV1Mr3Xs0nMzGab1073JEYQlzCL2/PJtQlfOfvebS3W4nI+KY/vL
efgtIafnzjWJJiIxyJYdbYFG6ZmMET7+1n6+HjjsTQ0ywReNnP39hN6YkGfn7rOEeHwk3r3Yv/vH4
0mw8dZ/EWKxEZ37Oz7CS/fXsf88anMScr1dFYHp03mpS4SB5enB+QJSJLBF60+dAxnlt/iIXTB
5HjA4PB3TIxg4kDevPfq3cHVecZ4/tUlR8t30FcrzD+a17Ptqi7EHGRYfxo3hgKK+t4fv0hp8PxOEs
EXtLerjy2fCcpcZH826wRTocDdNwMe2z+WI7XN/O/a/c4HY4x/7CqoIKNB47xnZnD6RMd7nQ4A
MwYlcxnhiXyq7f2Ul3X5HQ4HmWJwEuW5pWSV1LDw9eNICrcd6aBGJsez93TBvLChkPsKKtx
OhxjaGxp4yerdjEyJZYFk31nEioR4dEbRtPQ3MYv3gysEydLBF7Q0NzGz9/YQ1Z6PDdN6HICNk
d959oR9IkK54dLd1jfAuO4P39QROmJBh69YQwhPjZi7tDkWBZeNohFm4opKAmcEydLBF7w1I
dFlNc08h9zR/nkUNDxUWE8fN0INh86zpodR5wOxwSx8poGfv/efmaPTWH6EO90HLtY/3rNMBK
iwvnR8sA5cbJE0MOOnGzkD+/vZ9aYFKZ6qUfkpbh1UgaZidH8cu0e2qw5qXHI/765lzZVfjBnlNO
hnFV8rzC+M7PjxOntXZVOh+MRlgh62G/e3kdLWzuPzB7pdCjnFBri4tszh7P3SB3L8kqdDscEoQ
NHT7FkSwn3TBtI/4Qop8M5p9tyMhiQEMUv1+4NiH44HkkEIjJLRPaISKGIPNLF+I+JyDb3Y6+In
Oi0rq3TumWeiMdXlJ5o4O+5xXx+cn+/mORizthURqXG8au1+wKyrbTxbb97ex/hoS6+6qH5OHp
SWIiLb1wzjJ3lJ1mzo8LpcLqt24lAREKAJ4DZwGjgDhH5p6mwVPVbqpqtqtnA74AlnVY3nF6nqv
O6G48v+f27hQD8y5VDHY7kwrhcwsPXDefwsXpeyS12OhwTRPZX1fH6tlLunT7IsR7EF+vGCel
kJkXzq7f2+n051RNXBFOAQlUtUtVmYBEw/xzb3wG85IHj+rSyEw28klvMbTn9Se/tP/OfXjUimUk
D+/C7twtpbGlzOhwTJH779j4iw0J44LOZTodywUJcwjdndJRTVxaUOx1Ot3giEaQDnU8fS9zLPk
VEBgKDgXc6LY4UkVwR2SAiN57tICJyv3u73KqqKg+E3bP+8N5+AL52pe9f5nYmInx35ggqTjba
VYHxin1HalmWV8a90wfRN8Y/rgZOuz4rlRH9Yvn1W3tp9eNyqrdvFi8AFqtq51PNge7JlO8Efi0iX
X5zquqTqpqjqjlJSUneiPWSldc08PKmYm6dlEFGH9++6dWVaZkJTBzQmyc/KPLrD7fxD795ex9
RYSHc70dXA6e5XMK3rh1GUdUpv74q8EQiKAU6DxKe4V7WlQWcURZS1VL3zyLgPWCCB2Jy
1J/eL6Jdla/5yb2BM4kIX/3cEEqON/j1h9v4vkPVp1hVUM490weR4CNDSVysmaNTGJIUzZMfFPl
tvwJPJlJNwDARGSwi4XR82X+q9Y+IjAT6AOs7LesjlhHu54nA5cBOD8TkmGOnmlm06TA3TUj3
+SZw5zJjVD+GJsfwx/f998NtfN/THx0gxCV86fJBTodyyVwu4f7PZrKj7CQfF1Y7Hc4l6XYiUNVW4

EFgDbALeEVVd4jIYyLSuRXQAmCR/vO3yiggV0TygHeBn6mqXyeCFzccorGlna/44WVuZy6X8
MBnM9lVfpL39/r+PRnjf46fauaV3GJuzE4nOc57s/T1hBsnpJMUG8GfPtjvdCiXxCOjn6nqKmDVG
ct+eMbrH3Wx3zogyxMx+ILGljaeW3+Izw1PYni/WKfD6bb52en8cu1e/vj+fq4ckex0OCbAPB8gJ0
0AEaEhfOGyQfzPmj3sKKthTFq80yFdFOtZ7EHL8so4WtfEVz7j/x9sgPBQF/ddMZgNRccCdmY
m44zGljaeW3eQq0YExkkTwN1TBxIdHsKfPyhyOpSLZonAQ1SVpz88wMiUWC4f6rtjCl2sBVMG
EBcZytMfHXA6FBNAlmwppfpUM/d/1r+aV59LfFQYC6YMYHl+OSXH650O56JYlvCQD/YdZc+R
Wr78mUyvT7Ddk2liQrk9pz9vbK/gyMlGp8MxAaC9XXnqwyKy0uOZlun8TH2e9KUrBiPAc+sOOh
3KRbFE4CFPfVhEcmwE88anOR2Kx907fRBtqry4lfCm6DPe98G+KoqOnuLLnxkcUCdNAOm9e
3HdmBReyS2hodl/euZbIvCAwspaPtx3lHunDyQ8NPD+pAP6RnH1iGT+tvEwTa3+8+E2vun59Yd
ljIlg9lhnJ6TvKfdOH0hNQ4tfjeIbeN9aDnhhw2HCQoQFU3xnWj1PW3jZII7WNbPKOpiZbig+Vs87
eyq5Y0r/gDxpApgyOIER/WJ5bt0hv+mDE5j/El5U39zKq5tLmJOVSqKfjZNyMa4YmkhmUjTPrrPy
kLl0L35yGJcId04N3JMmEeGe6QPZWX6SLX7S2s4SQTct21ZGbVMrd08b6HQoPcrlEhZOH0Re
8Qm2FZ84/w7GnKGxpY2XNx1mxqhkUuP9Z0TeS3HThHRiI0L563r/OHGyRNANqsrzGw4xMiW
WnIF9nA6nx90yKYOYiFD+6mctIoxvWJlfzvH6Fu6dPsjpUHpcdEQot0zKYFVBOVW1TU6Hc16W
CLphW/EJdpSd5K5pAwOu9UNXYiJCuWlCOisKyjlR3+x0OMbP/HXDITKTornMRyel97R7pg+kp
U1ZtPGw06GclyWCbnhhw2Giw0O4aUKX0y8EpAVT+tPc2s5rW/2nRYRxXkFJDXnFJ7gnSE6aA
lYkxXDF0EQWbSr2+RnMLBFcouOnmlmeX8ZNE9OJifDIkE1+YUxaPOMy4lm0sdhvWkQY5y3a
dJiIUBc3T8xwOhSvWjClP6UnGvio8KjToZyTJYJL9NrWUppb27lramDfJO7KgskD2HOklq1209hc
gIbmNpZtK2NuVirxvcKcDserrh3djz5RYby8ybfLQ5YILoGq8kpuMeMz4hmVGud0OF43LzuNqPA
Qv6h9GuetKiintqmV2yf3P//GASYiNISbJ2awducRqut896axJYJLUFBaw+6KWm7LCb4PNnTcNL
5hXBrL88qpbWxxOhzj417eVMygvlFMHRxY4wpdqM9P7k9Lm/r0fTWPJAIRmSUie0SkUEQe6W
L9F0SkSkS2uR9f7rRuoYjscz8WeiKenvZKbjERoS7mZQfeuEIXasGU/jS0tLEsr8zpUIwPK6qqY+
PBY9w+uX/Q3CQ+0/B+sUwc0JtFm3z3vlq3E4GIhABPALOB0cAdIjK6i01fVtVs9+Mp974JwKPA
VGAK8KiI+HSD/MaWNpZuK2NOVipxkcFV7+wsu39vRqbEsmhjsdOhGB/2Sm4JIS7h1iC7SXym
BZMHUFhZ57M9jT1xRTAFKFTVlIVtBhYB8y9w3+uAtap6TFWPA2uBWR6Iqce8sb2C2sZWbg/S
stBpIsKCyf0pKK1hV/lJp8MxPqilrZ3Fm0u4akSy309F2V1zx6USHR7isydOnkgE6UDn367EvexMt
4hIvogsFpHT36IXuq/PeHlTMQMSgrfe2dm87HTCQoRXN5c4HYrxQe/uruRoXRMLgvAm8ZmiI0
KZl53Givxy6ppanQ7nU7x1s3g5MEhVx9Fx1v/cxb6BiNwvIrkiklV5cxk6oer61lfVM3tORm4XMFZ
7+wsITqcq0Yk8/q2Mlrb2p0Ox/iYxZtLSIqN4MoRSU6H4hNunZRBQ0sbb2yvcDqUT/FEligFOqf8
DPeyf1DValU93XbqKWDShe7b6T2eVNUcVc1JSnLmg7V4SwkiHWPumA63TMrgaF0TH+xzJjk
b33TsVDPv7qnkxuw0QkOscSLAxAF9GNg3iiVbfO8K2hP/QpuAYSIyWETCgQXAss4biEjnGSjm
Abvcz9cAM0Wkj/sm8Uz3Mp+jqry2tYQrhiYG/MiJF+OqEcn0iQrj1c2+2zTOeN+K/DJa2jToehKfi4h
w84QM1hdVU3qiwelw/km3E4GqtgIP0vEFvgt4RVV3iMhjIjLPvdm/isgOEckD/hX4gnvfY8D/R0cy2
QQ85l7mc3lPHaf4WENQjSt0IcJDXczPTmftziPU1FufAtPh1S2ljEqNC8oOl+dy88R0VOF1H+tT4
JFrNlVdparDVXWIqj7uXvZDVV3mfv59VR2jquNV9SpV3d1p37+o6lD34xlPxNMTlmwppVdYCNe
NSXE6FJ9z66QMmtvaWZ5vfQoMFFbWkVd8glsm2knTmfonRDFlcAKvbinxqT4FVry7AI0tbazIL2
PW2BSig2iAuQs1Ji2OEf1iWWythwzw2tYSXEJQd7g8l1smplNUdcqnJniyRHAB3tldSW1jq5WFz
kJEuGVSOtuKT7C/qs7pclyD2tuV17aU8tnhSSTHBnffgbOZk5VKRKiLJVt8pzxkieACLNlSSnJsB
JcPTXQ6FJ91Y3Y6IrDUx2qfxrs2HKimrKbRbhKfQ2xkGNeNSWF5fhlNrW1OhwNYIjivY6eaeW9
PJfOz0wixvgNnlRwXyWVD+rI0r8ynap/Gu5ZsKSU2IpSZo/s5HYpPu3liOifqW3hvj280u7ZEcB4r8
stobbdmcBdifnY6h6rrfar2abyn0d1ZatbYFCLDQpwOx6ddMTSRvtHhLNvmGw0sLBGcx2tbSxmZ
EmvN4C7ArLEphIe6WOojH27jXe/srqSuqZUb7V7aeYWGuLh+XCpv7TriE0O5WyI4h8PV9Ww9f
lL52fbBvhBxkWFcMzK54yrKhpwIOku3lZIUG8G0zOCYnL675mWn09Tazps7jjgdiiWCczndLv6G
8ann2dKcNj87naN1zT4/R6vxrJqGFt7dU8UN4+xe2oWaOKA3GX16sdQH5vSwRHAOS7eVkjO

wDxl9opwOxW9cNTKJuMhQn6l9Gu9Ys6OC5tZ26ztwEUSE+dlpfLSviqpaZ6extERwFrsrTrL3SJ
19sC9SRGgIc7JSWbOjgoZm32gaZ3resm1IDOwbxfiMeKdD8Svzs9NpV1jpcK98SwRnsWxbGS
EuYU6WlYUu1rzsNE41t7F2l/O1T9PzKk82sm7/UeaPTwva6Sgv1fB+HQ1RnC4PWSLogqqyLK+
My4cmkhgT4XQ4fmfa4L70i4tguQ/UPk3PW5FfTrvakBKXan52GlsPn+Bwdb1jMVgi6MKWwycoO
d7A/PH2wb4ULpcwNyuN9/dUcdIHmsaZnrU0r4wxaXEMTY51OhS/dIP7e2ZZnnO98i0RdGF5Xh
kRoS5mjrHekZfq+vGpNLf5RtM403MOV9eTV3yCeXbSdMnSe/di0sA+rMgvdywGSwRnaG1rZ0V
+OVePTCY2MszpcPzWhP69Se/dixU2NHVAO93Eeu44u5fWHTeMS2V3RS2FlbWOHN8SwRk2
HjjG0bqmf1yumUsjltwwPo2P9h3l+Klmp8MxPWRFfjkTBvS2JtbdNCcrFRFYnufMVYFHEoGIzBK
RPSJSKCKPdLH+2yKyU0TyReRtERnYaV2biGxzP5adua+3Lc8vJyo8hKtGJDsdit+7flwqre3KGzt
8b7Ju0337q+rYVX6S68fZSVN3JcdFMnVwAivynRm0sduJQERCgCeA2cBo4A4RGX3GZluBHF
UdBywGft5pXYOqZrsf83BQa1s7b2wvZ8aofvQKt0GzumtMWhyZidHWeihArcwvRwTmWhNrj7h
+XBr7q06xu8L75SFPXBFMAQpVtUhVm4FFwPzOG6jqu6p6um3UBsAnh/Jct7+a4/UtVu/0EBHh
+nGpbCiqprK20elwjIetyC9j8sAEUuJtAhpPmD02hRCXOHLi5IlEkA4Ud3pd4l52NvcBqzu9jhSRX
BHZICI3nm0nEbnfvV1uVVXPjOG9Mr+cmIhQPjc8qUfePxjdMD6NdoXVBVYeCiR7j9Sy90gd19s
4XB7TNyaCy4b0ZUV+udfLQ169WSwidwM5wP90WjxQVXOAO4Ffi8iQrvZV1SdVNUdVc5KSP
P9F3dzazhs7Krh2dD8bS92DhvWLZUS/WGs9FGBW5JXhEpg91hKBJ10/LpXDx+opKK3x6nE9
kQhKgf6dXme4l/0TEZkB/DswT1X/McKSqpa6fxYB7wETPBDTRfu48Cg1DS1cb2Uhj5s7LpXcQ8
epqLHyUCBQVVbklzMtsy9Jsdbz3pOuG5NCqEu83qfAE4lgEzBMRAaLSDiwAPin1j8iMgH4Ex1J
oLLT8j4iEuF+nghcDuz0QEwXbUV+ObGRoVwxzOYl9rQ5WamowurtznWYMZ6zq7yWoqOnrLV
QD+gdFc5nhiWy0svloW4nAlVtBR4E1gC7gFdUdYeIPCYip1sB/Q8QA/z9jGaio4BcEckD3gV+pq
peTwRNrW28ubOC68akEBFqZSFPG5ocw8iUWFYVWCIIBCsLOgZkvM563veIOVmplJ5oIK/Ee
+WhUE+8iaquAladseyHnZ7POMt+64AsT8TQHR/uPUptY6u1FupBc7JS+eXavVTUNForEz+mq
qwqqGB6Zl/62oCMPWLm6BR+EFLAqoJysvv39soxrWcxsKqgnLjIUC4fYmWhnnJ6OG8rD/m3X
eW1HDh6yoZn70HxUWFcMdS75aGgTwRNrR3j5s8c0zHxuukZVh4KDKsKyq0s5AXeLg8F/Tffx4
XuspCd4fS4uVmpbDporYf8VUdZqJxpmQlWFuphM0enEBYiXjtxCvpEsDK/oqMsNNTKQj1tzjgrD
/mz3RUdrYWsLNTz4qPCuNyL5aGgTgTNre2s3VnBtaOtLOQNQ5I6ykMrHRx33Vy6VQXluKSjrb
vpeafLQ/leKA8F9bffx4VHOdnYytxx9sH2lrlZ1rnMH6kqKws6OpHZ9K3ecZ27PLTSC+WhoE4EK
wvKiY2wspA3zXaXFd6w8pBf2V1RS1GVlYW8yZvloaBNBM2t7bzpHlvIOpF5z9DkGEb0i2XVdhu
Ezp+sdpeFZo21q2dvOl0e6umxh4I2Eazb31EWsjMc75udlcKmg8dsaGo/smp7BVMHW1nI22aO7
keoS1jVw6P3Bm0iWF1QQUxEKJ8ZbmUhbzs99tAauyrwC/uO1FJYWcecLLsa8LbeUeFMH9KX
1dt7tjwUIlmgpa2dNTsrmDEq2cpCDhiWHMOQpOgeP8sxnrGqoAKx1kKOmZOVyqHqenaWn+yx
YwRIIvik6Bgn6lv+cePSeJeIMDcrlU8OVHO0run8OxhHrd5ezuSBCSTH2RhRTpg5uh8hLunRyZ2
CMhGs2t4xQb3NROac2VmptCussYntfdr+qjp2V9Qy28pCjukbE8G0zARWFfRceSjoEkFbu7Jme
wVXj0y2mcgcNDIllsGJ0TaFpY97w30fx1oLOWv22FSKjp5i75G6Hnn/oEsEGw8co/pUs7UWcpiI
MHtsCuuLqjl2qtnpcMxZrCooZ+KA3qTG93l6lKB23ZgUROixsYeCLhGs3l5OZJiLK0dYWchpc7JS
aWtX1u60qwJfdKj6FDvKTtpJkw9Iio1gyqCEHhunyyOJQERmicgeESkUkUe6WB8hIi+7138iIoM6
rfu+e/keEbnOE/GcTXu7snp7BVeNSCYq3CNz8phuGJMWx4CEKGs95KNWW1nIp8zJSmXvkTo
KK2s9/t7d/jYUkRDgCeBaoATYJCLLzphy8j7guKoOFZEFwH8DnxeR0XTMcTwGSAPeEpHhqtr
W3bi6svnwcapqm6y1kI8QEWZnpfD0hweoqW8hPirM6ZBMJ6sLyhmXEU9GnyinQzF0dMSMjggl
pQfKdJ64IpgCFKpqkao2A4uA+WdsMx94zv18MXCNiIh7+SJVbVLVA0Ch+/16xKqCcsJDXVw9M
rmnDmEu0uyxqbS2K2t3HXE6FNNJyfF68kpqrCzkQ5JjI7I1UgYxEZ6vZngiEaQDxZ1eI7iXdbmNe
7L7GqDvBe4LgIjcLyK5IpJbVVV1SYG2tSuzxqT0yB/SXJrxGfGkxUfaIHQ+5nRrodlWFgoKfvONq
KpPAk8C5OTkXFJj2sfmj/XaHKDmwnSUh1J5fv0hahtbiI208pAvWFVQzpi0OAb2jXY6FOMFnrgi
KAX6d3qd4V7W5TYiEgrEA9UXuK9HdVSkjC+Zk5VCc1s77+yudDoUA5TXNLDl8AkrCwURTyS

