```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "dzNng6vCL9eP"
   },
   "source": [
    "## CS231n Python Tutorial With Jupyter Notebook"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "0vJLt3JRL9eR"
   },
   "source": [
    "This tutorial was originally written by [Justin Johnson](https://web.eecs.umich.edu/~justincj/) for cs231n and adapted as a Jupyter notebook for cs228 by [Volodymyr Kuleshov](http://web.stanford.edu/~kuleshov/) and [Isaac Caswell](https://symsys.stanford.edu/viewing/symsysaffiliate/21335).\n",
    "\n",
    "This current version has been adapted as a Jupyter notebook with Python3 support by Kevin Zakka for the Spring 2020 edition of [cs231n](http://cs231n.stanford.edu/)."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## What is a Jupyter Notebook?"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "A Jupyter notebook is made up of a number of cells. Each cell can contain Python code. There are two main types of cells: `Code` cells and `Markdown` cells. This particular cell is a `Markdown` cell. You can execute a particular cell by double clicking on it (the highlight color will switch from blue to green) and pressing `Shift-Enter`. When you do so, if the cell is a `Code` cell, the code in the cell will run, and the output of the cell will be displayed beneath the cell, and if the cell is a `Markdown` cell, the markdown text will get rendered beneath the cell.\n",
    "\n",
    "Go ahead and try executing this cell."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The cell below is a `Code` cell. Go ahead and click it, then execute it."
   ]
  },
```

```
  {
   "cell_type": "code",
   "execution_count": 101,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "1\n"
     ]
    }
   ],
   "source": [
    "x = 1\n",
    "print(x)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Global variables are shared between cells. Try executing the cell below:"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 102,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "2\n"
     ]
    }
   ],
   "source": [
    "y = 2 * x\n",
    "print(y)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Keyboard Shortcuts\n",
    "\n",
    "There are a few keyboard shortcuts you should be aware of to make your notebook experience more pleasant. To escape editing of a cell, press `esc`. Escaping a `Markdown` cell won't render it, so make sure to execute it if you wish to render the markdown. Notice how the highlight color switches back to blue when you have escaped a cell.\n",
    "\n",
    "You can navigate between cells by pressing your arrow keys. Executing a cell automatically shifts the cell cursor down 1 cell if one exists, or creates a new cell below the current one if none exist.\n",
```

```
    "\n",
    "* To place a cell below the current one, press `b`.\n",
    "* To place a cell above the current one, press `a`.\n",
    "* To delete a cell, press `dd`.\n",
    "* To convert a cell to `Markdown` press `m`. Note you have to be in `esc` mode.\n",
    "* To convert it back to `Code` press `y`. Note you have to be in `esc` mode.\n",
    "\n",
    "Get familiar with these keyboard shortcuts, they really help!"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "You can restart a notebook and clear all cells by clicking `Kernel -> Restart & Clear Output`. If you don't want to clear cell outputs, just hit `Kernel -> Restart`.\n",
    "\n",
    "By convention, Jupyter notebooks are expected to be run from top to bottom. Failing to execute some cells or executing cells out of order can result in errors. After restarting the notebook, try running the `y = 2 * x` cell 2 cells above and observe what happens."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "After you have modified a Jupyter notebook for one of the assignments by modifying or executing some of its cells, remember to save your changes! You can save with the `Command/Control + s` shortcut or by clicking `File -> Save and Checkpoint`."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "This has only been a brief introduction to Jupyter notebooks, but it should be enough to get you up and running on the assignments for this course."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "qVrTo-LhL9eS"
   },
   "source": [
    "## Python Tutorial"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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": {
"colab_type": "text",
"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": {
"colab_type": "text",
"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.**\n",
"\n",
"You should have activated your `cs231n` virtual environment created in the [Setup Instructions](https://cs231n.github.io/setup-instructions/) before calling `jupyter notebook`. If that is\n",
"the case, the cell below should print out a major version of 3.7."
]
},
{
"cell_type": "code",
"execution_count": 2,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 34
},
"colab_type": "code",

"id": "1L4Am0QATgOc",
"outputId": "bb5ee3ac-8683-44ab-e599-a2077510f327"
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"Python 3.7.6\r\n"
]
}
],
"source": [
"!python --version"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"id": "JAFKYgrpL9eY"
},
"source": [
"## Basics of Python"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"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",
"execution_count": 3,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 34
},
"colab_type": "code",
"id": "cYb0pjh1L9eb",
"outputId": "9a8e37de-1dc1-4092-faee-06ad4ff2d73a"
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"[1, 1, 2, 3, 6, 8, 10]\n"

```
     ]
    }
   ],
   "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]))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "NwS_hu4xL9eo"
   },
   "source": [
    "### Basic data types"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "DL5sMSZ9L9eq"
   },
   "source": [
    "#### Numbers"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "MGS0XEWoL9er"
   },
   "source": [
    "Integers and floats work as you would expect from other languages:"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "KheDr_zDL9es",
    "outputId": "1db9f4d3-2e0d-4008-f78a-161ed52c4359"
```