CTcAwERksIuF03PxddsY2y4CF7ue3Au9ox6n5MmCBu1XRYGAYsNEDMRk/MqF/H/rFRdjE9j7
CykLBp9uIIVVtFZEHgTVACPAXVd0hIo8Buaq6DHgaeF5ECoFjdCQL3Nu9AuwEWoGv91SLIeO
7XC5h9thUXtp4mFNNrUTbPRxHrS6oYGRKLJlJMU6HYrzEI/0IVHWVqg5X1SGq+rh72Q/dSQB
VbVTV21R1qKpOUdWiTvs+7t5vhKqu9kQ8xv/MHptCU2s77+6x8pCTKk82sunQMWaPtbJQMA
m6nsXGN+UMSiAxJsLGHnLYmh0VqGJzDwQZSwTGJ4S4hFlj+/HO7koamq066OKZgxsAABU
DSURBVJRVBRUMTY5hWL9Yp0MxXmSJwPiMOVmpNLS08Z6VhxxxtK6JTw5U203iIGSJwPiM
KYMS6BsdbhPbO2TNjgraFWs2GoQsERifERriYuaYFN7ZdYTGFisPeduqgnIyE6MZmWJloWBji
cD4lLlZqZxqbuP9vZc2jIi5NNV1TWwoOsbsrBTrdBmELBEYnzI1M4E+UWGsts5lXvXmziO0tauV
hYKUJQLjU8JCXMwcncJbuyqtPORFqwrKGdQ3itGpcU6HYhxgicD4nDnjUqlrauWjfUedDiUoHD/
VzLr91czOSrWyUJCyRGB8zmVD+hLfK8zGHvKSN3dW0NauzLWyUNCyRGB8Tkd5qB9rdx6hq
dXKQz1tZUEFAxKiGJNmZaFgZYnA+KS541KpbWrlw71WHupJJ+qbWVd4lDlWFgpqlgiMT7p8a
CLxvcJYaeWhHvXmjiO0tquNLRTkLBEYnxQW4uK6Mf14a6d1LutJKwrKGZAQRVZ6vNOhGAdZ
ljA+a+64tI7ykLUe6hHHTzXzceFR5o6zslCws0RgfNZlQ/rSOyqMlflITocSkNbssNZCpkO3EoGIJljI
WhHZ5/7Zp4ttskVkvYjsEJF8Efl8p3XPisgBEdnmfmR3Jx4TWMJCXMwak8JaKw/1iJUF5QxOjLb
WQqbbVwSPAG+r6jDgbffrM9UD96rqGGAW8GsR6d1p/cOqmu1+bOtmPCbAzLGxh3pEdV0T6/
ZXM9daCxm6nwjmA8+5nz8H3HjmBqq6V1X3uZ+XAZVAUjePa4LE9CF96RNlncs87Y3TZaFxV
hYy3U8E/VT19P/QCqDfuTYWkSlAOLC/0+LH3SWjX4IlxDn2vV9EckUkt6rKzg6DRViIi1ljO8pDN
nOZ56zMLyczyYacNh3OmwhE5C0R2d7FY37n7VRVAT3H+6QCzwNfVNV29+LvAyOByUAC8L
2z7a+qT6pqjqrmJCXZBUUwuWFcGvXNbTaxvYdU1Taxoaia660sZNxCz7eBqs442zoROSIiqapa
7v6i7/J/qojEASuBf1fVDZ3e+/TVRJOIPAN896KiN0FhamZfEmMiWJ5XZsMke8Ab28tp147mucZ
A90tDy4CF7ucLgaVnbiAi4cBrwF9VdfEZ61LdP4WO+wvbuxmPCUAhLuH6cam8s7uS2sYWp8P
xe8vyyhjeL4YRVhYybt1NBD8DrhWRfcAM92tEJEdEnnJvczvwWeALXTQTfVFECoACIBH4cTfj
MQHqhvGpNLW289auI06H4tdKTzSw6eBx5o23qwHz/5y3NHQuqloNXNPF8lzgy+7nLwAvnGX/q
7tzfBM8JvTvQ3rvXizPK+emCRlOh+O3TnfOu97KQqYT61ls/ILLXR76YG8VJ+qbnQ7Hby3LK2N
8RjyDEqOdDsX4EEsExm/cMD6N1nblje0VTofiI4qq6theepIbrCxkzmCJwPiNMWlxDE6MZrmNPX
RJluWVIYIIAvMplgiM3xARbhiXyvr91VSebHQ6HL+iqizLK2Pq4AT6xUU6HY7xMZYIjF+Zl51Ou8
LyfBty4mLsKDtJUdUp5o1PdzoU44MsERi/MjQ5hrHpcSzdVup0KH5leV4ZoS5h9libicx8miUC43d
uzE4nv6SG/VV1TofiF9rbO8pCnx2eRJ/ocKfDMT7IEoHxOzeMT8MlsHSrXRVciA0HqimvaeSmC
VYWMl2zRGD8Tr+4SC4bksjr28roGOvQnMvrW0uJiQhlxqhzDg5sgpglAuOX5mencfhYPVsOn3A
6FJ/W2NLG6oIKZo1NoVd4iNPhGB9licD4pVljU4gIddlN4/N4a9cRaptaudnKQuYcLBEYvxQbGca
M0f1YkV9OS1v7+XcIUq9vLSUlLpKpmX2dDsX4MEsExm/dmJ3OsVPNfGDzGXfp2Klm3ttTxfzs
NEJcNgGNOTtLBMZvfW54EgnR4SzZYuWhrqzIL6O1XbnRykLmPCwRGL8VHupifnYaa3cesRF
Ju/Da1lJGpsQyKjXO6VCMj+tWIhCRBBFZKyL73D/7nGW7tk6T0izrtHywiHwiIoUi8rJ7NjNjLtitkzJ
obmtnWZ4NRNdZYWUdWw+f4OaJdjVgzq+7VwSPAG+r6jDgbffrrjSoarb7Ma/T8v8GfqWqQ4Hjw
H3djMcEmTFp8YxKjWPx5hKnQ/Epf99cTIhLbBIfc0G6mwjmA8+5nz9Hx7zDF8Q9T/HVwOl5jC9
qf2NOu3VSBvklNeypqHU6FJ/Q2tbOki2lXDUimaTYCKfDMX6gu4mgn6qeHgayAjhb18VIEckVk
Q0icvrLvi9wQlVb3a9LgLNex4rI/e73yK2qslYi5v+Zn51GqEt4dYtdFQC8v7eKqtombs+xqwFzYc6b
CETkLRHZ3sVjfufttKOv/9n6+w9U1RzgTuDXIjLkYgNV1SdVNUdVc5KSki52dxPAEmMiuGpkMk
u2lNJqfQr4e24JiTHhXDUy2elQjJ84byJQ1RmqOraLx1LgiIikArh/Vp7lPUrdP4uA94AJQDXQW0R
C3ZtlANYO0FySWydlcLSuiQ/2BffVYnVdE2/tOsJNE9IJC7FGgebCdPeTsgxY6H6+EFh65gYi0kd
EItzPE4HLgZ3uK4h3gVvPtb8xF+KqEckkRIfz8qZip0Nx1OvbOvoO3JbT3+lQjB/pbiL4GXCtiOwD
ZrhfIyI5IvKUe5tRQK6I5NHxxf8zVd3pXvc94NsiUkjHPYOnuxmPCVLhoS5unZTB27sqg3YaS1Xl
77nFjO/fm+H9Yp0Ox/iRbiUCVa1W1WtUdZi7hHTMvTxXVb/sfr5OVbNUdbz759Od9i9S1SmqOl
RVb1PVpu79OiaYLZjcn9Z25e9B2pQ0v6SG3RW13DbJbhKbi2NFRBMwMpNimJaZwKJNh2lvD

755Cl785BBR4SHMz05zOhTjZywRmIByx5QBFB9r4KPCo06H4lU1DS0syytjfnY6sZFhTodj/IwlA
hNQZo1NoU9UGC9tPOx0KF712pYSGlvauWvqAKdDMX7IEoEJKBGhIdw6KYO1O49QWRscN
41VlRc/Ocz4jHjGpsc7HY7xQ5YITMBZMGUAre0aNOMPbTp4nH2Vddw1daDToRg/ZYnABJwh7
pvGf/vkMG1BcNP4xU8OERsZyvXjU50OxfgpSwQmIN07fRAlxxt4e9cRp0PpUdV1TawuqOCWiR
lEhYeefwdjumCJwASkmaP7kRYfybPrDjodSo96ObeY5rZ27rSbxKYbLBGYgBQa4uKe6YNYt7+a
3RUnnQ6nR7S0tfPXdYe4fGhf60lsusUSgQlYCyb3JzLMxXMBelWwqqCcipON3HfFYKdDMX7O
EoEJWH2iw7lpQjpLtpRy/FRgzWmsqjz90QEyk6K5crgNN226xxKBCWgLLxtEU2s7iwJsVNLcQ8f
JL6nhi5cPxuUSp8Mxfs4SgQloI1PiuGxIX55ff5CWAJq05ukPDxDfK4xbbHJ64wGWCEzA+9Llgy
mraWRFfpnToXhE8bF63txZwZ1TB1iTUeMRlghMwLt6ZDlj+sXy+3f3B8SopM98fBCXCAunD3l6
FBMgLBGYgOdyCV+7agj7Kut4y887mB2ta+JvGw8xLzuNlPhIp8MxAaJbiUBEEkRkrYjsc//s08U2
V4nItk6PRhG50b3uWRE50GlddnfiMeZs5malMiAhiife20/HLKn+6akPD9DU2s7XrxrqdCgmgHT3i
uAR4G1VHQa87X79T1T1XVXNVtVs4GqgHniz0yYPn16vqtu6GY8xXQoNcfHA5zLJKz7B+v3VT
odzSY6faub59Qe5flwaQ5JinA7HBJDuJoL5wHPu588BN55n+1uB1apa383jGnPRbpmYQVJsBE
+8V+h0KJfkmY8PcKq5jQftasB4WHcTQT9VLXc/rwD6nWf7BcBLZyx7XETyReRXIhJxth1F5H4R
yRWR3Kqqqm6EbIJVZFgIX/nMYD4urGbL4eNOh3NRTja28My6g8wak8KIFBtOwnjWeROBiLwlI
tu7eMzvvJ12FF7PWnwVkVQgC1jTafH3gZHAZCAB+N7Z9lfVJ1U1R1VzkpKSzhe2MV26a+pA+
kaH8z9v7PGrewV/XXeQ2sZWHrzargaM5503EajqDFUd28VjKXDE/QV/+ou+8hxvdTvwmqq2dHr
vcu3QBDwDTOner2PMuUVHhPLg1UNZX1TNh/v8Y17jmvoW/vzhAa4emWwzkJke0d3S0DJgof
v5QmDpOba9gzPKQp2SiNBxf2F7N+Mx5rzunDqA9N69+Pma3X7Rr+CJ9wo52djCw9eNcDoUE
6C6mwh+BlwrIvuAGe7XiEiOiDx1eiMRGQT0B94/Y/8XRaQAKAASgR93Mx5jzisiNIRvXzuc7aUn
WbW9/Pw7OKj4WD3PfnyQWyZmMCo1zulwTIDqVv90Va0GrulieS7w5U6vDwKfGhRFVa/uzvG
NuVQ3TkjnyQ+K+MWaPVw3JoWwEN/sW/mLN/fgcsF3Zg53OhQTwHzz029MDwtxCQ9fN4KD1
fUs2njY6XC6lF9ygqXbyrjvisGkxvdyOhwTwCwRmKB1zahkpg5O4Bdv7uVoXZPT4fwTVeWnq3a
TEB3OA58b4nQ4JsBZIjBBS0T48Y1jOdXUyk9X7XY6nH+yPL+c9UXVfHPGMOIiw5wOxwQ4Sw
QmqA3rF8v9n83k1S0lbCjyjaEnjp9q5r+W7WB8Rjx3TR3odDgmCFgiMEHvoauHkdGnF//x+naa
W52fvObxVbuoaWjhpzePI8RmHzNeYInABL1e4SH817wxFFbW8ecPixyN5aN9R1m8uYQHPpfJ
6DRrLmq8wxKBMcA1o/oxe2wKv3lrH9tLaxyJoaG5je+/lk9mYjQPXT3MkRhMcLJEYIzbT27KIiE6
nIde2kpdU6vXj//osu0UH2vgJzdnERkW4vXjm+BlicAYtz7R4fxmQTaHqk/xw9e9O9rJy5sO80puC
Q9dPZRpmX29emxjLBEY08nUzL5845rhLNlayqubS7xyzO2lNfzn0h1cMTSRb86wHsTG+ywRG
HOGB68eytTBCfzH69t7fN6CE/XNfPWFzfR1X41YKyHjBEsExpwhxCX8350TSY6L4IvPbGLvkdo
eOU59cyv3P7+ZIycbeeKuifSNOeu8TMb0KEsExnQhKTaCF+6bSkSoi3ue/oTiY56dXbWhuY0vP
buJ3IPH+N/bs5k4oI9H39+Yi2GJwJiz6J8QxV/vm0JDcxv3PP0JJcc9kwwamtu477lNbDxwjF/ens2
88WkeeV9jLpUlAmPOYWRKHM98cQrVdc3M+7+PWVfYvVnNymsaWPiXjawvquYXt43nxgmfG
p3dGK+zRGDMeUwa2IelD15O3+hw7n76E578YP8lzXe8ZkcFs3/zIdvLavj157O5eWJGD0RrzM
XrViIQkdtEZIeItItIzjm2myUie0SkUEQe6bR8sIh84l7+soiEdyceY3pKZlIMr339cmaNTeEnq3Zz2x/
X8+G+qgtKCIer6/ne4nweeH4z/ftEseKhK5ifbVcCxnfIpZzZ/GNnkVFAO/An4LvumcnO3CYE2Atc
C5QAm4A7VHWniLwCLFHVRSLyRyBPVf9wvuPm5ORobu6nDmVMj1NVXtpYzO/e2Ud5TSMT
BvTmzikDGJsez9DkGMJCXLS3K1V1TWwvreHFTw7z7p5KXCJ8+TOD+c61IwgPtQtx4wwR2ay
qnzpp7+5Ulbvcb36uzaYAhapa5N52ETBfRHYBVwN3urd7DvgRcN5EYIxTRIQ7pw7glknpLN5cw
u/f3c/Di/MBCA91kRQTQWVtly1tHSdYiTERPHTVUO6YOsBmGTM+q1uJ4AKlA8WdXpcAU4G+
wAlVbe20/KzXyyJyP3A/wIABA3omUmMuUERoCHdNHciCyQM4cLSOHWUn2Vl2kqraJlLiI0nt3
YsBCVFMz+xrVwDG5503EYjIW0BKF6v+XVWXej6krqnqk8CT0FEa8tZxjTmXEJcwNDmWocmx
Vvc3fuu8iUBVZ3TzGKVA/06vM9zLqoHeIhLqvio4vdwYY4wXeeOadRMwzN1CKBxYACzTjrvU7
wK3urdbCHjtCsMYY0yH7jYfvUlESoDpwEoRWeNeniYiqwDcZ/sPAmuAXcArqrrD/RbfA74tIoV03

DN4ujvxGGOMuXjdaj7qFGs+aowxF+9szUetOYMxxgQ5SwTGGBPkLBEYY0yQs0RgjDFBzi9vF
otIFXDoEndPBLo3lrD/s7+B/Q2C/feH4PwbDFTVpDMX+mUi6A4Rye3qrnkwsb+B/Q2C/fcH+xt0Z
qUhY4wJcpYIjDEmyAVjInjS6QB8gP0N7G8Q7L8/2N/gH4LuHoExxph/FoxXBMYYYzqxRGCMM
UEuqBKBiMwSkT0iUigijzgdjzeJSH8ReVdEdorIDhH5htMxOUVEQkRkq4iscDoWJ4hIbxFZLCK7
RWSXiEx3OiZvE5Fvuf8fbBeRl0Qk0umYnBQ0iUBEQoAngNnAaOAOERntbFRe1Qp8R1VHA9
OArwfZ79/ZN+gYEj1Y/QZ4Q1VHAuMJsr+FiKQD/wrkqOpYIISOeVKCVtAkAmAKUKiqRaraDCw
C5jsck9eoarmqbnE/r6XjP3/Qza0oIhnAXOApp2NxgojEA5/FPfeHqjar6glno3JEKNBLREKBKKD
M4XgcFUyJIB0o7vS6hCD8IgQQkUHABOATZyNxxK+BfwPanQ7EIYOBKuAZd3nsKRGJdjoob1
LVUuAXwGGgHKhR1TedjcpZwZQIDCAiMcCrwDdV9aTT8XiTiFwPVKrqZqdjcVAoMBH4g6pOA
E4BwXa/rA8d1YDBQBoQLSJ3OxuVs4IpEZQC/Tu9znAvCxoiEkZHEnhRVZc4HY8DLgfmichBO
kqDV4vIC86G5HUlQImqnr4aXExHYggmM4ADqlqlqi3AEuAyh2NyVDAlgk3AMBEZLCLhdNwcW
uZwTF4jIkJHXXiXqv7S6XicoKrfV9UMVR1Ex7//O6oaVGeCqloBFIvICPeia4CdDobkhMPANBGJ
cv+/uIYgu2F+plCnA/AWVW0VkQeBNXS0EviLqu5wOCxvuhy4BygQkW3uZT9Q1VUOxmSc8R
DwovuEqAj4osPxeJWqfiIii4EtdLSm20qQDzdhQ0wYY0yQC6bSkDHGmC5YIjDGmCBnicAYY4K
cJQJjjAlylgiMMSbIWSIwxpggZ4nAGGOC3P8PYUfkhBhZgZUAAAAASUVORK5CYII=\n",
          "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":
"iVBORw0KGgoAAAANSUhEUgAAAZAAAAEWCAYAAABIVsEJAAAABHNCSVQICAgIfAhkiAA
AAAIwSFIzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW
9uMy4yLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy+WH4yJAAAgAElEQVR4nOydd3hU55X/P0d
dQgXUCyCaKOp0424DphmEjXuJneYkm2w2cTZZJ9mNvcl6f042m7KJN1mn2Ym7sU3HBveCM
YgiVChCNEIIQkggVFA/vz/uyJGxJISm3Lmj+3me+8zMrd8B3Tn3Pee854iqYmNjY2Njc6n4mS3A
xsbGxsaa2AbExsbGxmZI2AbExsbGxmZI2AbExsbGxmZI2AbExsbGxmZI2AbExsbGxmZI2AbEZ
IggIneLyBazdVwMEXIHRL7k4Wv+QET+6MIr2vgGtgGx8RIE5EoR2SYiDSJSLyIfishsAFV9RIVv
MFujs4jIZBF5SUROO77nPhF5UET8h3pOVf1PVo0bLxDWwDYuMTiEgksAH4DRANpAD/DrS
ZqcuViMhE4GOgHMhS1SgVqyQCq+pyqdqnqeVXdoqr7AETkTkr
GdnEVER+aqIIIrIWRF5XESk1/YviMh+ETkjqIq+LSGp/F3aMCKodI4L3RCSj17YnHefeKCKNIvKx
wxD0bF8oIgccx/4WkD4vYvDvwDZVfVBqxzf96Cq3qWqZx3nWyEiLi
iMt+x/hERedrxfpzj3+Y+ETnhGOn8sNc5/ETkIREpEpE6EXlRKIv+r9j45PYBsTGVzgEdInUyKy