```
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "3 <class 'int'>\n"
     ]
    }
   ],
   "source": [
    "x = 3\n",
    "print(x, type(x))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 86
    },
    "colab_type": "code",
    "id": "sk_8DFcuL9ey",
    "outputId": "dd60a271-3457-465d-e16a-41acf12a56ab"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "4\n",
      "2\n",
      "6\n",
      "9\n"
     ]
    }
   ],
   "source": [
    "print(x + 1)    # Addition\n",
    "print(x - 1)    # Subtraction\n",
    "print(x * 2)    # Multiplication\n",
    "print(x ** 2)   # Exponentiation"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 6,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "U4Jl8K0tL9e4",
    "outputId": "07e3db14-3781-42b7-8ba6-042b3f9f72ba"
```

```
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "4\n",
       "8\n"
      ]
     }
    ],
    "source": [
     "x += 1\n",
     "print(x)\n",
     "x *= 2\n",
     "print(x)"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 7,
    "metadata": {
     "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 52
     },
     "colab_type": "code",
     "id": "w-nZ0Sg_L9e9",
     "outputId": "3aa579f8-9540-46ef-935e-be887781ecb4"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "<class 'float'>\n",
       "2.5 3.5 5.0 6.25\n"
      ]
     }
    ],
    "source": [
     "y = 2.5\n",
     "print(type(y))\n",
     "print(y, y + 1, y * 2, y ** 2)"
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "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": {
    "colab_type": "text",
    "id": "EqRS7qhBL9fC"
   },
   "source": [
    "#### Booleans"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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",
   "execution_count": 8,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 34
    },
    "colab_type": "code",
    "id": "RvoImwgGL9fE",
    "outputId": "1517077b-edca-463f-857b-6a8c386cd387"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "<class 'bool'>\n"
     ]
    }
   ],
   "source": [
    "t, f = True, False\n",
    "print(type(t))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "YQgmQfOgL9fI"
```

```
  },
  "source": [
   "Now we let's look at the operations:"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 9,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 86
   },
   "colab_type": "code",
   "id": "6zYm7WzCL9fK",
   "outputId": "f3cebe76-5af4-473a-8127-88a1fd60560f"
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "False\n",
     "True\n",
     "False\n",
     "True\n"
    ]
   }
  ],
  "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;"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "id": "UQnQWFEyL9fP"
  },
  "source": [
   "#### Strings"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 10,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 34
   },
   "colab_type": "code",
   "id": "AijEDtPFL9fP",
   "outputId": "2a6b0cd7-58f1-43cf-e6b7-bf940d532549"
```

```
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "hello 5\n"
    ]
   }
  ],
  "source": [
   "hello = 'hello'    # String literals can use single quotes\n",
   "world = \"world\"    # or double quotes; it does not matter\n",
   "print(hello, len(hello))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 11,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 34
   },
   "colab_type": "code",
   "id": "saDeaA7hL9fT",
   "outputId": "2837d0ab-9ae5-4053-d087-bfa0af81c344"
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "hello world\n"
    ]
   }
  ],
  "source": [
   "hw = hello + ' ' + world  # String concatenation\n",
   "print(hw)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 12,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 34
   },
   "colab_type": "code",
   "id": "Nji1_UjYL9fY",
   "outputId": "0149b0ca-425a-4a34-8e24-8dff7080922e"
  },
  "outputs": [
   {
    "name": "stdout",
```

```
    "output_type": "stream",
    "text": [
     "hello world 12\n"
    ]
   }
  ],
  "source": [
   "hw12 = '{} {} {}'.format(hello, world, 12)   # string formatting\n",
   "print(hw12)"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "id": "bUpl35bIL9fc"
  },
  "source": [
   "String objects have a bunch of useful methods; for example:"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 13,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 121
   },
   "colab_type": "code",
   "id": "VOxGatlsL9fd",
   "outputId": "ab009df3-8643-4d3e-f85f-a813b70db9cb"
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "Hello\n",
     "HELLO\n",
     "  hello\n",
     " hello \n",
     "he(ell)(ell)o\n",
     "world\n"
    ]
   }
  ],
  "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"
  ]
 },
```

{
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "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": {
   "colab_type": "text",
   "id": "p-6hClFjL9fk"
  },
  "source": [
   "### Containers"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "id": "FD9H18eQL9fk"
  },
  "source": [
   "Python includes several built-in container types: lists, dictionaries, sets, and tuples."
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "id": "UsIWOe0LL9fn"
  },
  "source": [
   "#### Lists"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "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",
  "execution_count": 14,
  "metadata": {
   "colab": {

```
    "base_uri": "https://localhost:8080/",
    "height": 52
   },
   "colab_type": "code",
   "id": "hk3A8pPcL9fp",
   "outputId": "b545939a-580c-4356-db95-7ad3670b46e4"
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "[3, 1, 2] 2\n",
     "2\n"
    ]
   }
  ],
  "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\""
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 15,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 34
   },
   "colab_type": "code",
   "id": "YCjCy_0_L9ft",
   "outputId": "417c54ff-170b-4372-9099-0f756f8e48af"
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "[3, 1, 'foo']\n"
    ]
   }
  ],
  "source": [
   "xs[2] = 'foo'     # Lists can contain elements of different types\n",
   "print(xs)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 16,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 34
   },
```

"colab_type": "code",
"id": "vJ0x5cF-L9fx",
"outputId": "a97731a3-70e1-4553-d9e0-2aea227cac80"
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"[3, 1, 'foo', 'bar']\n"
]
}
],
"source": [
"xs.append('bar') # Add a new element to the end of the list\n",
"print(xs)   "
]
},
{
"cell_type": "code",
"execution_count": 17,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 34
},
"colab_type": "code",
"id": "cxVCNRTNL9f1",
"outputId": "508fbe59-20aa-48b5-a1b2-f90363e7a104"
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"bar [3, 1, 'foo']\n"
]
}
],
"source": [
"x = xs.pop()   # Remove and return the last element of the list\n",
"print(x, xs)"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"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": {
    "colab_type": "text",
    "id": "ovahhxd_L9f5"
   },
   "source": [
    "#### Slicing"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "YeSYKhv9L9f6"
   },
   "source": [
    "In addition to accessing list elements one at a time, Python provides concise syntax to access sublists; this is known as slicing:"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 18,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 139
    },
    "colab_type": "code",
    "id": "ninq666bL9f6",
    "outputId": "c3c2ed92-7358-4fdb-bbc0-e90f82e7e941"
   },
   "outputs": [
    {
     "name": "stdout",
     "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"
     ]
    }
   ],
   "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]\""
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "id": "UONpMhF4L9f_"
  },
  "source": [
   "#### Loops"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "id": "_DYz1j6QL9f_"
  },
  "source": [
   "You can loop over the elements of a list like this:"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 19,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 69
   },
   "colab_type": "code",
   "id": "4cCOysfWL9gA",
   "outputId": "560e46c7-279c-409a-838c-64bea8d321c4"
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "cat\n",
     "dog\n",
     "monkey\n"
    ]
   }
  ],
  "source": [
   "animals = ['cat', 'dog', 'monkey']\n",
   "for animal in animals:\n",
   "    print(animal)"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
```

```
    "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",
   "execution_count": 20,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 69
    },
    "colab_type": "code",
    "id": "JjGnDluWL9gF",
    "outputId": "81421905-17ea-4c5a-bcc0-176de19fd9bd"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "#1: cat\n",
      "#2: dog\n",
      "#3: monkey\n"
     ]
    }
   ],
   "source": [
    "animals = ['cat', 'dog', 'monkey']\n",
    "for idx, animal in enumerate(animals):\n",
    "    print('#{}: {}'.format(idx + 1, animal))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "arrLCcMyL9gK"
   },
   "source": [
    "#### List comprehensions"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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",
 "execution_count": 21,
 "metadata": {
  "colab": {
   "base_uri": "https://localhost:8080/",
   "height": 34
  },
  "colab_type": "code",
  "id": "IVNEwoMXL9gL",
  "outputId": "d571445b-055d-45f0-f800-24fd76ceec5a"
 },
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "[0, 1, 4, 9, 16]\n"
   ]
  }
 ],
 "source": [
  "nums = [0, 1, 2, 3, 4]\n",
  "squares = []\n",
  "for x in nums:\n",
  "     squares.append(x ** 2)\n",
  "print(squares)"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {
  "colab_type": "text",
  "id": "7DmKVUFaL9gQ"
 },
 "source": [
  "You can make this code simpler using a list comprehension:"
 ]
},
{
 "cell_type": "code",
 "execution_count": 22,
 "metadata": {
  "colab": {
   "base_uri": "https://localhost:8080/",
   "height": 34
  },
  "colab_type": "code",
  "id": "kZxsUfV6L9gR",
  "outputId": "4254a7d4-58ba-4f70-a963-20c46b485b72"
 },
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "[0, 1, 4, 9, 16]\n"
```

```
    ]
   }
  ],
  "source": [
   "nums = [0, 1, 2, 3, 4]\n",
   "squares = [x ** 2 for x in nums]\n",
   "print(squares)"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "id": "-D8ARK7tL9gV"
  },
  "source": [
   "List comprehensions can also contain conditions:"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 23,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 34
   },
   "colab_type": "code",
   "id": "yUtgOyyYL9gV",
   "outputId": "1ae7ab58-8119-44dc-8e57-fda09197d026"
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "[0, 4, 16]\n"
    ]
   }
  ],
  "source": [
   "nums = [0, 1, 2, 3, 4]\n",
   "even_squares = [x ** 2 for x in nums if x % 2 == 0]\n",
   "print(even_squares)"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "id": "H8xsUEFpL9gZ"
  },
  "source": [
   "#### Dictionaries"
  ]
 },
 {
```

```
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "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",
    "execution_count": 24,
    "metadata": {
     "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 52
     },
     "colab_type": "code",
     "id": "XBYI1MrYL9gb",
     "outputId": "8e24c1da-0fc0-4b4c-a3e6-6f758a53b7da"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "cute\n",
       "True\n"
      ]
     }
    ],
    "source": [
     "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\""
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 25,
    "metadata": {
     "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 34
     },
     "colab_type": "code",
     "id": "pS7e-G-HL9gf",
     "outputId": "feb4bf18-c0a3-42a2-eaf5-3fc390f36dcf"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "wet\n"
      ]
```

```
    }
   ],
   "source": [
    "d['fish'] = 'wet'     # Set an entry in a dictionary\n",
    "print(d['fish'])      # Prints \"wet\""
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 26,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 165
    },
    "colab_type": "code",
    "id": "tFY065ItL9gi",
    "outputId": "7e42a5f0-1856-4608-a927-0930ab37a66c"
   },
   "outputs": [
    {
     "ename": "KeyError",
     "evalue": "'monkey'",
     "output_type": "error",
     "traceback": [
      "\u001b[0;31m---------------------------------------------------------------------------
----------\u001b[0m",
      "\u001b[0;31mKeyError\u001b[0m                                  Traceback
(most recent call last)",
      "\u001b[0;32m<ipython-input-26-78fc9745d9cf>\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\u001b[0;34m\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\u001b[0m",
      "\u001b[0;31mKeyError\u001b[0m: 'monkey'"
     ]
    }
   ],
   "source": [
    "print(d['monkey'])  # KeyError: 'monkey' not a key of d"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 27,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "8TjbEWqML9gl",
    "outputId": "ef14d05e-401d-4d23-ed1a-0fe6b4c77d6f"
   },
   "outputs": [
    {
```

```
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "N/A\n",
       "wet\n"
      ]
     }
    ],
    "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\""
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 28,
    "metadata": {
     "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 34
     },
     "colab_type": "code",
     "id": "0EItdNBJL9go",
     "outputId": "652a950f-b0c2-4623-98bd-0191b300cd57"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "N/A\n"
      ]
     }
    ],
    "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\""
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "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": {
     "colab_type": "text",
     "id": "IxwEqHlGL9gr"
    },
    "source": [
```