RERGDeKYG4HZQDZwG7AIQETygB8ANwNxwPvAcwOcZzOQBsQDu4FnLth+B8aP/yjgMPCo
4zqxwCvAvwKxQBlwxQDXWQCs7m+jiEx26PyWQ/cmYL2IBInIFOAbwGxVjXB812MDXOtKYAo
wH/hRL0P0j8BK4BogGTgDPD7AeWx8GNuA2PgEqnoO40dPgT8AtSKyTkQSBjjsMVU9q6ongL
eBXMf6rwL/T1X3q2on8J9Abn+jEFX9s6o2qmob8AiQIyJRvXZ5VVV3OM71TK/rLAWKVXW1qnY
AvwKqB9AbA1QNsP12YKOqbnWc7+dAKHA50AUEA+kiEqiqx1S1bIBz/btjFFcAFAA5jvVfBX6oq
hW9vu8tIhIwwLlsfBTbgNj4DI4f/PtVdTSQifGE/KsBDun9Y90ChDvepwK/driBzgL1GK6llAtPICL+I
vKYw6Vzjr8/1ccO4jrJGPGMHv3a+3Mf1AFJA2xPBo73Ol+343wpqnoYY2TyCHBKRJ4XkeQBzjX
Qv82rvf5t9mMYp4EMtY2PYhsQG59EVQ8AT2IYkkulHPiKqo7stYSq6rY+9r0LyMNwL0UB4xzrB
4pl9FAFjOn54IjBjOl/d94AVg2w/STGD/yF56sEUNVnVfVKxz4K/HQQGi+kHFhywb9NiKpWDuFc
NhbHNiA2PoGITBWR74jIaMfnMcCdwPYhnO73wPd7guEiEiUit/azbwRGplcdEIbh7hosG4EMEb
nZ4QL6JpA4wP4PA5eLyH+JSKJD2yQReVpERgIvAstEZL6IBALfcWjbJiJTROR6EQkGWoHzQ
PclaO3h98CjPe48EYlzxIxshiG2AbHxFRqBucDHItKMYTiKMH5ELwlVfRXj6fx5h1uqCFjSz+5/xX
AbVQIIXILBUtXTGGm4j2EYoDTgwwH2LwPmYYxyikWkAXgZyAcaVfUgcA9GKvNpYDmwXFXb
MeIfjznWV2ME/L8/WK29+DWwDtgiIo0Y33fuEM5j4wOI3VDKxsbGxmYo2CMQGxsbG5shYRsQ
GxsbG5shYRsQGxsbG5shYRsQGxsbG5shMaxmj8bGxuq4cePMlmFjY2NjKXbt2nVaVeMuXD+
sDMi4cePlz883W4aNjY2NpRCR432tt11YNjY2NjZDwjYgNjY2NjZDwjYgNjY2NjZDwjYgNjY2NjZ
DwjYgNjY2NjZDwlQDIiJ/FpFTIlLUz3YRkf8RkcMisk9EZvTadp+jHWmpiNznOdU2NjY2NmD+CO
RJYPEA25dgVChNAx4Afgfg6MH8MEYV0DnAw4NsYWpjY2Nj4yJMnQeiqu+JyLgBdskD/uro1L
ZdREaKSBJwLbBVVesBRGQrhiEaqG/10Cl4HpprISYNYtNgZCr4W2cKTVe3srf8LLWNrZw738m
51g6SR4Yye1w0cRHBZsuz8QUaa6DuMDRWQWM1BARD8gxIzDTeW4izLe1sK6ujqa2T7m6lW
2Fi3Ahmpo4iwN/sZ27vwtt/BVP4dIvPCse6/tZ/BhF5AGP0wtixY4emovhVOPTa3z8HR8Gs+2Hu1
yByoA6j5nKoppGXd1ewZk8lNefa+txnQuwIFqQn8NVrJhI9IsjDCm0sTXcXlG6F/D9D6RaMJocX4
BcIKTPhym/B5MUgg2nU6Hma2jp5ZXcFrxVV8/HRerq6P/tdIkMCuHpyHHm5KSyYFo946XfxJN5
uQJxGVZ8AngCYNWvW0Jqf3PUCtNTD6VKoKzVumm2/gY/+F7JvgwWPQHi860Q7ydmWdh5Z
V8yavSfx9xOunRzHD5elMDFuBJEhgUSEBHD0dDM7j9Wz/Ug9f3z/CM99fIKvXjuRL1wxntAgf7
O/go23U/oGbPg2NJyA8AS46jsw7kqISIKIBGhrgpN7jKX4FXjuDkieDtf9ECYt8BpDoqq8XlzNI+tK
qD7XysS4EXzl6gksSE8gLjwYfz9D576Ks7x14BRvH6xlw74qrpwUyyMr0pkUH2HyNzAX0xtKOV
xYG1T1M72rReT/gHdU9TnH54MY7qtrgWtV9St97dcfs2bNUpeVMqk/Ctv/F3Y9BaGj4JY/GTeQy
WwtqeEHrxZyprmdr107kfsuH0ds+MAuhNKaRn762kHe2F9DUlQIv79nJjljRnpIsY2l6DgPW38EO
56A+HS49vswZQn4B/Z/TFeH4QZ+72dw9gTM+Bws/bnprq3Ks+f5tzVFvHXgFNOSIvmPlRnMTI
0e8JjOrm6e3XGCn79+kJb2Lr541Xj++YYpBPq4a0tEdqnqrM+s93IDsgz4BrAUI2D+P6o6xxFE3
wX0ZGXtBmb2xET6w6UGpIfqInjpPqg/YjxdXfkg+Hn+j6m7W/nxhhKe3HaMqYkR/PdtOWQkR13
SOXYcrefbL+zldFMbv7gtl2XZ3ueszGB2kPwwj1w+iBc9nWY/yMIDBn88Z3t8O5j8P5/w+jZcNvf
THMBF59s4L4/76SlvZMHF07m/svHXVJ843RTGz977QAv5ldw7ZQ4Hr9rBiOCfdeh45UGRESe
wxhNxAI1GJlVgQCq+nsxnIy/xQiQtwCfV9V8x7FfAH7gONWjqvqXi13PLQYEoK0R1n8LilZD9u2
w8vceNSJd3cpDL+/jpV0VfP6KcXx/yTSCAoZ2/dNNbXzlb7vYdfwM31k4mW9cP8n29drAqQPw1
HJA4eYnYOL1Qz9X8RpY8w8QHG64h5Onu0zmYNh+pI4vP5VPeEgAf/3CHNIShu6Gem7HCX
74aiGZKVH8+f7ZFx3tWxWvNCCexm0GBEAV3vs5vP0fMPN+uPFXHvHzdnR18+CLBawvOMk/
zU/jWwvSnP7Bb+3o4vuvFPLqnkq+ft1EvrtoqovU2liSUwfgqRtB/OC+DRA32flz1pTAs7dDexN84
XXXnHMQbCmu5hvP7WHMqFD+9sW5JI8Mdfqcb+6v4evP7iY+IoRnvzyX0aPCXKDUu+jPgPi2
486TiMA13zWCibuehNd/aBgVN9LdrXzr+b2sLzjJQ0um8u2Fk10yWggJ9OcXt+Vw55yxPP52GU
9v77OSs81wwB3GAyAhHT63Bvz84emboaHSNecdgPxj9Xzj2T1MS4pk9Vcvd4nxAJg/LYHnvnw
ZZ1ra+eKT+TS2drjkvFbANiCu5vp/g7lfhe2Pw7s/c+ulfv1mKRsLq/jB0ql89ZqJLj23iPCTvAzmT43
nR2uL2FpS49Lz21iAlnp45lbDeNy/0fWjhJiJcPdqOH8Wnl5lXM9NlNe38JW/7SJ5ZAhPfX42o1yc

sj597Ch+d/dMDtc28Y/P7aGzq9ul5/dWbAPiakRg8WOQcye8859Gyq8b2FJcza/fLGXVjNF8+aoJ
brlGgL8fv7lrOlkpUfzjc7vZW37WLdex8UK6u+DlL0FTNdzxnDGB1h0k58Kdz0J9Gbz4OeO6Lqap
rZMv/zWf9q5u/njfbEaGuWe+05VpsfwkL5N3Dtbykw0lbrmGt2EbEHcgAjf+EhIy4ZUHoKHCpac/f
KqRb7+wl5zRUTx6U6Zbg9xhQQH8yREc/Pozuzk3jIbnw5p3fwplb8KSn8Home691virYfmv4dj7R
oaWC+lx85aeauJ/757BpPhwl57/Qu6aO5YvXzWepz46zjMf+77r1zYg7iIwFG59ErraYfUXjFx4F9
Dc1skDf91FaJA/v793JiGB7p/0FxsezK/vmE71uVYeXlvs9uvZmMzB1wwDknuPkRDiCXLuhKzb4
J3/Bye2u+y0f/3oGG/sr+Ffl03jqrTPtPR2Cw8tmcZVabH8ZEMJR083e+SaZmEbEHcSm2Y8WZV/
DG/9xCWn/K/XD3K0rpnf3jWDpCjXBAEHw8zUUXzz+jRe3VPJ2r3uD3jamETTKVjzVUjMhmU/9
9yMcRFY9t9GnbmXvwTnzzh9yiO1TTz22gGunRLH/ZePc17jIPH3E35+aw7BAf58+4W9Ph0PsQ
2Iu8m6xXiK+/DXUL7DqVPtOFrPk9uOcd+8cVw2IcY1+i6Br183kZmpo/jXV4sor2/x+PVtPMDm70
F7M6z6kzGK9iQhkUZFh8YqWPdNp7IYu7qVf36pgOAAf366Ktvjc5kSIkP4j5WZ7C0/y+/fLfPotT2
JbUA8wQ2PQuRoWP9PQ3ZlnW/v4nurCxgTHcr3Fk9xscDBEeDvx69uz0WB764uYDjNIRoWHN
hkFA695nsem5fxGVJmGhUd9q+Dg5uGfJo/vH+E3SfO8uO8DBIiL2G2vAtZnpPM8pxkfvVGKUW
VDaZocDe2AfEEweGw9L/gVIlRhHEI/GLrQY7VtfDTVdmEBZlXMmFMdBg/WDqN7UfqWVdw0j
QdNi6mtQE2PmgkflzxLXO1XP6PEJ8Bm//FGA1dIqU1jfxiyyGWZCayIifZDQIHz0/yMogJD+KfXyr
wSVeWbUA8xdSlMG25EZysP3pJhxaUn+VPHxzl7rljuXxirJsEDp7bZ48he3QU/7Fx/7CaNOXTb
H0YmmpgxW8GLozoCfwDjXhIQ/klz6VSVR5ZX0xokD8/WeneDMXBMDIsiEeWZ3CgupHndpww
VYs7sA2IJ1nyM6M/wsYHB+3fVTWKJEaPCOahJd5RUsTfT/hxXianm9r49RulZsuxcZaKfNj1F7js
HyBlxsX39wSp84wssI9+a8yGHyRbS2r48HAd316Q5jV1qRZnJjJvQgz/vfUQZ5rbzZbjUmwD4kki
k40KpmVvwYENgzpkY2EVu46f4buLJhMRYvKTYS9yx4zkjtlj+Mu2YxysbjRbjs1QUTXK7oyIN0qz
exMLfwzBEbDxO4N64Grr7OLRTftJiw/n7stSPSBwcIgID69I59z5Dn6x9ZDZclyKbUA8zawvQOw
UeOPfoatzwF1bO7p4bPMBpiZGcMvMMR4SOHi+u2gqESEB/GhtkR1Qtyr710P5drj+h0aszpsYE
WM0azv+AZSsuejuf/nwGMfrWvjR8nSv688xNTGSey9L5ZmPj7O/6pzZclyGd/0rDwf8A4yboq4U
9vx1wF3/8uExKs6c51+XpX/SGc2biB4RxHcWTubjo/W8deCU2XJsLpXOdnjjYYibZriLvJHp9xr63
np0wAeuU42t/ObNUhZMi/fYhMFL5dsLJxMVGsgj64p95oHLNiBmMGUJjLkM3nms3yyT001tPP7
2YeZPjefKNPMD5/1xx5yxjI0O4+dbDtHdRx9pGy8m/89GI7SFPzYebLwRP3+Y/2/GA1fBs/3u9sut
h2jv6uaHy9I9KO7SGBkWxLcdD1zvHqo1W45LsA2IGYgYN21TjdFXvQ9+82YprR1d/GDZNA+Lu
zQC/f349sI09ledY1NRldlybAbL+bNGRuD4ayBtodlqBmbKUqOD4TuPGS11L6C8voWX8iu4a85
YxseOMEHg4Llj9lhSRobyi62HfGIUYqoBEZHFInJQRA6LyEN9bP+liOx1LIdE5GyvbV29tq3zrHIX
MHYuTL3RmKHefPpTm6obWnluRzm3zBzNxDgv80v3wYqcFCYnhPOLLYd8MtfdJ/not3C+Hm7
4iefKlQwVEZj/MJyrhJ1//Mzm37xVip+f8A/XTTJB3KURFODHN+dPYl9FA2/st77b1zQDIiL+wOPA
EiAduFNEPjX+VNVvq2ququYCvwFe6bX5fM82VV3hMeGuZP7D0NEMH/zyU6t//24Z3ap83QI3B
Bhpvd+5YQpHTjfzyh67TpbXc/4sfPx/xrykpByz1QyO8VfBxPnw/i+MSY8Ojp1u5uXdldw9d6xpM8
4vlZtnjCY1JoxfbLW+29fMEcgc4LCqHlHVduB5IG+A/e8EnvOIMk8RNxkyb4H8v3zSTKfmXCvP7j
jBzTNSGBNtndaYN6QnkDM6il+/UUpbp+t7Oti4kB1PQNs5uPp7Ziu5NOb/yBg1bf/dJ6v+561SAv
2Fr13r2oZq7iTQ349/mm+4fV8rrjZbjlOYaUBSgPJenysc6z6DiKQC44G3eq0OEZF8EdkuIiv7u4iI
PODYL7+21gsDV1c9aIxCHDfF798to6tb+cZ1bmrg4yZEjFFI5dnzvLzLHoV4LW2N8NHjMHkJJG
WbrebSSM41dH/8e2hv5khtE2v2VHLvZanER1hj9NFDXm4KE+NG8Muth+iy8CjEKkH0O4DVqtr
70TbV0eT9LuBXItLnI4iqPqGqs1R1VlycF6b3xU8zYiE7/o/a2lqe/fgEN09PYWyMdUYfPVyVFkv26
CieeK/M0jeFT7PjD9B6Fq75rtlKhsZVDxql3nc9xW/eOkxwgD9fcXE7Z0/g7yd8a8FkSk818VqRdU
chZhqQSqD37LjRjnV9cQcXuK9UtdLxegR4B5jueoke4up/htYG9q35bzq7lW9cb43Yx4WICF+7Zi
LH6losfVP4LO3NRvB80gKj6q0VGTMHUq+k88PfsLngBPdcNtZrSpZcKkuzkhgXE8YT75VZNiPL
TAOyE0gTkfEiEoRhJD6TTSUiU4FRwEe91o0SkWDH+1jgCsC6TYiTp9Mx/npyK57hlqxoUmO8
OxVxIG7ISGR87Ah+/651bwqfZdeT0FJnvdjHhVz1bQKaTrLS732+cOV4s9UMGX8/4UtXTaCgoo

HtR+rNljMkTDMgqtoJfAN4HdgPvKiqxSLyYxHpnVV1B/C8fvrXaBqQLyIFwNvAY6pqXQMCrI+6ix
g5x7djPrr4zl6Mv5/wwNUTKKxsYFtZndlybHro6jTmHI27ykghtzANSVdTouN4MGwzSRFBZstxilt
mjiZmRBBPvGfNplOmxkBUdZOqTlbViar6qGPdj1R1Xa99HlHVhy44bpuqZqlqjuP1T57W7kraO7
v5ackoDgRlklj854vWyPJ2bp6RQnxEML97x5o3hU+yfy2cq4B5XzdbidM8veMEj3esIL693KjlZWF
CAv257/JxvH2w1pJFSa0SRPdpNuw7Sc25NjrnfA0aTjjVic0bCA7w5wtXjueDw6cprPDNTmyWQt
XIvIqeCGmLzFbjFG2dXTy57RiNE5YY32fb/5gtyWnuvSyV0EB/nnjviNlSLhnbgJiMqvKH94+SFh9
OxnV3wMixn8pztyp3zx1LRHAAf/zAejeFz1G+Ayp3wWVfAz9r3/Jr95yktrGNB66ZDHO/anyvinyz
ZTnFqBFB3D57DOsKKqlq+GypFm/G2n9NPsC2sjr2V53jS1eNR/wDYM5X4MQ2OLnXbGlOER
ESyKqZo9lUWMWpxlaz5QxvPvothIyE3LvMVuIU3d3KH94/QnpSJFdMioHcOyEowphVb3G+eO
V4urqVJ7cdM1vKJWEbEJP5w/tHiA0Pli/XMYdyxr0QFG5MlrI4n5uXSkeX8uzHvtfK0zKcOWY0L5
t5PwRZN7sP4MOy05SeajIetkSMZlPT74HiV6HR2mnjY6LDuCE9kRd3ltPaYZ1KDrYBMZEjtU28
c7CWey5LJSTQ31gZEgW5d0PhamisMVegk0yIC+eayXE88/EJ2jvtIoum8PETlH4w5wGzlTjN3z
46TvSIIJZlJ/195ZwvQ3enUZre4nxuXipnWjrYsM86Va1tA2Iiz3x8ggA/4a65Yz+9Ye5XfOamuP/yc
dQ2tlm+5o8laW+GPX+D9JUQ1WeVIMtQefY8b+yv4fbZYwgO8P/7hpiJMHmRca90tpkn0AXMm
xjDpPhw/vrRMbOlDBrbgJjE+fYuVu+qYFFm4mfr+HxyU/zJ6BpnYa6ZHMe4mDCesphv1ycoXG0
UTZzzZbOVOM2zHx8HjOSMzzD3K9Bca7iyLIyI8Ll5qeyraGBv+dmLH+AF2AbEJNbvO0nD+Q7
uvSy17x1mf8m4KQ5s8KwwF+PnJ9w7bxy7jp+hqNJO6fUYqsYDSHw6jLH2xMG2zi6e31HO9VM
TGD2qjxpxE66D2Ck+ETe8aXoKI4L8LTMKsQ2ISTyz/Thp8eHMHR/d9w4TrzdSenf9xbPC3MCts
0YTFuRvuQwTS3NyN1QVwKwveH/DqIuwubCauuZ27p3Xz8OWiPHAdXKPsViYnuzFDQVV1D
V5v0vONiAmsK/iLAUVDdxzWaqRTdIXfv4w4z44+h6cPuxZgS4mMiSQldNT2OAYddl4gJ1/hsAR
kH272Uqc5m/bjzMuJoyrJsX2v1P2bRAQCrue8pwwN3HvZam0d3Xz/M7yi+9sMrYBMYGntx8nN
NCfm2ZcJLA5/V7wC/CJUcids8fS2tHN2r12rxC3c/4MFL0MWbdASKTZapyi+GQDu46f4Z7LUvH
zG2AkFToSMm+GwpegrclzAt1AWkIE8ybE8NyOE17fsdA2IB6moaWDdQUnWTk9mciQwIF3jkiA
KUth77PQYe3JeFmjo8hIjuS5HeV2lV53U/ACdJ6H2V80W4nTvLCznKAAP26ZOfriO8+8H9qbDO
Npce6YM4aKM+e9viCpbUA8zJq9lbR2dHP33H78uRcy6/NGG0+LF40DuGPOWPZXnWOfXR/Lf
agaKa0pM63T77wfWju6eHVPJYszEhkZNoiqu6NnQ9w0o2y9xVmUkUhUaCAv5Hu3G8s2IB7m
hZ3lZKZEkpkSNbgDxl8Lo8b5hBsrLzeZ0EB/nt9pz0x3Gye2w+mDMPPzZitxmteKqmls7eSO2W
MuvjMYwfSZ9zsSCPa5VZu7CQn056bpKbxeVM2ZZu9N5bcNiAcpqmygpOoct80a5A0BRvG7mf
fD8Q+h9pDbtHmCyJBAlmUnsW7vSZrbrF2y3mvZ87RRCifjJrOVOM3zO08wNjqMyybEDP6g7N
sgIAR2Wz+YfvvsMbR3dfPqHu+NG9oGxIO8lG/4c/NyLnFWcM5dIP6w92n3CPMgd84ZQ3N7F+
sLTpotxfdoazIm02WshOBws9U4xfG6ZrYfqee2WaMHDp5fSFi0MfN+34vGTHwLMy0pkpwxI3lhp
/fGDU01ICKyWEQOishhEXmoj+33i0itiOx1LF/qte0+ESl1LPd5Vvml09rRxZq9J1mckUhU2EWC5
xcSkQBpNxjBUYs3m5oxdhRp8eE8Z4EURctRsgY6mo3sPYvzYn45fgK3zLyE0XoPM+8zZuD7Q
txw9hgO1jR67cx00wyIiPgDjwNLgHTgThFJ72PXF1Q117H80XFsNPAwMBeYAzwsIqM8JH1IbC
mpoeF8B7cP1p97IdPvhqZqKHvLtcl8jIhw++wxFJSf5VCN9TqweTV7noaYSZafed7Z1c1L+RVcO
yWexKiQix9wIWPnGXHDvc+4XJunWZ6TTFiQPy946QOXmSOQOcBhVT2iqu3A80DeII9dBGxV
1XpVPQNsBRa7SadLeCm/nJSRocy7FH9ub9IWQViMT7ixVk5PIcBPeHlXhdlSfIfTh+HER0Z5c4
vPPH/3UC2nGtuG/rAlYlS0PvoenLV2wkZ4cAA3ZiexvuAkLe3e530w04CkAL3NaoVj3YWsEpF9Ir
JaRHr+ogZ7LCLygIjki0h+bW2tK3RfMhVnWvjg8GluvVR/bm8CgiDrNji4GVrqXSvQw8SGB3PtlD
he3VNJZ5dd5t0l7H3GiJPl3Gm2EqdZvauC2PAgrp8aP/ST5NxhvBa84BpRJnLLTCNu+LoXVrT2
9iD6emCcqmZjjDIuObVCVZ9Q1VmqOisuLs7lAgfDy7uMLIpBTYYaiOl3Q1e7UWXV4qyaMZpTj
W18cPi02VKsT3cXFDwHkxZARKLZapzibEs7b+4/xYqcFAL9nfh5GjkWxl9tGFYvDUAPllmpoxgT
HfrJ74g3YaYBqQR6j1FHO9Z9gqrWqWpPRbE/AjMHe6y3oKq8sqeCeRNi+q4keikkZkFitk+4sa6f
Fk9UaCAv7/bK/zZrUfYWNFYZ7iuLs2FfFe1d3dx8sTI/gyHnLjhz1JgbY2H8/ISbpo/mw7LTXtcz3U