"It is easy to iterate over the keys in a dictionary:"
]
},
{
"cell_type": "code",
"execution_count": 29,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 69
},
"colab_type": "code",
"id": "rYfz7ZKNL9gs",
"outputId": "155bdb17-3179-4292-c832-8166e955e942"
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"A person has 2 legs\n",
"A cat has 4 legs\n",
"A spider has 8 legs\n"
]
}
],
"source": [
"d = {'person': 2, 'cat': 4, 'spider': 8}\n",
"for animal, legs in d.items():\n",
"   print('A {} has {} legs'.format(animal, legs))"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"id": "17sxiOpzL9gz"
},
"source": [
"Dictionary comprehensions: These are similar to list comprehensions, but allow you to easily construct dictionaries. For example:"
]
},
{
"cell_type": "code",
"execution_count": 30,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 34
},
"colab_type": "code",
"id": "8PB07imLL9gz",
"outputId": "e9ddf886-39ed-4f35-dd80-64a19d2eec9b"
},
"outputs": [
{

```
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "{0: 0, 2: 4, 4: 16}\n"
      ]
     }
    ],
    "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)"
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "id": "V9MHfUdvL9g2"
    },
    "source": [
     "#### Sets"
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "id": "Rpm4UtNpL9g2"
    },
    "source": [
     "A set is an unordered collection of distinct elements. As a simple example, consider
the following:"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 31,
    "metadata": {
     "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 52
     },
     "colab_type": "code",
     "id": "MmyaniLsL9g2",
     "outputId": "8f152d48-0a07-432a-cf98-8de4fd57ddbb"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "True\n",
       "False\n"
      ]
     }
    ],
    "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"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 32,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "ElJEyK86L9g6",
    "outputId": "b9d7dab9-5a98-41cd-efbc-786d0c4377f7"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "True\n",
      "3\n"
     ]
    }
   ],
   "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;"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 33,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "5uGmrxdPL9g9",
    "outputId": "e644d24c-26c6-4b43-ab15-8aa81fe884d4"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "3\n",
      "2\n"
     ]
    }
   ],
   "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))   "
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "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",
    "execution_count": 35,
    "metadata": {
     "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 69
     },
     "colab_type": "code",
     "id": "K47KYNGyL9hA",
     "outputId": "4477f897-4355-4816-b39b-b93ffbac4bf0"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "#1: dog\n",
       "#2: fish\n",
       "#3: cat\n"
      ]
     }
    ],
    "source": [
     "animals = {'cat', 'dog', 'fish'}\n",
     "for idx, animal in enumerate(animals):\n",
     "   print('#{}: {}'.format(idx + 1, animal))"
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "id": "puq4S8buL9hC"
    },
    "source": [
     "Set comprehensions: Like lists and dictionaries, we can easily construct sets using set comprehensions:"
    ]

},
{
 "cell_type": "code",
 "execution_count": 36,
 "metadata": {
  "colab": {
   "base_uri": "https://localhost:8080/",
   "height": 34
  },
  "colab_type": "code",
  "id": "iw7k90k3L9hC",
  "outputId": "72d6b824-6d31-47b2-f929-4cf434590ee5"
 },
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "{0, 1, 2, 3, 4, 5}\n"
   ]
  }
 ],
 "source": [
  "from math import sqrt\n",
  "print({int(sqrt(x)) for x in range(30)})"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {
  "colab_type": "text",
  "id": "qPsHSKB1L9hF"
 },
 "source": [
  "#### Tuples"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {
  "colab_type": "text",
  "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 trivial example:"
 ]
},
{
 "cell_type": "code",
 "execution_count": 37,
 "metadata": {
  "colab": {
   "base_uri": "https://localhost:8080/",
   "height": 69
  },