wDshNIE5HxIhIE3AGs672DiPRqPcYKYL/j/evADSIyyhE8v8GxzuvYfeIsx+tauGm6ixr6TL/HqLJa
XeSa85lEcIA/ebnJbCmu5lyrXWDRKQqeh9BRMNmrw4CD4pXdFUxJiCAj2QU1vNJXGHNifCCY
vmpGCqqwZo93pb+bZkBUtRP4BsYP/37gRVUtFpEfi8gKx27fFJFiESkAvgnc7zi2HvgJhhHaCfzY
sc7reGV3BSGBfizJSrr4zoMh61bwC4R9z7vmfCayasZo2jq72WihFp5eR1sjHNgIGTcbcTILc/R0M
7tPnOXmGSn9V6m+FIJGGHNCitdAe4vz5zOR1JgRzEodxcu7K7xqToipMRBV3aSqk1V1oqo+6l
j3I1Vd53j/fVXNUNUcVb1OVQ/0OvbPqjrJsXhlnY+2zi427KtiUUYi4cEBrjlpWLQxJ6RwtUhlsUcAA
CAASURBVOH7tjDZo6OYFB/Oajsba+jsX28UTvSBsu2v7q7AT4wsPZeRexe0N1q+MRvAzTNG
c/hUE4Ve1JjN24PolubtA7U0nO9wnfuqh+zbDJ/3sfdde14PIyKsmjGaXcfPcPS0tWcNm0bB8zBq
PIyZY7YSp+juVl7ZU8kVk2JJiBzC3I/+GDsPosbCPutnYy3LTiIowI9XvChuaBsQN/Lqngpiw4O5cq
BGOENh8mIIjjTKNVicldOTEcHuEzIUGiqNuQ7Zt1t+7sfOY/VUnDnPqhlOZipeiJ+f0Rel7G1oMie
N31VEhQayMD2BtXsrae/0jvR324C4ibMt7bx14BR5uckEOJOO2BeBIUaAsGSd5X27SVGhzB0f
zdq9J73Kt2sJilYDaoxILc4ruysZEeTPDRkJrj959u2gXVD8iuvP7WFWzUjhTEsH7x7yDmNoGxA3
sWFfFR1d6nr3VQ/Ztxu+3UOb3XN+D7IyN4Wjp5vtPiGXSsELRg+MmIlmK3GK1o4uNhVVsTgzi
bAgF8UKexM/1UiB9wE31lVpcYwKC2SdlxQjtQ2Im1izp5LJCeGuSUfsi9QrITLFJ9xYS7KSCPL3
Y+1e77gpLEF1IZwq9ong+TsHa2ls7SQvN9l9F8m6DSp3QV2Z+67hAQL9/ViWncTWkmqavKAlg
m1A3EB5fQv5x8+Ql+uidMS+6PHtHn4Dmq09mzsqNJDrpsaxft9Jury8B7TXsO8F8Asw0nctzrqC
SmLDg7h84hDrxA2GrFsAMXqmW5y83BRaO7rZWmJ+aRPbgLiB9fuMJ+kVOW58ogLj6bO70+
gBYXFW5qZQ29jGtjJrG0OP0N0NRa8YpUtGuPFH1wM0tnbwxv5T3JjthlhhbyKTYfxVhuG1eKxt5
thRpIwM9YoRu21A3MC6vSeZMXYkY6KdLF1yMRIyID7DJ9xY102NJyI4wOtm2noI5dvhXCVkrjJ
bidO8XlxDe2c3K9zpvuoh6zaoPwKVu91/LTfi5yesyE3m/dLT1DW1XfwAd2ox9eo+yKGaRg5UN7
p/9NFD1iqo2GH5stUhgf4szkzk9eJqWjusPUHS7RSuhoBQmLLUbCVOs3ZvJWOiQ5k+ZqT7L5a
+AvyDodD6D1x5ucl0dSubCs2t4mAbEBezbu9J/ASWZXvIgPQ8hRZZP0Vx5fQUmto6eWN/jdISv
JeuDqPz4JTFlm9bW9vYxoeHT5OX48ZYYW9ComDyDYbL1+JVHKYmRjIIIYI1JruxbAPiQISVd
QUnuWJSLHERwZ656KhxkDLLMSfA2lw2IYa4iGA2FNi1sfrlyLvQUgeZt5itxGk2FVbRrbg3++pC
MldBUw0c/9Bz13QTK3KT2XX8DOX15s0Fsw2IC9lbfpYT9S0s95T7qofMVUZaZ+0hz17Xxfj7Cc
uyknjr4Cka7Qq9fVP0MgRHQdpCs5U4zdq9lUxNjCAtIcJzF01bBIEjjH9Hi9PjJjdzTohtQFzIuoKTB
AX4sTjTw019Mm4CxCduiuU5SbR3drO1xHZjfYaOVqMo4LTlEOChEa6bKK9vYfeJs54JnvcmKA
ymLoWStYY70MKMiQ5j+tiRbDCxmrVtQFxEV7eyYV8V102JlzIk0LMXj0yCcVcaBsTiKYrTxxgpi
mbeFF5L6RZoO2ckTlicjY7g73JPxQp7k7kKzp+BI+94/tou5sbsZPZXnaOstsmU69sGxEXsOFpP
bWOb591XPWTeDHWlhivLwvj5Ccuyk3jvUC1nW9rNluNdFK2GEXEw7mqzlTjNhn0nyRnjgVT3v
ph4vRFQ94ER+7KsJEQwLW5oqgERkcUiclBEDovIQ31sf1BESkRkn4i8KSKpvbZ1ichex7Luwm
M9zYZ9JwkN9Of6qfHmCJiWZ8xM9oFg+vLsZDq7ldeLzZ9p6zW0NcGhLUaDJH831IvyIMdON1
NUeY7l2S5qsnapBAQbbsD9Gwy3oIVJjAphdmo0G/aZEwcxzYCIiD/wOLAESAfuFJH0C3bbA8x
S1WxgNfCzXtvOq2quY1mBiXR2dfNaUTXXT4t3TzG4wTAiBiZcZ6TzWtyNlZkSSWpMGOvtbKy/
c+g1o3FUpvVLl/T82C11VZfOoZC5yihGenireRpcxI05SZSeauJgdaPHr92vARGR34jl//S3uODac
4DDqnpEVduB54G83juo6tuq2pOjth1wcbMA17D9SD11ze3mPVH1kHkzNJQbReMsjIiwPDuZb
WWnOW3yTFuvofhVCE+EMZeZrcRpNuyrYlbqKJJHhponYtzVEBbrE26sJZlJ+AmmjEIGGoHkA
7sGWJwlBSjv9bnCsa4/vgj0rl0eIiL5IrJdRFb2d5CIPODYL7+21j019DcWnmREkD/XTjHJfdXDlKV
Gv3QfqI11Y04S3QqbTZ5p6xW0noPSrZCx0iiiaWEOnzIqNdxo9sOWfwCk58HB16Dd2t0w4yKC
mTcxhg37qjzeU6ffv0ZVfar3Arx0wWePISL3ALOA/+q1OlVVZwF3Ab8SkT6bIqjqE6o6S1VnxcXF
uVxbR1c3m4uqWZCeQEigv8vPf0mEjoRJ86F4jVFwz8JMSYggLT6c9XY2luG+6mrzicq76wuqE
DHZfdVDxk2GW7B0i9lKnObG7GSOnm6m+OQ5j173oo8zIjJPREqAA47POSLyvy64diUwptfn0Y
51F15/AfBDYIWqfuLPUNVKx+sR4B1gugs0XTLbyuo429LBjWakI/ZFxk1wrgIq881W4hQiRjbWz
mP1nDpn7UCn0xS9YvR+GT3bbCVOoaps2HeSueOjiXdl3/Ohkno5jIj3iRH74oxEAvzkk0rgnmIw4

+FfAYuAOgBVLQBckUe4E0gTkfEiEgTcAXwqm0pEpgP/h2E8TvVaP0pEgh3vY4ErgBIXaLpkNh
ScJCI4gKsnu7jv+VCZsgT8g3zipliWlYQqbC4axtlY589C2ZtG9pXF3VcHaxopq232XJ24i+HnbxR
YPLTF8m6sUSOCuGJSLJsKPevGGtRfpKqWX7DK6UpkqtoJfAN4HdgPvKiqxSLyYxHpyar6LyA
ceOmCdN1pQL6IFABvA4+pqscNSHtnN68XV7MwI4HgAJPdVz2ERMGkhT7hxkpLiGByQvgnk8
6GJQc3QVe7o9qAtdm4rwo/gSWertQwED1urEOvm63EaZZlJVFef57CSs+1hh6MASkXkcsBFZ
FAEflnjB98p1HVTao6WVUnquqjjnU/UtV1jvcLVDXhwnRdVd2mqlmqmuN4/ZMr9FwqHx4+zbnW
TvMDgheScRM0njTKvFucpVnD3I1V/CpEjYHRs8xW4hSqysbCKi6bEENsuBeVYRk7D8ITfGLEf
kNGAgF+4tEHrsEYkK8CX8fIkDoJ5Do+D3s2FlYRERLAlZNcH5x3iimLjb4HPnBTDGs31vkzUPa
2kS3kiXLnbuRgTSNHapu9I3jeGz9/49+3dIsxWdPCjAwL4nIPu7EuakBU9bSq3u0YCcSp6j2qWu
cJcd5Me2c3W4qrWZieQFCAl/mmgyOMaq0la203lpU5sAm6O3wi+2qTw33l8UKjgyF9JXS2Qqn1
3Vg3etiNNZgsrAkisl5EakXkIlisFZEJnhDnzXxYZrivlnnbE1UPGTdBY5XR/tTiLMtKHp5urJI1EDU
WUmaYrcQpetxXc8d7mfuqh7GXGZM0fWDE7mk31mAenZ8FXgSSgGTgJeA5d4qyApv2VRER
HMCVaV6SfXUhkxcZbqyStWYrcZpI2YnDz411/qzDfbXC8u6rQzVNlNU2s9TbYoU9fOLG2gptni8
H4ko87cYajAEJU9W/qWqnY3ka8IlkbvPo6OpmS0kNC9O9KPvqQoIjYNICKFlneTfWpPgIpiREs
HE4TSo8uNlwX6X3W2TBMmwsdLivMrzQfdVDRo8by/qTCpdlJVJef56iSvdPKhyoFla0iEQDm0X
kIREZJyKpIvI9YJPblXkxHx4+TcP5Du8LCF5IxkojG8vikwrBkY11fBi5sUrWQORon8i+2lRYxZzx0
Z5r8zwUxsx1ZGOtMVuJ09yQbkwq3FDo/kmFA41AdmHUw7oN+ArGfIt3gK8Bt7tdmRezubCaiO
AArvKWyYP9MXmRY1Kh9W+KHjfWa8OhxHtrA5S95RPZV4dqmjh8qsl7Y4U9+PnDtBWGG8sH
JhV6yo01UC2s8ao6wfF64TJsg+gdXd28XmLUvvJa91UPIVEwcb4RB7F4ifdJ8UZtrE3DIRvr4Gv
G5MH0vIvv6+VsKjRqXy3yxuyrC0nPc9TGsn6J9x43lrtrYw0q/1REMkXkNhH5XM/iVlVezEeO2lde
NZt2INLzHLWxrF3iHWBJVtInnR99mpI1EJFs+dpXAJuLqpgzLpr4CAuETVMvN0q8l1h/xL4wPRF
/P3H7A9dg0ngfBn7jWK7DaOpkagMnM9lcVEV4cABXT/ayyYP9MWWJUeLdB26KpVmJdCu+3a
mw9RwcftMw/BavfXX4VCOHapq8P1bYg5+/0anw0BZob7n4/l5M9Igg5k2IcbsbazB/obcA84FqVf
08kANEuU2RF9PZ1c3rxTXMnxZvfun2wRI6EiZeB8XWd2NNSYhgQtwINhf5sBvr0OtG6XYfcF9t
LjQMvVdOHuyPjJXQ0WwUsLQ4S7ISOVbXwgE3diocjAE5r6rdQKeIRAKn+HQZ9mHDjqP11De
3syTTIk9UPaTnQcMJOLnbbCVOISIszUzio7I66ny1U2HJGkfnwblmK3GaTUXVzEodRYI3lG4fLK
lXQmi0TySeLMpIxE/c25RtMAYkX0RGAn/AyMzaDXzkNkVezKaiKkID/bnGKu6rHqYsBb8AY06Ix
VnicGNtKakxW4rraWuCw28Ykwct7r46erqZ/VXnWGIV91UP/gEw7UajiVeHtVPGY8ODmTs+hk1
unIA7mFpY/6CqZ1X198BC4D6HK2tY0dWtvFZUw/VT4wkNsoj7qoewaBh/jU9kY6UnRZIaE+ab2
VilW4zJbL7gvnK4GS3lvuohfSW0N/mEG2tpViKHTzVRWuMeN9ZAEwlnXLgA0UCA4/2wIv9YPa
eb2liSZcEbAowfpTNHobrQbCVOISIszUpiW1kdZ5rbzZbjWkrWwog4o8S4xdlUWEXumJGkjAw1
W8qlM/5qCBnpEyP2RRmJiMCmQveMQgYagfz3AMvP3aLGi9lcVE1wgB/XTYk3W8rQmHojiL9P
1MZamplEV7eydb8PubHaW4wRyLTlRjaQhTlR10JR5TmWWvVhyz/QuF8OboZOa8fa4iNDmJ0
a7bbEk4EmEl43wHK9Ky4uIotF5KCIHBaRh/rYHiwiLzi2fywi43pt+75j/UERWeQKPf3R3a1sLqri2il
xjAgOcOel3MeIGBh3pRGktbgbKzMlktGjQt0aHPQ4ZW9CR4tPua8sl2zSm/Q8aGuAI++arcRplm
QlcqC6kbJa1/c7MS1SJyL+wOPAEiAduFNE0i/Y7YvAGVWdBPwS+Knj2HSMHuoZwGLgfx3ncw
t7ys9Qc67NOvns/ZGeB3WH4ZRLGkqaRo8b6wNHTTKfoGStkf2TeqXZSpxmU1E1mSmRjIkOM
1vK0JlwDQRH+sSIfVlWEv/v5iy31CIzM9VjDnBYVY+oajvwPHDh41ce8JTj/WpgvoiIY/3zqtqmqke
Bw47zuYVNhdUE+ftx/VSLuq96mLYcEJ+4KZZkJtLRpbzpC26sjlajfMm0G40sIAtTcaaFgvKz1n/Y
Cgg2JuEe2ABd1n5IiY8M4c45Y4kMCXT5uc00IClAea/PFY51fe6jqp1AAxAzyGMBEJEHRCRfR
PJra2uHJPR8RxcL0uOJcMN/gEcJj4fUK3zCgOSOGUlyVIjbgoMe5cjb0N7oE+6r1xwpo5Z2X/W
QngetZ+Hoe2Yr8VoGU8rkChEZ4Xh/j4j8QkRS3S/NNajqE6o6S1VnxcUNbf7Gf96UxeN3+UjiWfo
KqN0PtQfNVuIUIsLizCTeK62lqa3TbDnOUbLWKHw57mqzlTjN5qJqpiVFMj52hNlSnGfi9RAUDvu
tn43lLgYzAvkd0CIiOcB3gDLgry64diWfntE+2rGuz31EJACjhErdII91KWLxstqfMG258eoDo5ClW

Ym0d3bz1oFTZksZOp3tRu/zqTdCQJDZapyiuqGVXcfPsNSKcz/6IjDUaImwfwN0WfwhxU0MxoB
0qlGNKw/4rao+DkS44No7gTQRGS8iQRhB8QtN/TrgPsf7W4C3HFrWAXc4srTGA2nADhdo8n0i
k40yGT6Q4z5j7CjiI4KtnY115B0j28cn3FeO7Curxz96k54HLafhxDazlXglgzEgjSLyfeAeYKOI+AF
OBwMcMY1vAK8D+4EXVbVYRH4sIj3Vfv8ExIjIYeBB4CHHscUYfdpLgNeAr6tql7Oahg3pK6Gm
EOrKzFbiFH5+wuLMRN4+eIqWdos+IZasNbJ9JlxrthKn2VxUzeSEcCbFh5stxXVMWgiBYT5RG8
sdDMaA3A60AV9U1WoMd9F/ueLiqrpJVSer6kRVfdSx7kequs7xvlVVb1XVSao6R1WP9Dr2Ucdx
U1R1syv0DBt8yI21JDOJ1o5u3jk4tAQJU+nqMLJ8piwxsn4sTG1jGzuO1ftG8Lw3QWGQthD2r4d
u+xn1QgZTC6taVX+hqu87Pp9QVVfEQGzMYuQYSJnlEz1C5oyPJjY8yJq1sY6+Z2T5+ID76vXia
lSxfvpuX6TnQfMpOLHdbCVex0C1sD5wvDaKyLleS6OIuLdPoo37Sc+DqgKoP2q2Eqfw9xMWZ
STy1oFTtHZY7AmxZK2R5TPRJYUdTGVzURUT4kYwOcGH3Fc9pC2CgBCfGLG7moFKmVzpe
I1Q1cheS4SqRnpOoo1bSHeEmXwgRXFpVhIt7V3WcmN1dRruq8mLjGwfC1PX1Mb2I/UsyUz0n
WzF3gSHw6QFxr3S3W22Gq9iMPNAFvSx7r6+9rWxEKPGQVKuTzxVzR0fTfSIIGt1Kjz+IbTUG
QkNFuf14hq6utU33Vc9pK+ExiqosJM9ezOYIPqPROR3IjJCRBJEZD2w3N3CbDxAeh5U7oKzJ8
xW4hQB/n4sykjgzf0WcmOVrDWyeyZ95vnMcmwuqmJcTBjpST7smJi8CPyDfeKBy5UMxoBcgz
F5cC/wAfCsqt7iVlU2nqEneOsDc0KWZCbR1NbJ+6WnzZZycbq7jKyetIVGlo+FqW9uZ1tZHUuzk
nzTfdVDSCRMmm8YENuN9QmDMSCjMAoVlmGk86aKT/+lDCNiJkJilk9kY82bGMPIsEBrZGMd
32Zk9fiA+2pLcbXvu696SM+Dc5XGqN0GGJwB2Q68pqqLgdlAMvChW1XZeI70PKjYCQ0VZitxik
B/P25IT+CNkhraOr3cjVWyBgIcZTIszqaiasZGh5GR7MPuqx4mLwa/QJ944HIVgzEgC1T1zwCqe
l5Vv4ljRriND5B+k/HqC26srCQa2zr5wJvdWN1dxr912kIIsnbBwbMt7Ww7fNr33Vc9hI40Uq5L1lq
+KZurGMxEwhMiMkpE5ojI1SJi/ZKhNn8ndhIkZEHxq2YrcZorJsYSGRLARm92Y534yHBfZfiC+6
qGzm5l2XBwX/WQsRIayqFyt9lKvILBpPF+CXgPo2bVvzteH3GvLBuPkpFnpCda3I0VFODHDR
mJbPVmN1bxGmNSWpovuK+qGD0qlMyUYeC+6mHKUsONVfyK2Uq8gsG4sP4JI/ZxXFWvA6Y
DZ92qysaz+JAba1l2Eo2tXurG6u4yJqOlLTQmp1mYhpYOPjx8mmXDxX3Vg+3G+hSDMSCtqto
KICLBqnoAmOJeWTYeJXYSJGT6RHDQq91YJ7ZDU41vZF+VVNPRNUyyry4k4yaHG8vOxhqM
AakQkZHAGmCriKwFjrtXlo3HSV8J5R9Dg1v7crmdoAA/FmUksrXYC91YJQ731eTFZitxmg37qh
gTHUr26CizpXieKUscbizrxw2dZTBB9JtU9ayqPgL8G0aPDus/Qtl8mp6grg/UxlqWbWRjvX/Ii9xY
3d2Gi3DSAsu7r840tzvcV8nDy33VQ+jIv08qHOZurMGMQD5BVd9V1XWq2u4uQTYmEZtmuLF
84KnqikmxRIUGepcb68RH0FRtuD8szpaSajq7lRuzh6H7qgfbjQVcogFxFSISLSJbRaTU8Tqqj31
yReQjESkWkX0icnuvbU+KyFER2etYcj37DXyUT9xY1s7GCnTUxnqjpMZ7amMVv+KYPOgb7qv
UmGEyebA/piwB/yCfeOByBlMMCMZExDdVNQ14k74nJrYAn1PVDGAx8CtHLKaH76pqrmPZ63
7Jw4DMm41XH2jfuSw72XBjeUM2Vlen4e6YvMjy7qu6pja2ldUNv+yrCwmJgonzjXtlGNfGGsw8k
H/sa4TgJHnAU473T9FHTEVVD6lqqeP9SeAUEOdiHTa9iZkIidk+keN+uaM21sZ9J82WAsc/gOb
avxtoC9NTun3ZcHZf9ZBxE5yrgMp8s5WYxmBGIAnAThF5UUQWu6iQYoKq9jioqx3X6BcRmQM
EYRR07OFRh2vrlyLSb0NpEXlARPJFJL+21kINh8wic5Xh1z1zzGwlThHo78eidGNSoelurKJXIH
AEpN1grg4XsLHwJBNiR/h26fbBMmWJUeK96GWzlZjGYLKw/hVIw8i+uh8oFZH/FJGJAx0nIm+I
SFEfy6caQKuqAv2mMohIEvA34POq2jNW/D4wFWOCYzTwLwPof0JVZ6nqrLg4ewBzUXqCvEX
WH4Usz0mmub2Ltw+cMk9EV4eR2TZlieU7D55uauOjsjqWZQ9z91UPIZEw+QYjDtLtJbE2DzOo
GIjjR77asXRilHhfLSI/G+CYBaqa2ceyFqhxGIYeA9HnHS4ikcBG4Iequr3XuavUoA34C0a5eRtXM
CoVUmb5hBvrsgnRxIYHsd5MN9aRd+H8GZ9wX20uqqZbGZ6TB/sjc5UxOfT4NrOVmMJgYiD/J
CK7gJ9hlHHPUtWvATOBVUO87jqgpy3ufcBn2nyJSBDwKvBXVV19wbYe4yMY8ZOiIeqw6YvMV
VBdCKdLzVbiFAH+fizLSuLN/adoaus0R0TxKxAc6ROdB9fvPcmk+HCmJkaYLcV7SFtkuCeHqRt
rMCOQaOBmVV2kqi+pageAw5104xCv+xiwUERKgQWOz4jILBH5o2Of24Crgfv7SNd9RkQKgUI
gFviPIeqw6YuMlYD4jBurrbObN0pqPH/xzjbYvwGmLoOAfsN0lqCq4Tw7jtWzImeYTh7sj6Awwz1
ZstZwVw4zBhMDeVhV+yxdoqr7h3JRVa1T1fmqmuZwddU71uer6pcc759W1cBeqbqfpOuq6vWq

muVwid2jqk1D0WHTD5HJMHaeT7ixZowdRXJUCOsLTHBjlb0FbQ2QYX331YYCI+dlRU6yyUq8
kMxVcL7ecFcOM8yaB2Lj7WTeDLUHoKbYbCVO4ecn3JiTzHultZxt8XABhcKXIDQaJl7n2eu6gX
UFJ8keHcW4WGs3wXILk+ZDcJRPPHBdKrYBsemb9JUg/IC4+uL7ejnLs5Pp6FJeL6723EXbmu
DAJsMd6B/oueu6gaOnmymsbLBHH/0REAzTboT96w235TDCNiA2fRMeBxOuNQyIxQvGZaZE
Mi4mjPUFHqyNdXAzdJ6HrFs9d003sW7vSUTgxmzbgPRL5s3Qdg5Kt5qtxKPYBsSmf7JuhYYT
UL7DbCVOISIsz0lmW9lpahs99IRY+BJEpsCYyzxzPTehqqwrqGT2uGgSo0LMluO9jL8WwmKh8
EWzlXgU24DY9M+0G43+FYUvma3EaVbkJNOtsMETc0Ja6qHsTSO46mftW2x/VSNltc22++pi+
AcYo5CDr0Frg9lqPIa1/7pt3EtwhJGiWPyK5VMU0xIiyEiOZM0eDzTMKlkD3Z2+4b4qOEmAn9iT
BwdD1m3Q1WbEQoYJtgGxGZis26ClDo68Y7YSp1mZm0JBRQNHat2c9V24GmInQ2KWe6/jZr
q7lXV7K7kyLZboEUFmy/F+Rs+CUeN9YsQ+WGwDYjMwkxZAyEifuCmW5yQjAmv2utGN1VAB
xz80Rh8Wn3D38dF6Tja0ctP0FLOlWAMR4//96HvQ6MGMPxOxDYjNwAQEQXqeMaO6vdlsNU
6RGBXC5RNjWLu3EnVXZllP2nPmUKv8eA9r9lQyIsifG9ITzZZiHbJvA+0eNqVNbANic3Gyb4OO
ZiM11eLk5aZwvK6FPeVnXX9yVdj3AoyebfRWsTCtHV1sKqxicWYSoUH+ZsuxDrFpkJQL+4ZHN
pZtQGwuztjLjZTUfS+YrcRpFmcmEhTgx1p3BNOrC+FUCWTffvF9vZw395+isa3TdI8NhezboGqv
5YuRDgbbgNhcHD8/40fx8JvQaEJRQhcSGRLIwmkJbNhXRUeXi1uRFjwPfoE+4b56dU8lCZHBz
JsYY7YU65G5CsTPJx64LoZtQGwGR86doF0+EUzPy02mrrmd90td2KGyq9OYRDZ5EYRFu+6
8JlDf3M47B0+Rl5uCv5+1EwFMISLRqOJQ8LzP90u3DYjN4IibDCkzoeA5s5U4zbVT4hkVFsjLu1
3oxip7y+h7nnOn685pEhv3naSzW1mZa7uvhkzOXdBQDsc/MFuJW7ENiM3gybkTaooMX7+FCQ
rwIy83ha3FNTS0uGiC5L7nIXSUT/Q9f3VPJVMTI0hPtvueD5mpy4xGYnut/8A1EKYYEBGJFpGtl
lLqeB3Vz35dvZpJreu1frylfCwih0XkBUf3Qht3k7nK8PH7wE1xy8zRtHd1s84VpU1aG+DARuPfJ8
Daf4pltU3sPnHWDp47S1CYUYm5ZK1RmdlHMWsE8hDwpqqmAW86PvfF+V7NpFb0Wv9T4Je
qOgk4A3zRvXJtAMO3P3mR4evvMqlFrIvISI5kamIEq/PLnT9ZyVrobPUJ99VL+RX4+wk3zbANiN
Pk3m2kv+9fd/F9LYpZBiQPgLXpRAAAHflJREFUeMrx/imMvuaDwtEH/Xqgp1HFJR1v4yS5dxm+/
rI3zVbiFCLCLTNHU1DRwKGaRudOtvc5iJlkxlgsTGdXN6/sruC6KXHER9iVd51mzFyIngB7nzVbi
dswy4AkqGpPc4ZqIKGf/UJEJF9EtotIj5GIAc6qas8jcAVgPy55ikkLjS57PnBTrJyeQoCf8PKuiqGf5
PRhOLHNeNq0eOmS90prOdXYxq2zxpgtxTcQMUalx96HsyfMVuMW3GZAROQNESnqY8nrvZ8
aNSX6qyuRqqqzgLuAX4nIJU/vFZEHHEYov7bWhWmbw5WAIGNOyMFN0FxnthqniA0P5top8by
yp5LOoc4J2fM3o3Nj7l2uFWcCL+VXEDMiiOunxpstxXflucN4LXjeXB1uwm0GRFUXqGpmH8tao
EZEkgAcr6f6OUel4/UI8A4wHagDRopIgGO30UC/+Ziq+oSqzlLVWXFxcS77fsOaGfdCV7tPTJS6
ZeZoahvbeL/09KUf3NVppDVPXmTk/luY+uZ23thfw03TUwj0t5MzXcbIsTD+atj7jE/OCTHrL2Udc
J/j/X3A2gt3EJFRIhLseB8LXAGUOEYsbwO3DHS8jRtJyICUWbD7Kcu3u71+ajzRI4J4YecQguml
W6CpBqbf63phHmbNnko6utR2X7mD6Z+DM8fg6LtmK3E5ZhmQx4CFIIlKLHB8RkRmicgfHftMA
/JFpADDYDymqiWObf8CPCgihzFiIn/yqHobmPE5qD0AFTvNVuIUQQF+rJqRwhv7azjV2HppB+/
+K4QnWH7uh6ryYn45OaOjmJlYYbYc32PacmOO0O6nLr6vxTDFgKhqnarOV9U0h6ur3rE+X1
W/5Hi/TVWzVDXH8fqnXscfUdU5qjpJVW9VVQ81urb5hMybIXCET9wUd8wZS2e3svpSgumN1c
YIJOdOo52phSmsbOBAdSO32KMP9xAYYvyd7N8AzUNwlXoxtrPTZmgERxhGpOgVaD1nthqn
mBgXztzx0Ty/o5zu7kG65PY+a9QG8wH31TPbTxAa6E9ert333G3MuA+6O3wie7E3tgGxGToz7
4eOFqNnusW5a+5YTtS38GHZIJ4Qu7thz9OQegXETnK/ODfScL6DdQUnyctNJjIk0Gw5vkv8VB
hzmU/EDXtjGxCboZMyE+LTYZf13ViLMhIZFRbIczsGka9/9F2oLzOeKi3Omj2VnO/o4u65qWZL8
X1m3gd1h42Wxz6CbUBsho6I8SN6cjec3GO2GqcICfRn1YzRbCmuobbxIiG1nX+EsBij1pGFUV
We+fg42aOjyBodZbYc3yd9JQRHwa4nzVbiMmwDYuMcOXdAYBjs+OPF9/Vy7pxrBNNf2jVASm
9DhTGJcsbnICDYc+LcQP7xMxyqaeLuuWPNljI8CAozuhWWrPOZYLptQGycI3SkMTO9aDW01
Jutxil6gunP7ThBV3/B9Py/GD7sWV/wrDg38Mz240QEB7A8xw6ee4zZX4SuNp/IXgTbgNi4gjlfNq
rR7vmb2Uqc5nPzxlFef563DvRRHKGz3bjxJy82ZhhbmPrmdjYVVnPzjBTCgqydhmwp4qcZM9N

3/tnyFa3BNiA2riAhA1KvNGID3V1mq3GKRRkJJEWF8JcPj3524/51RiXi2V/yvDAX88LOctq7urnL
Dp57nrIfhXMVcHCj2UqcxjYgNq5hzpeNiqOlW8xW4hQB/n7cOy+VbWV1HKy+oMz7zj/CqPEw8X
pzxLmIjq5untp2jCsmxdgzz82gZwT78f+ZrcRpbANi4xqmLoOIZNjxhNIKnObO2WMJDvDjyW3H/r
6yuhBOfGT4sP2sfdtsKqyi+IwrX7hivNlShid+/jD7y0Y6b3WR2Wqcwtp3go334B8Isz4PZW9B7SG
z1TjFqBFB3DQ9hVf3VHC2pd1Yue23EBRu+ZnnqsqfPzjKhNgRXDfFLttuGtPvgYBQ2PH/27v3s
KrK7IHj3wWCoGgoKhZewLQQRSjIS14iLE3Hqay8dZlKcxL1pTlzPSbcWa62TjI5DQ2qZU2UzIo2Xt
DGdGstSQw1QxAuZWioYGoMFoqJc1u+PfXAIIW7nnH2OvJ/n4Qn22WefdXbCOu/77r2Wd49C
TAlxnCfuPvBtDFtfsTuSeruvbziFRaUsTc6EvKPWVWZX/8K66syLpR7+np1ZedzfNxwfH+9ugOX
VmrS0LuINX+bVVy+aBGI4T1Abq7FS2jtw8rjd0dRLZNvm9OkUwuKkQ5RunQdaai1+ermFm7/hk
kA/bo9rZ3coRq+JUHwGkr23mLhJIIZzXTvVajbI5UNzgHH9IsjPO0FJ8hvWXcQtvPuKpcwTp/Iwzz
HG9OxgLt31BKFR0GUwbJsH507bHU2dmARiOFfI5Vb/g+QFcPZk9ft7sMTINkwN3oJfcQGIfaba
HU69vfH5IXxEuPda706EF5V+j8LpXKs4pxcyCcRwvr7ToDAPtnv3jYU+Wsw98gFbS7vycX6Y3e
HUS27BWd754gg3x1zGpZcE2h2OUaZjH6tKb9JcKCmyO5pas2UcKyItgXeBcOAQMFJVv6+wz/
XAS+U2RQKjVXWViLwJXAfkOR67T1XT6hJLUVERWVIZFBbWshvdRSogIIB27drh51eP0t7t4q0
bC7e8Yt0f4uuIZcJ3r6TJmWO8FziWLzcc4IaubRDxzoXnBZu/obC4hEnXe3f5+YtSv0fhnVFWb52
YUXZHUyt2TYTOAD5W1edFZIbj5yfL76CqG4BYOJ9wDgDI71KbrqrL6xtIVIYWzZo1Izw83Gv/O
DiLqpKbm0tWVhYREfW8R6Dvw/D2SNi1HGLHOCdAdyopho0vQJsooq8aybur97LIYC7Xdm5ld2
S19sPpcyxOOsTPoi+Ic5sgu8MxKuoyyGqLsPkIiB7hVfcZ2RXpLUBZNbFFQHV1se8A1qmq01ea
CgsLCQkJafDJA0BECAkJcc5orPONEBoNG18NG//knTV/di+3ejckzOCO+A60btaYv3160O6o6uT1zw9
x6IwJUxLN6MMj+fhA30cgJwP2f2h3NLViVwIJVdVsx/fHgNBq9h8NvFNh2zMiki4i4iL4IIIXW1RWSC
iKSISEpOTk5V+9Q07oue086Fjw9c/2ur8VL6Uucc011KiuGzWRDaHSJ/ToCfL+P+P7Rb
D5wH/ZceT
76p/vQfILi3jj828Y3C2UyLbN7Q7HqEr32yC4I2x41up46SVclkBEZL2I7K7k65by+6mqAlrNca4Ar
S4FooHxq/hXWmsg1QEsqTH9VOP5rqhqvqvGtW7euz1syauz1syauvKIXDZ1fDpLKuSrbfY9U848TUkz
Dg/nXBX746ENPVn9kf7bA6udhYnHeJkYTFTE7vYHYrxU3z9rA9cx9IhY7Xd0dSYyxKIqt6gqt0r+
VoNHHckhrIEUUnt7PNGAu+p6vILFFQ1Wy1ngTeAnq56H+7wzDPP0K1bN3r06EFsbCxbt26lpKSEmTNn0rlr4dbbLbl5+ooGPSrZRAsrnk9
x7N271+7Q6kcEEn8DmZ2kL47O6mZLkoLsAdLkHM0pyZXCBmFLmUK6AX1g2/QSUnOJjxjsssy6uIIyU
WAIOBrRzZ1dVdqmDrNtqtpmTmBCaXrXn/MVodmRA9iqPrN3TkT+d6H0ZW15fzv/hD2d38tTRZtwcL9jkEXBvdf4rYiyr69l6Ox/2bgaNBxt5uDOsZd38aSBarQmUNVBmKcpyAj4eN
W2wmTW1mkysQl4ZNSwv+DSn2EyIXVvvkCtkZO3
nOOWvAe5MUDwglUf9RqQ7b6RJr8kY5LaqJTWhQ6n16PM6WEmLNKpgqGHm+J3JNX6vq6IKhJKG9Fx6ntD2yyfUJhCdvWzbKsIa8sWBxuMbspZOIFaWoyTXW+F3Ko4+4+qxpHBDpAFsQk9Li
GZdCiSh2VoA+a4e69SzzmQpLmtqtxRkFbj4FrIUH35w1i
ubIb18NxOfpknTvPm54e47ap2RF1m1j68RuTPHNO+z1u1sjy8bbItIxBVzVXVgaraxTHVdcKaSV
UseTh+PqSqYapaWuH5iaoa7ZgUC9zk5F5Hw0pr9fHMY0ceAcAUHxJcRDe3m
vdu7cyYABA5g/fz4A06ZN49FHyHyU5OZkVK1YwfryHNjqK6A+A+dE
e5s2B/FG567oLRR
5IbrwrjytBm/Pmjrygq8dzFzj9/tA8ReHzwFXaHYtSGCAz8rdVwKuV1u6OplimIU85PjRRcJSgoiNT
UVDZt2sSGDRsSYNWRsYNWoUzz///Il/28ff3+8ff3Z9gwaz4+kBEgkdf6D34RxD+fn5FBQUbFRRVPFKfv8sTYZd8hRjj8ZCtN1QnQ2x7FAxjRUiXVbTaeX0fI4WyhCiTqs2/ziyGmzzF4Ckoo6iUQ17mDzZp+t4/4+iHpk66z+ZMLfLK1xWXr9Eu5b3HT4SQ2mffJXgqmFO5Tkm3bl3CgXBrxHJ+VVwhCj17M4lzTJzKOctdFN9oQUpdb76bC0pFqq8EWZVjRXHPpSwGqgcZUyyO
+cB9Du7
cIJMhzphIUlWfXZtCyqT8TEy63OxyjLkRgyCyY1xc+ngk3z7U7oip5zy2PF6l9+/axf//+8z+npaXRs
WPNPtUOGjSIuXP/948Pm948rLa1O1Vzc5/pfQWALWPsEqActQmd+AbuWwbVTaIRxV0T44y3dOXW2
mOfWfemGAGtuTXo2W77OZdrALjQP8NISMga06Qq9H4LtiyEz2e5oqmQSiM0KCgq49957
57iYqK
okePHuzdu5e5c+cfMcccS5YsIV
yNSVFfnAJlqU6mSXSPzt95q+Vl4M9yPsWVy4dyn9A6MrnXWy3z7hkFVrk2rbXYzUIZuS72S2
WxaaRhbdA8N2U03
+HTcWHmH3ddcaXqq3VY1r4TLt80dVHcxAcf8VKUbCcSyg+0fP2S4T+WrMNYjGhDMt0vgUz0FeXC
d5ztTd2u+Ckd0Z

KGBhwCDK/
oqQJ2uXKHYP+evdxr1syTaJJdzqGvo0yEGzFNlvzSXNtZjiKuxw0KjVEF
WpRrg/zmIkBHBY4kQ3DF4oqHy6kPeSMjC/3TKTXFJsBLFd1hBBW+HMGJoFL+SEy0ctk0HQr4fo8zUkPc82/XTSTDH2j+QYFuYNbBBOwcNUKEUNVa5
8yFY/Qr0Pj6/1LRU0/JUXtG8PnvBiKX0LRkXPxyyzxHPdpkUbbkq89MfBLkOAmG5WAokFaXDtcDTULfOO2BGkoEmR4b3Lb2UaSEFtZrVwJqzNk7l1XNMc9sdM20Wfpy0zCIRx2Hx4HLeT9eO9Ul9Kn38u99zt
J8Ni0TKqv1lD2ABy47rwzrrhEh7KIuKrV30lExj1TNz6yPTnjr27lCnS+gjSCAtrV4fPwnf9fl8p0
WtMiE9kHgFtNM3d2lKC0V6l+qKY5z+qtL3MjSrAp6l3a4E4yb3iWHzHIfyoaw8RqDYp4hfjF9b6UJVQMnQaOxo8Y4n8
8+JzS0Jz3YY3a8MaAwtDjRz8T0DGYcYYuZQkLVMqqZeuV2HpRoe+LVqmFLg9BbtxIHKjVXOSTxIOy3D3Ym
6ajK/WFZQ3eXZQ1cApN+Qu
R0Xe40ubgl7r4N5Nt7Gu
Tk5tgZfP1zRRVgW
FRKVZ1
okePHuzdu5e5c+cfMcccS5YsIV
UVDZt2sSGDRsSYNWoUzz///I/28ff3Z9gwaz4+Li6O//znPwCsX7+e6ZP
WXu0PG56Bn/3Z7mguIJcFm+dYXNi4xXIQXZlZXHqRvmZRwu
bnr3Bt1SoCI66xRSPRIaBpid0RVMgnEA/j6+pKQkEBCgLR0dEsWr4ToR4/7/7fmdv7zW19eX4m
Jrga20tJStW7cSEGPBDg9phrLbSb1Qo2eb5V9r1ttN0R/dj6mVbF3Rv/UO2uA7u2oXenIsz+cB9Du7
cIJMhzphIUlWfXZtCyqT8TEy63OxyjLkRgyCyY1xc+ngk3z7U7oip5zy2PF6l9+/axf//+8z+npaXRs
WPNPtUOGjSIuXP/948rLa1O1Vzc5/pfQWALWPsEqActQmd+AbuWwbVTaIRxV0T44y3dOXW2
mOfWfemGAGtuTXo2W77OZdrALjQP8NISMga06Qq9H4LtiyEz2e5oqmQSiM0KCgq49957iYqK
okePHuzdu5e5eZM2fW6Lkvv/wyKSkp9nJ6f0H6Vxut8Pg1iGtqEbZnrmT/gG3ov1pTmDvS8yMNKyXnj
okePHuzdu5e5c+cfMcccS5YsIV
okePHuzdu5e5eZM2fW6Lkvv/wyKSkp9OjRg6ioKF599VUAUpq8Fwrg4xBoT1X9W82LcyHKAwU1ENDwc0WPa8zk6sDwaNqAWZh9FvnAS4Q5KMeSYVubV0hTf7qVYYWw