```
    "colab_type": "code",
    "id": "9wHUyTKxL9hH",
    "outputId": "cdc5f620-04fe-4b0b-df7a-55b061d23d88"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "<class 'tuple'>\n",
      "5\n",
      "1\n"
     ]
    }
   ],
   "source": [
    "d = {(x, x + 1): x for x in range(10)}   # Create a dictionary with tuple keys\n",
    "t = (5, 6)          # Create a tuple\n",
    "print(type(t))\n",
    "print(d[t])         \n",
    "print(d[(1, 2)])"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 38,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 165
    },
    "colab_type": "code",
    "id": "HoO8zYKzL9hJ",
    "outputId": "28862bfc-0298-40d7-f8c4-168e109d2d93"
   },
   "outputs": [
    {
     "ename": "TypeError",
     "evalue": "'tuple' object does not support item assignment",
     "output_type": "error",
     "traceback": [
      "\u001b[0;31m---------------------------------------------------------------------------\u001b[0m",
      "\u001b[0;31mTypeError\u001b[0m                                 Traceback (most recent call last)",
      "\u001b[0;32m<ipython-input-38-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"
     ]
    }
   ],
   "source": [
```

"t[0] = 1"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"id": "AXA4jrEOL9hM"
},
"source": [
"### Functions"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"id": "WaRms-QfL9hN"
},
"source": [
"Python functions are defined using the `def` keyword. For example:"
]
},
{
"cell_type": "code",
"execution_count": 39,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 69
},
"colab_type": "code",
"id": "kiMDUr58L9hN",
"outputId": "9f53bf9a-7b2a-4c51-9def-398e4677cd6c"
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"negative\n",
"zero\n",
"positive\n"
]
}
],
"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))"

```
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "id": "U-QJFt8TL9hR"
  },
  "source": [
   "We will often define functions to take optional keyword arguments, like this:"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 40,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 52
   },
   "colab_type": "code",
   "id": "PfsZ3DazL9hR",
   "outputId": "6e6af832-67d8-4d8c-949b-335927684ae3"
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "Hello, Bob!\n",
     "HELLO, FRED\n"
    ]
   }
  ],
  "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)"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "id": "ObA9PRtQL9hT"
  },
  "source": [
   "### Classes"
  ]
 },
 {
  "cell_type": "markdown",
```

```
   "metadata": {
    "colab_type": "text",
    "id": "hAzL_lTkL9hU"
   },
   "source": [
    "The syntax for defining classes in Python is straightforward:"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 41,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "RWdbaGigL9hU",
    "outputId": "4f6615c5-75a7-4ce4-8ea1-1e7f5e4e9fc3"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Hello, Fred!\n",
      "HELLO, FRED\n"
     ]
    }
   ],
   "source": [
    "class Greeter:\n",
    "\n",
    "    # Constructor\n",
    "    def __init__(self, name):\n",
    "        self.name = name  # Create an instance variable\n",
    "\n",
    "    # Instance method\n",
    "    def greet(self, loud=False):\n",
    "        if loud:\n",
    "            print('HELLO, {}'.format(self.name.upper()))\n",
    "        else:\n",
    "            print('Hello, {}!'.format(self.name))\n",
    "\n",
    "g = Greeter('Fred')  # Construct an instance of the Greeter class\n",
    "g.greet()            # Call an instance method; prints \"Hello, Fred\"\n",
    "g.greet(loud=True)   # Call an instance method; prints \"HELLO, FRED!\""
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "3cfrOV4dL9hW"
   },
   "source": [
    "## Numpy"
```

]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"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": {
"colab_type": "text",
"id": "lZMyAdqhL9hY"
},
"source": [
"To use Numpy, we first need to import the `numpy` package:"
]
},
{
"cell_type": "code",
"execution_count": 42,
"metadata": {
"colab": {},
"colab_type": "code",
"id": "58QdX8BLL9hZ"
},
"outputs": [],
"source": [
"import numpy as np"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"id": "DDx6v1EdL9hb"
},
"source": [
"### Arrays"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"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": {
    "colab_type": "text",
    "id": "_eMTRnZRL9hc"
   },
   "source": [
    "We can initialize numpy arrays from nested Python lists, and access elements using square brackets:"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 43,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "-l3JrGxCL9hc",
    "outputId": "8d9dad18-c734-4a8a-ca8c-44060a40fb79"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "<class 'numpy.ndarray'> (3,) 1 2 3\n",
      "[5 2 3]\n"
     ]
    }
   ],
   "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)                  "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 44,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "ma6mk-kdL9hh",
    "outputId": "0b54ff2f-e7f1-4b30-c653-9bf81cb8fbb0"
   },
   "outputs": [
```

```
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "[[1 2 3]\n",
    " [4 5 6]]\n"
   ]
  }
 ],
 "source": [
  "b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array\n",
  "print(b)"
 ]
},
{
 "cell_type": "code",
 "execution_count": 45,
 "metadata": {
  "colab": {
   "base_uri": "https://localhost:8080/",
   "height": 52
  },
  "colab_type": "code",
  "id": "ymfSHAwtL9hj",
  "outputId": "5bd292d8-c751-43b9-d480-f357dde52342"
 },
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "(2, 3)\n",
    "1 2 4\n"
   ]
  }
 ],
 "source": [
  "print(b.shape)\n",
  "print(b[0, 0], b[0, 1], b[1, 0])"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {
  "colab_type": "text",
  "id": "F2qwdyvuL9hn"
 },
 "source": [
  "Numpy also provides many functions to create arrays:"
 ]
},
{
 "cell_type": "code",
 "execution_count": 46,
 "metadata": {
  "colab": {
   "base_uri": "https://localhost:8080/",
```

```
      "height": 52
     },
     "colab_type": "code",
     "id": "mVTN_EBqL9hn",
     "outputId": "d267c65f-ba90-4043-cedb-f468ab1bcc5d"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "[[0. 0.]\n",
       " [0. 0.]]\n"
      ]
     }
    ],
    "source": [
     "a = np.zeros((2,2))   # Create an array of all zeros\n",
     "print(a)"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 47,
    "metadata": {
     "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 34
     },
     "colab_type": "code",
     "id": "skiKlNmlL9h5",
     "outputId": "7d1ec1b5-a1fe-4f44-cbe3-cdeacad425f1"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "[[1. 1.]]\n"
      ]
     }
    ],
    "source": [
     "b = np.ones((1,2))    # Create an array of all ones\n",
     "print(b)"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 48,
    "metadata": {
     "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 52
     },
     "colab_type": "code",
     "id": "HtFsr03bL9h7",
```

"outputId": "2688b157-2fad-4fc6-f20b-8633207f0326"
},
"outputs": [
 {
  "name": "stdout",
  "output_type": "stream",
  "text": [
   "[[7 7]\n",
   " [7 7]]\n"
  ]
 }
],
"source": [
 "c = np.full((2,2), 7) # Create a constant array\n",
 "print(c)"
]
},
{
"cell_type": "code",
"execution_count": 49,
"metadata": {
 "colab": {
  "base_uri": "https://localhost:8080/",
  "height": 52
 },
 "colab_type": "code",
 "id": "-QcALHvkL9h9",
 "outputId": "5035d6fe-cb7e-4222-c972-55fe23c9d4c0"
},
"outputs": [
 {
  "name": "stdout",
  "output_type": "stream",
  "text": [
   "[[1. 0.]\n",
   " [0. 1.]]\n"
  ]
 }
],
"source": [
 "d = np.eye(2)   # Create a 2x2 identity matrix\n",
 "print(d)"
]
},
{
"cell_type": "code",
"execution_count": 50,
"metadata": {
 "colab": {
  "base_uri": "https://localhost:8080/",
  "height": 52
 },
 "colab_type": "code",
 "id": "RCpaYg9qL9iA",
 "outputId": "25f0b387-39cf-42f3-8701-de860cc75e2e"
},
"outputs": [