QW+v2yxk/rEtqMCQM6sTw1i21f57owwJr7/tQ5Zr6/hx7tLuFuD5xaM2opYQY0uxTWPmb9O/VA
op70SdDF4uPjNSUI5UfbMjIy6Nq1q00ReSaXnpPSEph/PRR8B5O22t/hb/McWP87uH0hRN9Rq
6eeOVfCjS99RqCfLx883B//RvZ+Hnt82U7e23GUf03pZ668uljsXgHLx8LQ2VZhUpuISKqqxlfcbk
Yghnv5+MKwOXAqB9ZOtzeW7760FvUjh0H322v99EB/X35/czf2f1fAgs1fuyDAmtu8/78sT83iwQ
GdTPK4mHS7zVpQ//iP8EOm3dFcwCQQw/3CroYBT1gIQ3avtCeGkmJYNREaN7MSWh1rgA3s
GsqQ7m2Z85/97D6aV/0TXODMuRJ+/d4uIlo15eGBpmTJRUUEfv4X0BJYOcHjprJMAjHs0f8xCI
uDNY9C/rfuf/3P58C3O6xLioPqVyPt2eHRtGzqz9R3dlBw1v0lKGa+v4cjJ07z7PBoAvx83f76hou1j
LD+nR5Jgk0v2h3Nj5gEYtjDtxEMf83qn756snuvyspMtq6x73YbdBte78O1aOrPnNGxHM49xW9X
u7eqzrvJR3g3JZOpiZ3pc7nn3i9g1FOPUdD9Dvj0OeuqQQ9hEohhn1adYdDTcPAT6xfDHfKOwtI
74ZIwp97Q2LtTCA8P7MLK7UdZkZrltOP+lN1H8/i/1Xvo17kVj9xg7ji/qInAsBeheRisGA9nPKOtgE
kghr3ix8JVd1v9N9JcXEDu3GlYOsYq6jhmKTRp6dTDT03sQq+Iljy1arfL+4bknS5i0pLthDT15y+jY
/H1MT1tLnoBl8DtC6wp36V3QZH9bbhNAvEAx44dY/To0Vx++eXExcUxdOhQvvrqq1odY+jQofz
www8uitCFRKxF7IgB8P5UOLTZNa+jak2VZafDHQutG7WczNdHmHvnVbRu1pj730zmq+Mnnf4
aAKfPFTPhrRSy887wyI1Xe9Sd8IaLdegFw1+Fw5/DygdsX1Q3CcRmqsrw4cNJSEjg4MGDpKa
m8txzz3H8+PFaHWft2rUEB9t8T0Vd+frByLegZSfrk9V3Gc49fmkprHsC9qyEG2bCFYOde/xy2jQ
L4B/jeuHv68M9C7eRecK5XZjPnCth3JspJB86wZ9HxnJ1hxZOPb7hBaLvgEHPQMb78O8ZtlZ1
MLWwyls3A47tcu4x20bDkOerfHjDhg34+fkxceL/KtLHxMSgqkyfPp1169YhIjz11FOMGjWK7OxsR
o0aRX5+PsXFxcybN4/+/fsTHh5OSkoKBQUFDBkyhH79+pGUIERYWBirV68mMDCQgwcPMnn
yZHJycmjSpAnz588NjLSue+3rgKD4a5/wsJB8PpNMPptCHdCafqSIIj1kFWqpM8U6Dut/sesRo
eQJiwe15ORr27hnoXbWPJAb8KC61/UsLCohPGLk9n2TS4vjozI5hjPLSdvuNi1U+BkNmz5K/g0s
tYSfdx/BZ4Zgdhs9+7dxMXFXbB95cqVpKWIsXPnTtavX8/06dPJzs7m7bffZvDgwecfi42NveC5+/f
vZ/LkyezZs4fg4GBWrLDKhkyYMIG5c+eSmprK7NmmzmTRpksvfX620CIdxH0HT1vDWrbBref2Od
+60tWC+axkM/J31S+asnu/ViGzbnDfu70luwTI+PnczSQf+W6/jHcsr5BcLvyDpYC6zR8Rw61VhT
orU8Fo3/hF6Pghb/wZvj4JC99+HZEYg5f3ESMHdNm/ezJgxY/D19SU0NJTrrruO5ORkmGsa
OHUtRURG33nprpQkkIiLi/Pa4uDgOHTpEQUEBSUlJjBgx4vx+Z8+eddv7qbGyJLL0TlgxzprOGv
A4+NXyE/yRbfCvaZDzpbXGEn+/S8L9KXEdW7B6SI8efCuVuxduY8aQSB7o3+I8ZeWa+nDPMZ
5ckc654ILmjIrIlliTPAzAxweGvgBtIq2qDgtuhNFLoJX7bia1ZQQiIiNEZI+IIDq6EFa1300isk9EDojIj
HLbI0Rkm2P7uyLi757Ina9bt26kpqbWeP8BAwawceuE+NGwsLCuO+++i8eDeF/5d9LS0s
JDg4mLS3t/FdGhpPXGpyISUu4ZxXEjIFNs+GvPWHPqprN9GvuuD4exJuGu5LcmjTKf
WQbw3uS83dW/Ls2u/ZMSrW9i0P4ea1KA7knuaJ5en8+BbqbRv0YQ1U/uZ5GFcKH6s9fty6jv4
W2/44DE4Wbs11LqyawprN3AbsLGqHUTEF3gFGAJEAWNEpKxTdTHwpjHNtuK6TmJ
jI2bNnee21185vS09PJzg4mHfffZeSkhJycnLYuHEj8PHEjPXv25PDhw4SGhvLAAw8w4sEHpeev2
68ORERESxbtgywFu937tzpkvfkFH4H4B1tUm966BgOaw7F6Yn2j1Vv92h7UwXqboDGT8y7o+/sVu
kPoG9J4Ek7dBlxvsew8OQY0b8cqdV/Ps8GiO/nCGexZ+ZbcZZ+wW3zkIidWkklGdv75zoalpcrx/EI+zjjO2D
eTuW72BpZvz+LB6zqx4qFr6dVnPTqX0Dqm/ByrNU2lWONVbjURWyZwlLVDK
C6oXxP4ICqfu3YdyIwi4hkAInAnY79FgEzgXmuiteVRITExMTp06Zhe6pdKhxV
KCAmJgYRIQXXniBtm3bsmjRIv70pz/h5+dH06ZNWfff7WXcw0S+vXGq1paVqjNTixDoJqhl2DcMy57tuHtRuNeaxPd2ElHXeCS9D68g3e0RSGDhgFV2iE7WwcIVg/7+XcoxLEH9n2O1L+G00xrgGMoyLMpsBxkpJXYBeNRkA2Q6ohNPa6lRJVK0+eh3DFTYwiW9WDhnX3XgaNaoJ2B+Ks1JDoEGnLS2wZH1rYV9+3o/BPK+U/kH8+6ma5BDSLU/wJz1ivDoj11lc3Rq4ZGZSoLJLo5mjJSPrbNDHcKQBiMjlI5dhEfYp+D91v2Ki+ru/m+hyrzuLjPkJnSuEyoqFb/TQHuXyJWSvBVZ7h/GhYEtVNAdu8bFuvX8Lm07mt4ZBpJHtwZ4UAUIwl3VeG9M4xrXXjK7t6qD4ObQoA9ngZn8/78FxX/pNx+0N5sb3M/KP4oj49xz3a3cRg+LXVhnUgKOnJa4AMEBCdyXjM2LGh8KLv1GG9B+OXUqNbMC/wHtpUeIOSkqHzdOh9OgIbgv+UV7Ft6+bRnN6Hcz0bKo7umx8BbH1pGBNjCTfTn+nyOm8ZVpM5yYl/KY3sb9EmcWdntqEhA8s3mn9c5rUu6kf

SF0Zq2POhWEYNWXXZbzDRSQL6AN8ICIfOrZfJiJrARyjiynAh0AG8E9V3eM4xJPAL0XkABAC
LKxrLAEBAeTm5po/nFjJlzc3l4CAALtDMQzDCzT4nuhFRUVkZWVRWGh/ZUtPEBAQQLt27fDz
847FaMMwXM/ti+jews/Pj4iICLvDMAzD8DqevAZiGIZheDCTQAzDMIw6MQnEMAzDqJMGtYguI
jnA4Wp3rFwroH41ub2fOQfmHDT09w8N8xx0VNXWFTc2qARSHyKSUtlVCA2JOQfmHDT09w/
mHJRnprAMwzCMOjEJxDAMw6gTk0Bq7rXqd7nomXNgzkFDf/9gzsF5Zg3EMAzDqBMzAjEMw
zDqxCQQwzAMo05MAqkBEblJRPaJyAERmWF3PO4kIu1FZIOI7BWRPSIyze6Y7CIiviKyQ0TW
2B2LHUQkWESWi8iXIpIhIn3sjsndRORRx+/BbhF5R0QadOlqk0CqISK+wCvAECAKGCMiUfZG
5VbFwGOqGgX0BiY3sPdf3jSs1gIN1V+Af6tqJBBDAzsXIhIGPAzEO1po+2L1KWqwTAKpXk/ggK
p+rarngKXALTbH5Daqmq2q2x3fn8T6o9HgerCKSDvgZ8ACu2Oxg4hcAgzA0XtHVc+p6g/2RmW
LRkCgiDQCmgDf2hyPrUwCqV4YkFnu5ywa4B9QABEJB64CttkbiS3mAE8ApXYHYpMIIAd4wz
GNt0BEmtodlDup6lFgNnAEyAbyVPUje6Oyl0kgRo2ISBCwAnhEVfPtjsedRGQY8J2qptodi40aAV
cD81T1KuAU0NDWA1tgzT5EAJcBTUXkbnujspdJINU7CrQv93M7x7YGQ0T8sJLHElVdaXc8Nu
gL3Cwih7CmMBNF5B/2huR2WUCWqpaNPpdjJZSG5AbgG1XNUdUiYCVwrc0x2cokkOolA11EJ
EJE/LEWzd63OSa3ERHBmvfOUNUX7Y7HDqr6K1Vtp6rhWP//P1HVBvXJU1WPAZkicqVj00Bgr
40h2eEI0FtEmjh+LwbSwC4kqKjBt7StjqoWi8gU4EOsqy5eV9U9NoflTn2Be4BdIpLm2PZrVV1rY
0yGPaYCSxwfpL4G7rc5HrdS1W0ishzYjnV14g4aeFkTU8rEMAzDqBMzhWUYhmHUiUkghmEY
Rp2YBGIYhmHUiUkghmEYRp2YBGIYhmHUiUkghuFGIpJUi30/FZH4avY5JCKtanHM+0TkrzXd
3zB+ikkghuFGqtqg71w2Li4mgRhGJUTkGhFJF5EAEWnq6AHRvZL9VolIquPxCY5tHUVkv4i0Eh
EfEdkkIoMcjxU4/nupiGwUkTRHb4n+1cQzT0RSHK/z+woPPyEiu0TkCxHp7Ni/tYisEJFkx1dfp5w
YwyjH3lluGJVQ1WQReR94GggE/qGquyvZdayqnhCRQCBZRFao6mERmQXMA74A9lZStfVO4
ENVfcbRc6ZJNSH9xvE6vsDHItJDVdMdj+WparSI/AKravAwrN4dL6nqZhHpgFVJoWvtz4RhVM0
kEMOo2h+waqEVYjUSqszDIjLc8X17oAuQq6oLRGQEMBGIreR5ycDrjkKVq1Q1rZJ9yhvpGOE0
Ai7Fam5WlkDeKfffIxzf3wBEWSWbAGjuqKhsGE5jprAMo2ohQBDQDLigdamIJGD9oe6jqjFYtZE
CHI81warcjOMYP6KqG7EaNB0F3nSMHioIhHA48BAVe0BfFAhHq3kex+gt6rGOr7CVLWg2nds
GLVgEohhVO3vwP8BS4BZITx+CfC9qp4WkUisIr9lZjme91tgfsUnikhH4LiqzsfqcvhTpdGbY/XfyB
ORUKz2yuWNKvffLY7vP8Iqflj2epWNggyjXswUlmFUwjEiKFLVtx3rDkkikqiqn5Tb7d/ARBHJAPY
BWx3PvQ64BuirqiUicruI3K+qb5R7bgIwXUSKgAKgyhGIqu4UkR3Al1jdMT+vsEsLEUkHzgJjHNs
eBl5xbG8EbMSaTjMMpzHVeA3DMIw6MVNYhmEYRp2YBGIYhmHUiUkghmEYRp2YBGIYhmHUiUkghmEYRp38P60khJOkqACmAAAAAElFTkSuQmCC\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+uAAAABHNCSVQICAgIfAhkiAAA
AAlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9
uMy4yLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy+WH4yJAAAgAElEQVR4nO3deVzU1f7H8ddh
2HcFFAVZVBQ3ZHPNbLG6mpZmmkuall3LVtv35bbdbrua3XJLM9MsM8tKK7PScgRRVFBcEF
UQARk387vD/D+bLFchvnO8nk+Hj4eMsLMe0bn7fme+Z7zVVprhBBC2C4nowMIIYS4MFLkQgh
h46TIhRDCxkmRCyGEjZMiF0IIGydFLoQQNk6KXDgspdSNSqIvjc4hxIVSch65sHdKqwX7AK0AX
oA5IB6ZqrbcYGkwIM3E2OoAQTUkp5QusBKYCSwFXoB9QZWQuIS4mWSMXdk0plQb8orX2P8fjNwOT
gTRgHHAMuENr/bPl8VuAR4AwoBB4TWv9geWxS4FPgFnAQ0At8CxQDbwDBAJvaK1fsUzfA5gJdAR
OAdO11g80xfMWoXnOMKoQkS7agr5eQHfdTp+dhc97D9fhJ9kvZ0ptZsP8GszfTf/sB8zSYP1sNwWRHd/rDzbCxN9w/3/ZXNwdpFNo4ldX
t9q9E8t6AXwTQ/Bn0++3bPGKtk/k9xU2SUGe7DJvPM7fG1qWtgMhvm09PA/cwEGW0W3ruRLUkOQ0ZmRG3VhS0W8zq4ETTsd3vZi9k8VZvyUajQ01pF5w4j1QmNZ3uJI5EuNamuSlrNoQqxLfxsk1izxjGRLKb6b0FIS3iVmXmn
CptBDmx+QpvaDt6zGNh3j7Mtkvu1+hT+v0XOjdf8Rp9/YV1ftMb5vsZd2XXz0gvUjSPDL4qPLbfu/a4xVjiUgOb0X0Ml9LrmLMkDv2AAv6FDN08dSDOD9nrvR0zMNsjE4uytdmOQR/n9QIIUc/mwTwRXfCSF9pWgWm+u8FXr0OZIx61fKnrfMSt+PAd/n9LeBaVUJ7VIRYJqdv5SjfKsvmljwOG96AibUnD3aXptK
1zSbbSL6/5jKMl6f8drvUeTvqrHnw+7xHhw1AbTwN9JwdgHfvkT5GVd0wqcvR/mOIg33nyiK4R8a+vXIxBdtn92Bux8tlfRy1GhQ2vrebeAhzu0nNJFM/gu/c5TcPcafzu5h6+OMj+71gtzTcQTTjSvwgrmW6qkd+Q8CybJzTuEM+7n1aJRhO8T8OSyNkNdmeAlojAF8rAK6F6eUrGSumMRYzIOgQs+UgvG
cmnlR7obLkz/Wr3Ex5pEwB8svROoUz0xM3JSvJ35tMQ4R9BkMwP6mJ+yU3rQ8DCCx6Gy1Syap7JeMRkiRgQNDZHcqmnIGe/zK55K1VqbcjdKcyq0JbNwjMa82KvG3Pw8+jbpi/r0yhJBHL8SYjrNnW2TQNc2dgRLxsyoJp1q8dSXKcMGfzjixJoKmUSmyqQu7rZEgBuP81CwtEajNB5wlmd9V+Xj3dBGgTHB6wNMuVIjPv2AzC9obJz7M8Dn4dkdN8Z0cmi5kIB1Gij7Jbrz/jHaP2OozNFYrWSx2kSpeC0PKvwXyaj+e5y5Y8y6kAVRPRTpcrjjcWRP80b+dJIfHzlmZFq/P2m5VOmeG6MC2CHFexRrlQnp3Zz07f+j0kaNVONyC3d38pp4kS2UqtAH2ajlXrBTXhaG2QH+QfD8pTvr4y6G0eyt8p4A29Bp+JJhdv3+fTEgR9aoB1mH9CfqqWOYr97YRQijQIbjxO8J9CYAb9CO2GKYEM7KRJZbMpT9wOu3FiS2WRcLEtIG5XKV7W8tAPg0l4mt4HwLD4ASAkF8wpBATy3XJdyl1aurOuAZKwt8+u1RtUVgIbLhsz/A/uRZaPOF2TSYHb3sN3zOGnLfv9SzE/mgQ+1u1Dbgs8Q/hb1u+rRb+/rUTiM2hmJLd5+UcHzqjG3yXiR5mj/7E0e9kkRrmwfG9J/dGSFvtXGOw8bhj9L+XMUdjCi5BfWNeA5PqDEtRXlk/MlFdt7eSb6fFzGXP5gtf6Dx/MmBUYbAaMVbDLCd8lyYe5FEBlRh8vT+K8avB0JSQhfX7fsUhgAtwI7VlH3HI9nGMyEnb9IM+51nqm/o6hdDmn2Lm5hIkrlKi6UPWz17mkamgJCjkSy7OF/wXrG1mLKiQuMZC4S4KjM04+/2h3cSf/J/uSM12UbbRA+U6uDNQ81+Gm3H1xSjQyb2oNMA+9kcqUX6M3XrKdJvGK2qWWTbsu3vtHVLRbZc/cI6vIvXA9e4JBUGDpzddA/wt7g0jqeDFPjF+J9yCY/TwnF2xJaVgfG+WrxDnfPNDR1cKw/PPHDdMNEBwZfZ1UiDZEOl+OcFb9CJs1aAnS89uqnY+Od2iLdcx2cg51hHr+m8F19NzbHN/wEYbF5crlPNjtXXyOgr0R7TwojHD2SKbVl6aK8S5jPpY3wN9aUVk8+y23sbOJNDV8V/XSy2h7aP7LfUhH+Lz8CfB9xxIowHVvnSDvgsYeIwzjL6LRlFnWaLaNlAM+IEJGBYKnb/XkjmKnzxGjIVQmLDvzdxuYq7B29cuXz/DthZk3wyvJfHCP1DgIT7GlDnRVkRbRMbQ6MfhhpeGsc+fR74gaI75jZtPlUXc6gSTI1kr/0M1wt5R8GQMhMiBJyjOnnt7tL5eYNdLCtz0dlaRXuqLrEjnYDhRoXFN63SZSOyxuUymhNZI7ERUgoMOrn9HOyRk/wtzYBTZiuO9xmXO2zjE3h48CPrzedZc1Na93N6xEN+zlb9/Q8/KZbYeuCqu2RN8MwLw94zy0T/lDiQ3Gn6jszsstg+zD8twKrWMHkx6m1GhQVSeUuW18JM
4VFwqrtDGRZttw0WWRxgpG0BVzE0x5LmKxnaQOF4qgIEaHTb+ApLCeugbG9lh5bEsoz2lQaocVKIfXYgA7EdB2J93sN2CJtMAFshNlhGGqEHEVDT6j7kFGfVR9mo9rxkzVj7ptQDIJb/cw
YG4QEx7Gyw/CktzvuPRuXrCTSm82BLiBA6fw/K7oPsiMnTh3n8o/+t/MYwyvvIYJEK0A0u7oz9uC35yYTkrpOq/KeQHC2+F+D5xeLMK8wbFvwaJVuBUXw39iEt5K4/kXj9G+tHRlzKL/AP2Kxn5rg
zc6bZV5kWFbAaHZt8HH6ec9sYvLz2m4WrNG8IJLObTI0/sf/PrcI0nlL4Cs1Qv8QDpuLu9eSGmthqVNDnT6XeD6uw8+9KlpX0rHf6LbZ9GTH4srxKnx4tvySdyvLrXYqzNyA
sGfwg6g1n4mNI8QuaEi6thQqbVZPuHk/ms5oOmVG82QjndsOgiyrjHLmwWY6WUncTvMhehWMXK9rJPuXgKqPhcWH7bP/0vGwfZvzjJjE7xXs8svv1u58SNCNBlcPwTXrm2HQFt/cr95Sos/6DnyXfDHnrGXb1w21kiRTOYSqqNxhTP3DmKeEc2PpxzvApRoLoK4lnpm/ObfD1yjqQ4yPK/7Ku62pjvCSyJRHGdaGpt4MR4jKLwR4ubB/ft4oh2Vb2UeRrCv4vS/Q5y2sRMprbJgkFykc8Ah5HQSqp0jHNZxZe9H1rbaAtvJHiBNd9Kfyi3X8e1cB79tlwpJkAeARaG3GPeD6d6eUMHpmSDY+WFdemAqwv6DkSYrn9t2lP3tO7yHrZIBF4N4yK1k9NKnnFjr4HOaWtYn6kJNMAtvbNHqJnubRJvTy1GenHqyFS3v8GXAibMqCpLPAbEhvXN+Gu4kwytYzcpwc6N
JS2sRo5XOdCvQuYtcqnQTKPDmHsjVbNfnVYvBPowq/3o1zWaj5VNx6tx9rKmDVWbhHHPPTw6zk/+W+DbGQBBOoFtTiTo+ePRrBuY+zsA4lDu9d0Dn6NIzCutyGqbqI9pzDN7fHEnV4uXFu+Vex+gPbUwIjwW17PW75HWrk68f+cyzGg6Ydt48Z8sDrsU7RcWiV1UrD
ViW8QSzBRpJj4z9O+WwDAdS8+a8KhGwEi/VyUNzT6+uEAjmIytEi8SWSbCyh39nPUKjkamC8OAw4MJ84mVSHtYd5sPYKZWPQm+jqzyhlPEhbOwhnZoOzNC3iEkzHA0e+JEhqO4KUsGxS8l8Jsh6M+D+Jpb2pqLCRIv/V6X/40/Ixo9IEmm4tT9GwBEpmLhaTgiPeaNcn
mTkFUaA9ZWsNWjn4e/rf8Cx7uJpvI8WH6oE+Uv+j1cW+PDWxI8HbH2cwAeV6DnqESoqWbdIgmXfRcFk8tfA18Jdb2rWGbkQLvkWqVoS5ShN0b7l21YJlDnW3qFOSvNm8wU+PMXxuJnMgwqCC1SVrxbpb4E2zvmhLKyXMSRjoVSyUGdqXYWdMLbCn01/V9S2H8gu