{
"name": "stdout",
"output_type": "stream",
"text": [
"[[0.5293933   0.83232089]\n",
" [0.54040558 0.42955453]]\n"
]
}
],
"source": [
"e = np.random.random((2,2)) # Create an array filled with random values\n",
"print(e)"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"id": "jI5qcSDfL9iC"
},
"source": [
"### Array indexing"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"id": "M-E4MUeVL9iC"
},
"source": [
"Numpy offers several ways to index into arrays."
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"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",
"execution_count": 51,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 52
},
"colab_type": "code",
"id": "wLWA0udwL9iD",
"outputId": "99f08618-c513-4982-8982-b146fc72dab3"
},

```
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[[2 3]\n",
      " [6 7]]\n"
     ]
    }
   ],
   "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)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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",
   "execution_count": 52,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "1kmtaFHuL9iG",
    "outputId": "ee3ab60c-4064-4a9e-b04c-453d3955f1d1"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "2\n",
      "77\n"
```

```
      ]
     }
    ],
    "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]) "
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "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",
    "execution_count": 53,
    "metadata": {
     "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 69
     },
     "colab_type": "code",
     "id": "G6lfbPuxL9iJ",
     "outputId": "a225fe9d-2a29-4e14-a243-2b7d583bd4bc"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "[[ 1  2  3  4]\n",
       " [ 5  6  7  8]\n",
       " [ 9 10 11 12]]\n"
      ]
     }
    ],
    "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)"
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "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",
   "execution_count": 54,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 69
    },
    "colab_type": "code",
    "id": "EOiEMsmNL9iL",
    "outputId": "ab2ebe48-9002-45a8-9462-fd490b467f40"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[5 6 7 8] (4,)\n",
      "[[5 6 7 8]] (1, 4)\n",
      "[[5 6 7 8]] (1, 4)\n"
     ]
    }
   ],
   "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)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 55,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 104
    },
    "colab_type": "code",
    "id": "JXu73pfDL9iN",
    "outputId": "6c589b85-e9b0-4c13-a39d-4cd9fb2f41ac"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[ 2   6 10] (3,)\n",
      "\n",
```

```
      "[[ 2]\n",
      " [ 6]\n",
      " [10]] (3, 1)\n"
     ]
    }
   ],
   "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)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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",
   "execution_count": 56,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "TBnWonIDL9iP",
    "outputId": "c29fa2cd-234e-4765-c70a-6889acc63573"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[1 4 5]\n",
      "[1 4 5]\n"
     ]
    }
   ],
   "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]]))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 57,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "n7vuati-L9iR",
    "outputId": "c3e9ba14-f66e-4202-999e-2e1aed5bd631"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[2 2]\n",
      "[2 2]\n"
     ]
    }
   ],
   "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]]))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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",
   "execution_count": 58,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 86
    },
    "colab_type": "code",
    "id": "ehqsV7TXL9iU",
    "outputId": "de509c40-4ee4-4b7c-e75d-1a936a3350e7"
```

```
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[[ 1   2   3]\n",
      " [ 4   5   6]\n",
      " [ 7   8   9]\n",
      " [10 11 12]]\n"
     ]
    }
   ],
   "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)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 59,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 34
    },
    "colab_type": "code",
    "id": "pAPOoqy5L9iV",
    "outputId": "f812e29b-9218-4767-d3a8-e9854e754e68"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[ 1   6   7 11]\n"
     ]
    }
   ],
   "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]\""
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 60,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 86
    },
    "colab_type": "code",
```

"id": "6v1PdI1DL9ib",
"outputId": "89f50f82-de1b-4417-e55c-edbc0ee07584"
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"[[11 2 3]\n",
" [4 5 16]\n",
" [17 8 9]\n",
" [10 21 12]]\n"
]
}
],
"source": [
"# Mutate one element from each row of a using the indices in b\n",
"a[np.arange(4), b] += 10\n",
"print(a)"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"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",
"execution_count": 61,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 69
},
"colab_type": "code",
"id": "32PusjtKL9id",
"outputId": "8782e8ec-b78d-44d7-8141-23e39750b854"
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"[[False False]\n",
" [True True]\n",
" [True True]]\n"
]
}
],
"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)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 62,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "cb2IRMXaL9if",
    "outputId": "5983f208-3738-472d-d6ab-11fe85b36c95"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[3 4 5 6]\n",
      "[3 4 5 6]\n"
     ]
    }
   ],
   "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])"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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": {
  "colab_type": "text",
  "id": "jTctwqdQL9ih"
 },
 "source": [
  "### Datatypes"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {
  "colab_type": "text",
  "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",
 "execution_count": 63,
 "metadata": {
  "colab": {
   "base_uri": "https://localhost:8080/",
   "height": 34
  },
  "colab_type": "code",
  "id": "4za4O0m5L9ih",
  "outputId": "2ea4fb80-a4df-43f9-c162-5665895c13ae"
 },
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "int64 float64 int64\n"
   ]
  }
 ],
 "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)"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {
  "colab_type": "text",
  "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": {
    "colab_type": "text",
    "id": "TuB-fdhIL9ik"
   },
   "source": [
    "### Array math"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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",
   "execution_count": 64,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 86
    },
    "colab_type": "code",
    "id": "gHKvBrSKL9il",
    "outputId": "a8a924b1-9d60-4b68-8fd3-e4657ae3f08b"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[[ 6.  8.]\n",
      " [10. 12.]]\n",
      "[[ 6.  8.]\n",
      " [10. 12.]]\n"
     ]
    }
   ],
   "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))"
   ]
```

},
{
 "cell_type": "code",
 "execution_count": 65,
 "metadata": {
  "colab": {
   "base_uri": "https://localhost:8080/",
   "height": 86
  },
  "colab_type": "code",
  "id": "1fZtIAMxL9in",
  "outputId": "122f1380-6144-4d6c-9d31-f62d839889a2"
 },
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "[[-4. -4.]\n",
    " [-4. -4.]]\n",
    "[[-4. -4.]\n",
    " [-4. -4.]]\n"
   ]
  }
 ],
 "source": [
  "# Elementwise difference; both produce the array\n",
  "print(x - y)\n",
  "print(np.subtract(x, y))"
 ]
},
{
 "cell_type": "code",
 "execution_count": 66,
 "metadata": {
  "colab": {
   "base_uri": "https://localhost:8080/",
   "height": 86
  },
  "colab_type": "code",
  "id": "nil4AScML9io",
  "outputId": "038c8bb2-122b-4e59-c0a8-a091014fe68e"
 },
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "[[5. 12.]\n",
    " [21. 32.]]\n",
    "[[5. 12.]\n",
    " [21. 32.]]\n"
   ]
  }
 ],
 "source": [
  "# Elementwise product; both produce the array\n",