PzQ7taxRtHOJ7kAye4bWESFdV1vDqyO1d3a2V0pD9YlpzD48t30MzTlbkTE23qtN4zFbm82/9C
ZU0dt3+YzJz12UzsG8HCSb2srsQB/DxdmH1TIndd1p7Fmw8xaUESpVW1RscSDua7XccYO3sj
3m7OfH7nRVZZ4gDXJ4Ty2R19UQpGvbeRXzILjI50waTIz6C4oobxczexZncezw/twrPXdsHFike5
Sike/EdH/j28G+szCxj57gbySiqNjiUcxOLNB7ltYRLRwT4sm9LX6tc6dGntx2d39CXE34OJ72/m8
5TDRke6INbbTAbKK6lk1Hsb2HaoiBlj4hjfJ8LoSGdtTM8w5k5I5MDxMkbN2siR4gqjIwk79/YPGT
z22Q76dwhi8eTeVnnU+mda+Xmw9PY+JIQ3Y+rH21jw636jI503KfLfOVZSyahZGzlYWM68iT0Y
EtPa6Ejn7NKOLVg4qScFJ6u44b0NHCosNzqSsFNvfb+X177dy/C4EGbflHhBqzON4OfhwoJben
JV55Y888VO5q3PNjrSeZEiP03eyUrGzN5IXkklCyf15OKoP1yIw2YkhDfnw1t7UVxew+hZGzlwv
MzoSMLOvPX9Xt76PoMRCaG8OrK7VU89/hU3ZxMzb4xnYJdgnlu5iznrsoyOdM5s85VvAvknqx
g7exNHiyuZf0tPqzm98EJ0b+PPR//sTXl1LWNnb5JpFmE2077P+F+J/+f6mL/c4MoWuJicmDE2jq
u7BfPCV+k2V+ZS5DScvjduziYOn6hg3sQe9Iiw/RI/pWuIHwsn9aKkouE5Hi+tMjqSsHHz1mfz5vd
7uT7ePkr8FBeTE9NG/3+Zf5J0yOhIZ83hi7yypo5JC7aQXVDGnAmJ9G4bYHQks+sa4secCYnk
nKhgwvubKamsMTqSsFGfpxzmuZW7GNglmFdG2E+Jn+JicuLNUbFcHBXIo5/t4NudR42OdFYc
ushr6+q566MUkg+e4I1R3bmofaDRkZpMr7YBvDsugd1HTvLPBUlU1dYZHUnYmLV78njwk1T6t
A3grdGxdlfip7g5m3h3XAJdQ/y4a3EKG/YdNzrS33LYItda8+TnaXyffoxnr+lik2ennKvLolvw+g3d2
ZRdyMOfbqe+3vKreoVtSj1UxJQPk4lu5cOsmxLs8qINp/Nyc2b+xB6EN/dk8gdJ7D120uhIf8lhi/yd
H/exZMsh7rqsPRP6Rhgdx2KGxobw8MCOrNiWyxvf7TU6jrABOSfKmbQgiSAfN+bf3NNhLj3YzM
uV+bf0xN3VxM3vbyH/pPV+vuSQRf7V9iO8unoPw2Jb88BVHYyOY3FTLmnHmJ5teHttJks2HzQ
6jrBiJZU1TJrfMBX3/sQeBNrIYh9zCfH3YO6ERI6XVXHrBw17yFgjhyvy1ENF3L90GwnhzXj5+hi
H3ClQKcVzQ7vSv0MQT3yexq92sNeEML9TnyHtyy/l3XEJtG9h3cvum0pMqD/TRsexPaeI+z7eZ
pVTkg5V5LlFFdz6QcMh4nvj7X+e76+4mJyYOTaOtoFe3PHRVg4el9Wf4rde+Cqdn/fm88KwrnZ9
IsDZ+EeXYJ64uhOrdh7lrTUZRsf5A4cp8sqaOm5bmExFdR3zHPAQ8c/4uDfsmqg1/PMD2TFR/L
+lSYeY/+t+JvWLZHTPMKPjWIVJ/SlZkRDK9DUZrEo7YnSc33CIItda88TyNHYcLubNUbGGXqn
E2kQEevH22Dgy8k5a7WGjsKyUgyd4cnkaF7UP4LFB0UbHsRpKKV4Y1pXYNv7cvzSV3UdLjI7
0Pw5R5At+3c+yrTncOyCKKzvb7vX6msrFUUE8Mbgz3+06xvQfrO+wUVhOXkklt3+YTEs/N94eE
y8XKPkddxcT741PwNvNmX9+kMSJsmqjIwEOUOSbso7z/FfpXNGpJfcOiDI6jtW65aIIrosLYdqa
DH7ck2d0HGGAmrp67li0lZKKWmaNT5QLep9BS1933h2fwLHiKqZayVGsXRd5XkkId36UQniAJ
2+M6o6Tna5EMwelFC9d142OLX24d8k22frWAb38zW6SDpzgPyNi6NTK1+g4Vi0+rBlPX9OZn/b
mW8VRrN0W+alTp8qqanl3XAK+DrKI4UJ4uDYsTa7XmimLkqmssc5zZoX5fbX9CHMbL2l4bXf7
X+VsDjf2CmN4vHUcxdptkb/67R427y/k38O7yYeb5yAi0Is3b4gl7XAJz36x0+g4wgL25Zfy8KepxI
X58/jVnYyOYzOUUrw4rOEodurH28g5YdxRrF0W+bc7j/LeT1mM6x3GsLgQo+PYnCs6t+SOS9u
xZMshlqfkGB1HNKGK6jqmfJiMm4uJd26Mx9XZLiuhyZw6iq2r09z5UQrVtfWG5LC7v7VDheU88
EkqMaF+PDWks9FxbNb9V3agZ2RznlieRmZeqdFxRBN55os0MvJKeWtULK38PIyOY5MiAr14Z
UQMqYeK+M+q3YZksKsir66t567FKQDMHBuPm7Pjrty8UM4mJ2aMicPDxcSdi7Za7R4T4vx9tj
WHpUk53HVZe/p3sN3LGlqDQd1aMbFvBHPXZxuyh7ldFfkrq3aTeqiIV0fE0Ka5p9FxbF5LX3feH
BXL3ryTPPNFmtFxhBll5p3kieVp9IxsLqflmsljV0fTLcSPBz9JtfhZX3ZT5N/vOsac9dlM6BPOwK6tj
I5jN/p3COLOS9uzNCmHFdsOGx1HmEFITR13LkrBw9XE9NFxsujHTNycTcwcG4/WcPfiFGrqLD
dfbhd/g7lFFTz4aSpdWvvymHzqbnZTr4giMbwZTyxPY39BmdFxxAV6buUu9hw7yRs3dCfYz93o
OHYILMCTl6+PYduhIl7/1nL7/dt8kdfVa6Yu2UZNbT1vj4136B0Nm4qzyYlpY+IwOSnuXmzcJ/Piw
n2z4wgfbTrIbf3bcmnHFkbHsUuDY1oxpmcY7/60j5/35lvkMW2+yGf8kMHm/YU8P6wrkYFeRsex
WyH+HrwyIoYdh4sN+2ReXJicE+U8smw73UP9eOCqjkbHsWtPD+lMh5be3L801SJXFrLpIt+cXc
j0NRkMjwtheHyo0XHs3j+6BDOhTzhz12ezVvZjsSm1dfVMXbKNeg0zxsj54k3Nw9XEjDHxnKys4
f6lTb8fi83+bRaVVzN1SQphzT15blhXo+M4jMeu7kR0sA8PLk0l72Sl0XHEWZq+JoOkAyd48bqu
hAXIGV2W0DHYh6ev6cy6jALmrM9q0seyySLXWvPosh3kl1YxY0w83m7ORkdyGO4uJmaMiaO

supYHlqZaxc5v4q9tzDrO22szGZkQytBYWelsSWN7hjGwSzCvrt7DjpziJnscmyzyxZsPsWrnUR7
+RzTdQv2MjuNwolr68NQQy4w0xIUpKq/mvo+3ER7gxbPXdjE6jsNRSvHy9d0I9HbjniUNm/g1Bb
MUuVJqoFJqj1IqUyn1qDnu80wyjp3kuZU7uTgqkEn9IpvyocRfODXSeGXVHrbnFBkdR/wJrTWP
LNtOQWkV00fH4SVHrobw93TlzVGx7D9exjNNtBHdBRe5UsoEzAQGAZ2BMUqpJtnkpLKmjrsX
p+Dl6szrN8j+4kY6NdII8nHjnsUpcr1PK/TR5oOs3nmMh/7RUY5cDda7bQB3XdaeT5NzWN0ES/j
NMSLvCWRqrbO01tXAEmCoGe73D15ZtYfdR0/y2sjutPCRhQxGOzXSOFBYLlveWpmMYyd5fu
UuLo4K5NZ+bY2OI4B7B0TxyMBoLo4KNPt9m6PIQ4BDp32d03jbbyilJiulkpRSSfn553eS/NXdgn
noHx25LFoWMliL00caX6TmGh1H8P9Hrp6uzrw+Uo5crYWzyYkpl7bD09X8U1wW+7BTaz1La52
otU4MCjq/ndYSI5pz52XtzZxMXKh7BkQRF+bPE5/tkEvEWYH/rNrdeOQaQwtfOXJ1BOYo8sNA
m9O+Dm28TTgIF5MT00fHAXDvkhRqLbhZkPittbvzeP+X/UzsG8Hl0S2NjiMsxBxFvgWIUkpFKq
VcgdHAF2a4X2FD2jT35IXrurL1YBHT1hh/MVpHIFdSyYOfpBId7MOjg6KNjiMs6IKLXGtdC9wFr
AbSgaVaa/nkywENjQ1hREIob6/NZMO+40bHcSj19Zr7l6ZSVl3LjDFxsnmcgzHLHLnW+mutdQet
dTut9YvmuE9hm/51bRciAry47+NtnCirNjqOw3jv5yzWZxbwzDVdiJKLjTscm1zZKayXl5szM8bEc
bysioeXbUdrWcLf1FIOnuD1b/cwuFsrRvdo8/c/IOyOFLkwu64hfjwyMJrvdh3jgw0HjI5j10oqa7hnS
Qotfd15aXg3lJJTDR2RFLloEpP6RXJ5dAte/CqdtMNNt1mQI9Na89hnO8gtqmT6mFj8PFyMjiQM
IkUumoRSitdGdqeZlwt3yxL+JvHR5oN8tf0ID1zVgYTw5kbHEQaSIhdNprmXK9NGx3HgeBlPfZ4
m8+VmlH6khOe+bFiCf3v/dkbHEQaTIhdNqnfbAO4ZEMXylMN8kpRjdBy7UF5dy10fbcXXw4U3R
8XKEnwhRS6a3t2XR3FR+wCeWpFG+pESo+PYNK01TyxPI6ugjGmjYgn0djM6krACUuSiyZmc
FG+NisPPw4U7F22V+fILsHjzIZanHGbqgA70bW/+XfSEbZIiFxYR5OPG9DFx7D9exmOf7ZD58v
OQdriYZ7/YSf8OQdx9uWweJ/6fFLmwmN5tA3jgqo58mZrLgl/3Gx3HphRX1DBlUTIB3q68JfPi4n
ekyIVFTbmkHVd0asELX6WzZX+h0XFsQn295oGl2zhSVMnbY+Np7uVqdCRhZaTIhUU5OSlevy
GW0GYe3LFoK3kllUZHsnozfsjk+/Q8nhzciYTwZkbHEVZIilxYnJ+HC++OT6C0spY7P9pKjexffkZr
0o/x5vd7GR4fwoS+EUbHEVZKilwYIjrYl5ev78aW/Sd4fuUuo+NYpeyCMqZ+vI2uIb68dJ3soyLOz
PwXjxPiLA2NDSHtcDGz12XTMdiHG3uFGx3JapRU1jD5gyScnRTvjkuQ/cXFX5IRuTDUo4M6c
WnHIJ5ZsVMuRtGotq6euz9KIbugjHduTCC0mafRkYSVkyIXhjI5KaaPiSM8wJM7FiVz8LhcvPnFr
9P5aW8+LwzrSp92AUbHETZAilwYztfdhTkTelCv4ZYFWygurzE6kmEWbTrA+7/sZ1K/SEb3DD
M6jrARUuTCKkQGevHuuAQOHi9n8sIkqmrrjI5kcWv35PH0ip1c1jGIx6/uZHQcYUOkyIXV6NMug
FdHxrApu5AHP9IOfb3jLONPPVTEHR9uJTrYh+lj4jDJyk1xDuSsFWFVhsaGkFtUyX9W7aa1nzu
POcDIdH9BGbfM30Kgjyvv39wDH3e50o84N1LkwurcfkIbcosqeO/nLPw9XZIyqf1eOCHvZCU3zd
uMBhbc3JMWPu5GRxI2SIpcWB2lFM9e24WSyhr+s2o3Xm4mbuoTYXQsszteWsWNszdRUFrF
olt70TbI2+hIwkZJkQurZHJquOZneXUdT6/YiaerMyMSQo2OZTZF5dWMm7uZg4XlzL+5J3Fhso
eKOH/yYaewWi4mJ2aMiaNf+0Ae/jSV5Sn2cam4ksoabpq3mX15pcy+KVHOFRcXTIpcWDV3Fx
Ozbkqgd9sA7l+aykebDhod6YIcL61i7OyNpB8p4b/j4unfIcjoSMIOSJELq+fp6sy8iT24tEMQjy/fwd
z12UZHOi9Hiiu44b0NZBwr5b3xCQzo1NLoSMJOSJELm+DuYuK98YkM6hrM8yt38drqPTZ1nnl
2QRkj/ruBvJIqFk7qxeXRUuLCfKTIhc1wdW6YMx+V2Ia312Zyz5IUKmusfwXohn3HGf7OL1TU1L
F4cm96RjY3OpKwM3LWirApziYnXr6+G5FBXrz8zW4OF1Uw+6ZEAr3djI72pxZtOsAzK3YSHuD
J3Ak9iAj0MjqSsEMyIhc2RynF7Ze04783xpN+pIQh09ezKcu6tsCtrKnjqc/TeGJ5Gv2iAll+50VS4q
LJSJELmzWoWys+vb0vHq4mxszeyIw1GdRZwbz5nqMnGTbzFxZuPMDk/m2ZO6EHvrLsXjQhK
XJh07qG+PHl3f24pntrXv9uLzfO2UhWfqkhWerrNfN/yeaat9dTUFrF+xN78PjVnWQDLNHklNaW
H8EkJibqpKQkiz+usF9aaz5JzuH5lbuoqqlnyqXtmHJpO4tdIm3boSKe+WInqYeKuKxjEK+M6E6Q
j3XO2wvbpZRK1lon/v52+bBT2AWlFDcktuHSjkG8sDKdaWsyWLHtMFOv6MCQmFY4m5rm4PN
IcQVvfLuXT5JzCPJx4/WR3RkeHyIXShYWJSNyYZfWZeTzwsp09hw7SWSgF3dc2o5rY1vj5mye
EfruoyXM+jmLL7blohTcclEkdw+lwttNxkai6ZxpRH5BRa6UGgk8C3QCemqtz6qdpciFJdTXa77dd
YzpazLYdaQEX3dnBse0Znh8CAlhzXA6x7nrYyWVrEo7ytc7jrApuxAPFxOjerRhUr9I2jSXCySLpt

dURd4JqAfeAx6UIhfWSGvNuowClqccZlXaUSpq6vDzcCEuzJ+EsGZ0auVLoI8bAV6u+Lq7UF5T
S1IVLUXlNew+epJdR0pIO1zM9pxiAKJaeDM0tjU39gqnmZerwc9OOJImmSPXWqc33vmF3I0Q
TUopRf8OQfTvEMQLw2r5Pv0YG/YdJ/nACX7ck/+3P+/r7kzn1r7cf2UHBnUNJqqljwVSC3H2LDa
hp5SaDEwGCAuTq4MLY3i5OTM0NoShsSEAFJfXkH28jMKyKgpKqzlZWYunqwkvN2d83J1pH+
RNaDMPGawIq/a3Ra6U+h4I/pM/ekJrveJsH0hrPQuYBQ1TK2edUIgm5OfpQqynv9ExhLggf1vk
WusrLBFECCHE+ZGVnUIIYeMu9KyV64AZQBBQBGzTWv/jLH4uHzhwng8bCBSc58/aC3kN5D
Vw9OcPjvkahGut/3BZKUMWBF0IpVTSn51+40jkNZDXwNGfP8hrcDqZWhFCCBsnRS6EEDbO
Fot8ltEBrIC8BvIaOPrzB3kN/sfm5siFsBSI1E7gTq31j0ZnEeKv2OKIXIg/pZQaq5RKUkqVKqWOK
KW+UUr1O9/701p3kRIXtkCKXNgFpdT9wFvAS0BLIAx4BxhqZC4hLMGmilwpNVAptUcplamUet
ToPJaklGqjlFqrlNqllNqplLrX6ExGUUqZlFIpSqmVjV/7Ac/RMA3ymda6TGtdo7X+Umv9kFLKTSn
1lllqt/HXW0opt8afDVRKrVRKFSmlCpVS65RSTo1/tl8pdUXj759VSi1VSn2glDrZ+HeQeFqm1kqp
ZUqpfKVUtlLqniZ8/v5KqU+VUruVUulKqT5N9VjWSil1X+PfQZpSarFSyt3oTEaymSJXSpmAmcA
goDMwRinV2dhUFlULPKC17gz0Bu50sOd/unuB9NO+7gO4A8vP8P1P0PCaxQLdgZ7Ak41/9gC
QQ8OitpbA48CZPji6FlgC+ANfAG8DNBb/l0AqEAIMAKYqpf52cdx5mgas0lpH0/B80v/m++2KUio
EuAdI1Fp3BUzAaGNTGctmipyGN1+m1jpLa11NwxvKYQ6btdZHtNZbG39/koY3b4ixqSxPKRUK
DAbmnHZzAFCgta49w4/dCDyntc7TWucD/wLGN/5ZDdCKhhVzNVrrdfrMZwCs11p/rbWuAxbSU
KIAPYAgrfVzWutqrXUWMJsmKJfGo4/+wFyAxscrMvfj2ABnwEMp5Qx4ArkG5zGULRV5CHDotK
9zcMAiA1BKRQBxwCZjkxjiLeBhGi5ocspxILDxTf1nWvPbLSEONN4G8CqQCXyrlMr6mym7o6f9
vhxwb3zMcKB14/RMkVKqiIaRfcuzfVLnIBLIB95vnF6ao5TyaoLHsVpa68PAa8BB4AhQrLX+1thU
xrKIIheAUsobWAZM1VqXGJ3HkpRSQ4A8rXXy7/5oA1AFDDvDj+bSULanhDXehtb6pNb6Aa11
WxqmTu5XSg04x2iHgGyttf9pv3y01lef4/2cDWcgHviv1joOKAMc7fOiZjQcjUfS8B+yl1JqnLGpjGV
LRX4YaHPa16GNtzkMpZQLDSW+SGv9mdF5DHARcK1Saj8NU2uXK6U+1FoXA08DM5VSw5
RSnkopF6XUIKXUK8Bi4EmlVJBSKrDxez+Ehv8clFLtVcOVI4qBOn472j8bm4GTSqlHlFIejR/Gdl
VK9TDLs/6tHCBHa33qaOxTGordkVxBw3+c+VrrGuAzoK/BmQxlS0W+BYhSSkUqpVxpmH/8wu
BMFtNYNHOBdK31G0bnMYLW+jGtdajWOoKGv/8ftNbjGv/sdeB+Gj7EzKdhlHwX8DnwApAEbA
d2AFsbbwOlAr4HSmkY2b+jtV57jrnqgCE0fJiaTcOOfHMAv/N9rn/xWEeBQ0qpjo03DQB2mftxrNx
BoHfjf9iKhtfAoT7w/T2bWtmplLqahjlSEzBPa/2iwZEspnFhyzoaiujUiPFxrfXXxqUyjlLqUhou+MbigI
gAABwhSURBVD3E6CyWppSKpeE/ClcgC7hZa33C2FSWpZT6FzCKhrO5UoBbtdZVxqYyjk0Vu
RBCiD+ypakVIYQQf0KKXAghwRBC2LgzLaBoUoGBgToiIsKIhxZCCJuVnJxc8GfX7DRLkS
ul5tFw+lVe494HfykiIoKkpCRzPLQQQjgMpdSfXrTeXFMr84GBZrovIYQQ58AsI3Kt9c+N+380qfQ
jJeSfrMLf0wU/Dxeae7ni4+7S1A8rhM04UVZNfmkVZVW1VFTXARDo40aQtxt+Hi44OSmDE4qm
YLE5cqXUZGAyQFhY2Hndx4cbD7Bo08Hf3BbW3JOYUD9i2/hzWXQL2gV5X3BWIWyB1podh4
v5ftcxth8uJv1ICcdKzrwmxs3Zie6h/iRENKNHRDP6tgvE3cVkwcSiqZhtQVDjiHzl2cyRJyYm6vOZI
88tqiC3qIKi8hqKK2o4WlLJjpxitucUkVtcCUCX1r5c070118WF0NLXofeaF3YqM+8kizYdZHXaUX
KLKzE5KaJaeNOplS+dWvnQys8DbzdnPF1N1GsoKK2ioLSKg4XlbD1wgp25JdTWa3zcnLm6W
yuuiw+hZ0RzGa3bAKVUstY68Q+321KR/5Xcogq+STvKl6m5bDtUhKvJiesTQritfzsiAh1ql09hh7T
WbMouZPbPWazZnYersxP9o4IY2DWYAdEtaObletb3VVFdx5b9hazYlss3aUcor66jQ0tv7hkQxd
VdW0mhWzG7L/LT7S8oY876LJYm5VBbV8+w2BAeHRRNCxmhCxuUfqSE577cxYas4zT3cuW
mPuGM7x1OgLfbBd93eXUt3+w4yn9/2kdmXiIRLby5/8oODOwaTMN+VMKaNGmRK6UWA5cC
gcAx4Bmt9dwzfX9TF/kpeSWVzFmfzfxf9+NqcuK+KzswoU84ziZZByWs34myat74bi+LNh3A18O
FqQOiGN0zrEnmtevqNV/vOML0NRlk5JVySYcgXhjWlTbNPc3+WOL8NfmI/FxYqshP2V9QxjNf7
OSnvflEB/vw1uhYooN9Lfb4QpyrtbvzeOjTVE6U1zCuVxj3XdkBf8+znz45X3X1mg827Oe11Xuo0
5p7B3Rgcv+2mGS6xSo4dJFDwxzj6p3HeGpFGsUVNTw5uBPje4fL4aOwKhXVdbz0dToLNx4gO
tiHN0fF0qmV5QcdR4oreGbFTr7ddYzebZszbXScnDxgBRy+yE8pKK3iwU9S+XFPPld0aslrI2Ms

MtIR4u9k5ZcyeWEymXml3NovkocGdsTN2bjTA7XWfJqcw9MrduLpauLNUbH07/CH1eHCgs5U
5A43WRzo7ca8CT14akhnftqbx3Xv/EpWfqnRsYSD+2lvPkNn/kJhWTUfTurFk0M6G1riAEopRia2
4Yu7LiLA25UJ729m5tpM5BoG1sfhihzAyUkxqV8kH/2zN8UVNVz3zq/8mllgdCzhgLTWzFmXxc3
vbybE34MVd15Ev6hAo2P9RlRLH1bc2Y9rYlrz6uo9PLJsOzV153pZU9GUHLLIT+kR0ZwVd15E
S183bpq3maVbDhkdSTiQ+nrNv77cxQtfpXNV52CWTelrtWeJeLiamDY6lnsub8/SpBwmvr+Z4oo
ao2OJRg5d5ABtmnuybEpf+rQL4OFl25m7PtvoSMIB1NTV88Anqcz/dT+T+kXyzo3xeLkZsqv0W
VNKcf9VHXltZHc2ZxcyetZGjpc67GUyrYrDFzmAj7sLcyYkMqhrMM+v3MX0NRkyDyiaTGVNHV
M+TGZ5ymEevKoDTw7uZFOrKUckhDJnQg+y8ksZNWsjeSWVRkdyeFLkjdycTcwYE8f18aG88d
1eXv5mt5S5MLvKmjr++UESa3bn8fywrtx1eZRNngJ7SYcg5t/ck9yiCm54bwOHiyqMjuTQpMhP4
2xy4tURMYzvHc57P2fxxnd7jY4k7Eh1bT13LNrKuowC/nN9w78zW9anXQALJ/XieGk1o2dt4Gixj
MyNIkX+O05Oin9d24VRiW2Y8UMmM9dmGh1J2IGaunru+mgrP+zO46XrunFDYhujI5lFQngzFt
7ai8LSasbN3SRz5gaRIv8TTk6Kl4Z3Y2hsw+lW8+QDUHEB6us1DyxN5dtdx/jXtV0Y2+v89uO3V
rFt/Jk7sQeHCsu5ad5mSirlbBZLkyI/A5OT4vWR3RnYJZjnVu5ixbbDRkcSNkhrzfNf7eKL1FweGR
jNhL4RRkdqEr3bBvDu+AT2HjvJpPlbqKypMzqSQ5Ei/wvOJiemjYmld9vmPPhJKr/IoiFxjmb9nMX
7v+znlosiuf2StkbHaVKXdWzBW6PiSDpwgqlLtlFXLycLWIoU+d9wczbx3vhE2gZ6c/vCZNKPlBgd
SdiI5Sk5/Pub3QyJacWTgzvZ5Nkp52pwTCueGtyZVTuP8uJX6UbHcRhS5GfBz8OF92/ugZebMx
Pf30yunGol/sbGrOM89Ml2+rQN4PUbutvUeeIX6pZ+kdxyUSTzfsmWBXYWIkV+llr7ezD/lh6UVd
Vx64IkyqtrjY4krNT+gjJu/zCZ8ABP3h2fYPjmV0Z4YnAnBnYJ5oWvdvHtzqNGx7F7UuTnIDrYl+lj
Ykk/WsIDS1OplzlA8TvFFTVMWrAFgLkTeuDn4WJwImOYnBRvjY4lJtSfqR9vY/dRmZJsSlLk5+jy
6JY8PqgT36Qd5a3vZcGQ+H+1jeeKHyws591xCQ5/0W93FxOzxifg7ebMrQuS5BzzJiRFfh5uvTi
SkQmhTP8hky9Tc42OI6zEy9/sZl1GAS8M60rvtgFGx7EKLX3dmXVTInknq5iyaCvVtbL9bVOQIj8
PSilevK4bieHNePjT7XLYKFix7TBz1mczoU84o3rY14KfCxXbxp9Xro9hc3Yhz6/cZXQcuyRFfp5c
nZ1458Z4fNyduW1hsuzN7MB25ZbwyLLt9IxozpNDOhsdxyoNiwthcv+2LNx4gGXJOUbHsTtS5B
egha87/x0XT25RBfd9vE0+/HRAReXV3PZhEv4ersy8MR4Xk7ylzuThf3SkT9sAHl++g525xUbHs
Svyr+4CJYQ35+khnflhdx7T1mQYHUdYUH29ZurH2zhWXMV/x8UT5ONmdCSr5mxyYsbYOJp5
unL7h8kUlVcbHcluSJGbwbje4QyPD2H6Dxn8vDff6DjCQt75MZMf9+Tz1DWdiQtrZnQcmxDo7c
Y74+I5WlwpR7FmJEVuBkopXhzWjQ4tfJj68TaOFMvKT3v3a2YBb3y3l6GxrRlnZ7sZNrX4sGY8f
U0X1u7J592f9xkdxy5IkZuJh6uJmTfGU1VTx90fpchVxu3YsZJK7lmSQtsgb166rptD7KFibuN6hT
EkphWvrd7DpqzjRsexeVLkZtS+hTcvDe9G0oETvLp6j9FxRBOoravn7sUplFXV8V8buGCytVJK8
e/h3QgP8OLuxSkUyGKhCyJFbmZDY0MY1zuMWT9n8cPuY0bHEWY2fU0Gm7MLefG6rkS19D
E6jk3zcXdh5th4iitqZL78AkmRN4EnB3emcytfHliaKvPlduSXzAJmrM1kZEIow+NDjY5jFzq39uVf1
3ZhXUYB//1J5svPlxR5E3B3MfH22Diqauu5d/E2amW+3Obln6zi3iXbaBfkzb+GdjE6jl0Z1aMN13
RvzRvf7SVpf6HRcWySFHkTaRvkzQvDurJ5fyHT5fxym1Zfr7l/6TZOVtbw9tg4PF1lXtyclFK8dF1X
Qvw9uGdxipxffh6kyJvQ8PhQRiSEMmNtJr/uk8vE2apZ67JYl1HAM9d0ITrY1+g4dsnH3YW3x8a
RX1rFw59uR2uZLz8XUuRN7LmhXYgM9OK+j7dRWCYjDVuTcvAEr63ew+BurRjTs43RcexaTKg
/jwyM5ttdx1i48YDRcWyKFHkT83R1ZvroOE6U1fDwp6ky0rAhJZU13LMkhZa+7rw0XM4Xt4RbL
ork0o5BvPBVuuwqeg6kyC2ga4gfjw6K5vv0PD7YICMNW6C15snlaeQWVTJ9TKzDXunH0pycF
K+N7l6vuwv3LE6horrO6Eg2QYrcQm6+KILLo1vw4tfppB+RkYa1W7b1MF+k5nLvgCgSwpsbHc
ehBHq78cYN3dl7rJQXvpL9y8+GWYpcKTVQKbVHKZWplHrUHPdpb5RSvDoiBj8PF+6WkYZVy
y4o4+kVafSKbM6dl7U3Oo5D6t8hiNv6t2XRpoOsSpOLN/+dCy5ypZQJmAkMAjoDY5RSsrv+nwh
oHGlk5slIw1pV19Zz75IUXExOvDkqFpOTzIsb5YGrOhIT6sejn22XhXV/wxwj8p5AptY6S2tdDSw
Bhprhfu3SxVEy0rBmr3+3h+05xfzn+hha+3sYHcehuTo7MW10HNW19Uxdso06WcJ/RuYo8hDg0
Glf5zTe9htKqclKqSSlVFJ+vmPv2f3AVR3pFiIjDWuzPqOA937KYmyvMAZ2DTY6jgAiA73417Vd2
JRdyLuyhP+MLPZhp9Z6ltY6UWudGBQUZKmHtUquzk5MH9Mw0rjvYxlpWIPjpVXct3Qb7Vt489

RgmRm0JiMSQv+3hH/rwRNGx7FK5ijyw8DpKyVCG28Tf+HUSGNjlow0jKa15qFPt1NcUcOMM
XF4uJqMjiROo5TihWFdCfZ1594lKZRUyoXOf88cRb4FiFJKRSqlXIHRwBdmuF+7JyMN67Dg1/3
8sDuPxwZF06mVLMG3Rn4eLkwfE0tuUSVPLk+ThXW/c8FFrrWuBe4CVgPpwFKt9c4LvV9HoJ
Tixeu60srPnXsWy0jDCOlHSnjpm91cHt2CiX0jjI4j/kJCeHOmDojii9Rclm2Vg/7TmWWOXGv9tda
6g9a6ndb6RXPcp6PwdXdh2ug4jhTLSMPSyqtruXtxCn4eLrw6IkaW4NuAOy5rT6/I5jy9Io2s/FKj4
1gNWdlpBRLCm/1vpPFJco7RcRzGc1/uYl9+KW/eEEuAt5vRccRZMDkp3hwVi4vJiXuWpFBdK3
v9gxS51bjjsvb0btucZ1bsJDNPRhpN7cvUXJZsOcTtl7SjX1Sg0XHEOWjt78Erl2JIO1zCK6t2Gx3
HKkiRWwmTk2La6IYzJu76aCuVNbKEv6kcKizn8c92EBfmz/1XdjA6jjgP/+gSzE19wpmzPluujYs
UuVVp6evOayNj2H30JP/+Ot3oOHappq6euxengILpo+NwMclbwFY9fnUnOjVeG/docaXRcQwl/4
qtzOXRLZnUL5IFGw6wKu2I0XHszqur97DtUBEvD4+hTXNPo+OIC/Cba+MuSXHohXVS5FbokY
HRdA/146FPt3OosNzoOHbjh93HmPVzFuN6hzE4ppXRcYQZtAvy5rmhXdmUXcg0B742rhS5FX
J1duLtsfEA3PXRVvlk3gxyiyq4f2kqnVv58qQswbcrlxJCuT4+lBk/ZLAuwzH3cZlit1Jtmnvy6ojupO
YU8+9vZL78QtTU1XPP4hRqauuZeWM87i6yBN/ePD+sC+2DvJm6ZBvHShxvvlyK3IoN7BrMzR
dF8P4v+/lmh8yXn69XVu0m6cAJXhrejchAL6PjiCbg6erMOzfGU15dx92LU6itc6yjWClyK/fYoE50
b+PPQ59ul5Vs52FV2hFmr8tmfO9whsb+YXdlYUeiWvrw4nVd2ZxdyGvf7jU6jkVJkVs5V2cn3rkx
HheTYsqHWymvrjU6ks3Iyi/lwU+2072NP08O6WR0HGEBw+NDGdMzjHd/2udQF26RIrcBIf4eTB
8Tx968kzz22Q7Zj+UsVFTXcceirbiYFO/cGI+bs8yLO4pnr+1M91A/Hvwk1WGOYqXIbcTFUUE8c
GUHVmzLZcGv+42OY9W01jyybDt7jp3krdFxhMgl2xyKm7OJd8Yl4GJS3P5hMmVV9n8UK0VuQ
+64tD1XdGrBC1+ls2HfcaPjWK1ZP2fxRWouD17VkUs6OPbVqBxViL8HM8bEk5lXysPLttv9Uaw
UuQ1xclK8MSqW8ABP7vxoqywW+hM/7c3nP6t2c3W3YO64tJ3RcYSB+kUF8vDAaL7afoSZaz
ONjtOkpMhtjK+7C7NvSqSmrp7JC5Plw8/T7C8o4+6PttKhpQ+vjugu+4sLbuvflmGxrXnt2718u9N
+P/yUIrdBbYO8mTEmjj1HS3jok+3UO/AeE6cUV9QwacEWnJwUs8Yn4uXmbHQkYQWUUrx8fQ
wxoX7c9/E29hw9aXSkJiFFbqMu7diCRwdF89WOI7z+3R6j4xiqpq6eOxYlc7CwnHfHJRAWIJthif/
n7mJi1vhEPN2cmbRgC/knq4yOZHZS5Dbsnxe3ZUzPMGau3cfSLYeMjmMIrTVPfZ7GL5nH+ffw
GHq3DTA6krBCwX7uzLkpkYLSKm79IImKavva71+K3IYppXhuaBcujgrk8eU7WJ9RYHQki3vv5y
yWbDnEXZe1Z0RCqNFxhBXr3saf6aPj2J5TxD12tu2tFLmNczE1rPxs38KbKR8msyu3xOhIFvPZ
1hxe/mY3g2NayZV+xFm5qkswzwzpzHe7jvH8yl12c1qiFLkd8HF3Yd7EHni7O3PTvM3sLygzOlK
T+2H3MR76dDt92wXwxg3dcXKSM1TE2Zl4USS39otk/q/7eefHfUbHMQspcjvR2t+DhZN6Uldfz7i
5m+z60lfJBwq5Y9FWOrXy4b3xCbL8Xpyzx6/uxHVxIby6eo9drJSWIrcj7Vv4sOCWnpwoq+ameZ
s4UVZtdCSzSztczC3zk2jl58H8m3vi4+5idCRhg5ycFK+OiOHKzi155oudLEvOMTrSBZEitzMxof7
MnpDI/uPl3DhnE4V2VOZph4u5cc4mvN2c+eCWngR6uxkdSdgwZ5MTM8bEcVH7AB5etp2vttvu
nv9S5Haob7tAZt+UyL78UsbO3sjxUts/b3ZnbjHj5jaU+JLJveXCycIsTp1jHh/mz92Lt7I8xTZH5lLk
duqSDkHMndCD7IIyxszeaNOLIHbkNIzEPV1MLP6nlLgwLy83Z+bf3JNekQHcvzTVJtdkSJHbsX5
Rgbw/sQcHC8u54b0NHDhue2ez/Lw3n1GzNuDl6sySyX1k1aZoEl5uzsyb2IN+7QN5eNl25v+Sb
XSkcyJFbuf6tg9k0a29OFFezfB3fiX1UJHRkc7a5ymHuWX+FsIDvFh+R18pcdGkPFxNzL4pkSs7
t+TZL3fxwspdNrOPkRS5A0gIb86yKX3xcDUxetZG1qQfMzrSX9JaM3NtJlM/3kZiRDM+vq03LXz
djY4lHIC7i4l3xyUwsW8Ec9Znc8eirTaxnF+K3EG0C/Lmszv60r6FN7d+kMS07zOscrRRWlXLlA+
38urqPVzbvTXzb+6Jr5xiKCzI5KR49touPDWkM6t3HWXUrA1Wv/e/FLkDaeHjzse39WZYbAhvfr
+XWxZsoajcek5P3JdfyrCZv/Bd+jGeHNyJaaNjcXeRxT7CGJP6RfLeuASy88sYPH2dVe9nLkXuY
DxdnXnjhu68MKwrv2YeZ/D09fySaexmW/X1mgW/7mfI9PUUllWzcFJPbr24rVwYQhjuqi7BrLynH
2EBnkxemMzzK3dRWWN9Uy3KiE1jEhMTdVJSksUfV/zWtkNF3PfxtoZTFHu24bGrO1l8GuNQY
TmPLNvOr/uO079DEP+5vhut/ORiycK6VNXW8eJX6Xyw4QCRgV68dF03+rSz/JbJSqlkrXXiH26
XIndslTV1vPndXmavy6KFjzuPDOrItd1DMDXxJlRlVbXM+jmL2euyUMCTQzozukcbGYULq7Y+o
4DHl+9oOKU3MZSH/hFNkl/lVhhLkYu/lHqoiMeX72BnbgnRwT48eFVHBnRqYfZiraqtY1nyYd78fi

/5J6sY3K0Vjw6KlkU+wmZUVNcxbU0Gs9dl4WJSTOgTweT+bQmwwJYRTVLkSqmRwLNAJ6C
n1vqs2lmK3DrV12u+2nGEN77bS3ZBGdHBPoztFcbQ2BD8PC5syuVIcQUfbTrI4s0HKSitJjG8G
Y8P7kR8WDMzpRfCsrILypixJoPPtx3G3cXEiIRQRia0oWuIb5MdWTZVkXcC6oH3gAelyO1DTV
09y7ceZsGG/ezMLcHdxYkrOwdzcftA+rYPILTZ34+etdZk5pXy45581u7JY1N2IfVaMyC6BTf1ieDi
qECZRhF2ITOvlJlrM/lqxxGqa+uJDvZhSEwrerUNICbUz6zbLDfp1IpS6kekyO3SjpxiPtp8kO92Ha
OgcfOtEH8PwgM8adPMk1b+7piUol5DXX09h4sqyS4oJaugjKLyGgA6tPTmik4tGdMzTKZQhN0q
Lq/hy+25fJKc878V1K7OTnRp7Utrfw+Cfd1p6evGoK6tzvt9YHiRK6UmA5MBwsLCEg4cOHDBjys
sR2vN3mOl/JJZQMqhIg4VlpNzouJ/5X5KsK87kYFeRAZ50aW1L5d2bEGIv5yFIhxLYVk1SfsL2Z
xdSFpuMcdKqjhaXElFTR0fTupFv6jA87rf8y5ypdT3QPCf/NETWusVjd/zIzIid0g1dfUAmJRCKWS
6RIgz0FpTWlWLq7PTeU+3nKnInc/iwa84r0cUDsHFJGvKhDgbSqkmu6KVvAuFEMLGXVCRK6
WuU0rlAH2Ar5RSq80TSwghxNkyZEGQUiofON9POwMBYzcHMZ68BvIaOPrzB8d8DcK11kG/v
9GQIr8QSqmkP5vsdyTyGshr4OjPH+Q1OJ3MkQshhI2TIhdCCBtni0U+y+gAVkBeA3kNHP35g7
wG/2Nzc+RCCCF+yxZH5EIIIU4jRS6EEDbOpopcKTVQKbVHKZWplHrU6DyWpJRqo5Raq5Tap
ZTaqZS61+hMRlFKmZRSKUqplUZnMYJSyl8p9alSardSKl0p1cfoTJamlLqv8X2QppRarJRyNzqT
kWymyJVSJmAmMAjoDIxRSnU2NpVF1QIPaK07A72BOx3s+Z/uXiDd6BAGmgas0lpHA91xsNd
CKRUC3AMkaq27AiZgtLGpjGUzRQ70BDK11lla62pgCTDU4EwWo7U+orXe2vj7kzS8eUOMTW
V5SqlQYDAwx+gsRlBK+QH9gbkAWutqrXWRsakM4Qx4KKWcAU8g1+A8hrKlIg8BDp32dQ4O
WGQASqkIIA7YZGwSQ7wFPEzDlakcUSSQD7zfOL00RynlZXQoS9JaHwZeAw4CR4BirfW3xq
Yyli0VuQCUUt7AMmCq1rrE6DyWpJQaAuRprZONzmIgZyAe+K/WOg4oAxzt86JmNByNRwKtA
S+l1DhjUxnLlor8MNDmtK9DG29zGEopFxpKfJHW+jOj8xjgIuBapdR+GqbWLldKfWhsJIvLAXK0
1qeOxj6lodgdyRVAttY6X2tdA3wG9DU4k6Fsqci3AFFKqUillCsNH258YXAmi1ENl96ZC6Rrrd8w
Oo8RtNaPaa1DtdYRNPz9/6C1dqiRmNb6KHBIKdWx8aYBwC4DIxnhINBbKeXZ+L4YgIN94Pt7f
3uFIGuhta5VSt0FrKbhU+p5WuudBseypIuA8cAOpdS2xtse11p/bWAmYYy7gUWNA5os4GaD81i
U1nqTUupTYCsNZ3Ol4ODL9WWJvhBC2DhbmloRQgjxJ6TIhRDCxkmRCyGEjZMiF0IIGydFLo
QQNk6KXAghbJwUuRBC2Lj/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"
     }
    ]
   }
  ]
```

```
}
```