```
    "print(x * y)\n",
    "print(np.multiply(x, y))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 67,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 86
    },
    "colab_type": "code",
    "id": "0JoA4lH6L9ip",
    "outputId": "12351a74-7871-4bc2-97ce-a508bf4810da"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[[0.2        0.33333333]\n",
      " [0.42857143 0.5       ]]\n",
      "[[0.2        0.33333333]\n",
      " [0.42857143 0.5       ]]\n"
     ]
    }
   ],
   "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))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 68,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "g0iZuA6bL9ir",
    "outputId": "29927dda-4167-4aa8-fbda-9008b09e4356"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[[1.         1.41421356]\n",
      " [1.73205081 2.        ]]\n"
     ]
    }
```

```
   ],
   "source": [
    "# Elementwise square root; produces the array\n",
    "# [[ 1.          1.41421356]\n",
    "#  [ 1.73205081  2.        ]]\n",
    "print(np.sqrt(x))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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",
   "execution_count": 69,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "I3FnmoSeL9iu",
    "outputId": "46f4575a-2e5e-4347-a34e-0cc5bd280110"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "219\n",
      "219\n"
     ]
    }
   ],
   "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))"
   ]
  },
  {
   "cell_type": "markdown",
```

```
  "metadata": {
   "colab_type": "text",
   "id": "vmxPbrHASVeA"
  },
  "source": [
   "You can also use the `@` operator which is equivalent to numpy's `dot` operator."
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 70,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 34
   },
   "colab_type": "code",
   "id": "vyrWA-mXSdtt",
   "outputId": "a9aae545-2c93-4649-b220-b097655955f6"
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "219\n"
    ]
   }
  ],
  "source": [
   "print(v @ w)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 71,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 69
   },
   "colab_type": "code",
   "id": "zvUODeTxL9iw",
   "outputId": "4093fc76-094f-4453-a421-a212b5226968"
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "[29 67]\n",
     "[29 67]\n",
     "[29 67]\n"
    ]
   }
  ],
  "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)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 72,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 121
    },
    "colab_type": "code",
    "id": "3V_3NzNEL9iy",
    "outputId": "af2a89f9-af5d-47a6-9ad2-06a84b521b94"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[[19 22]\n",
      " [43 50]]\n",
      "[[19 22]\n",
      " [43 50]]\n",
      "[[19 22]\n",
      " [43 50]]\n"
     ]
    }
   ],
   "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)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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",
   "execution_count": 73,
   "metadata": {
    "colab": {
```

"base_uri": "https://localhost:8080/",
"height": 69
},
"colab_type": "code",
"id": "DZUdZvPrL9i0",
"outputId": "99cad470-d692-4b25-91c9-a57aa25f4c6e"
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"10\n",
"[4 6]\n",
"[3 7]\n"
]
}
],
"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]\""
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"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",
"execution_count": 74,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 104
},
"colab_type": "code",
"id": "63Yl1f3oL9i3",
"outputId": "c75ac7ba-4351-42f8-a09c-a4e0d966ab50"
},
"outputs": [
{
"name": "stdout",

```
    "output_type": "stream",
    "text": [
     "[[1 2]\n",
     " [3 4]]\n",
     "transpose\n",
     " [[1 3]\n",
     " [2 4]]\n"
    ]
   }
  ],
  "source": [
   "print(x)\n",
   "print(\"transpose\\n\", x.T)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 75,
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 104
   },
   "colab_type": "code",
   "id": "mkk03eNIL9i4",
   "outputId": "499eec5a-55b7-473a-d4aa-9d023d63885a"
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "[[1 2 3]]\n",
     "transpose\n",
     " [[1]\n",
     " [2]\n",
     " [3]]\n"
    ]
   }
  ],
  "source": [
   "v = np.array([[1,2,3]])\n",
   "print(v )\n",
   "print(\"transpose\\n\", v.T)"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {
   "colab_type": "text",
   "id": "REfLrUTcL9i7"
  },
  "source": [
   "### Broadcasting"
  ]
 },
 {
```

"cell_type": "markdown",
"metadata": {
 "colab_type": "text",
 "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",
"execution_count": 76,
"metadata": {
 "colab": {
  "base_uri": "https://localhost:8080/",
  "height": 86
 },
 "colab_type": "code",
 "id": "WEEvkV1ZL9i7",
 "outputId": "3896d03c-3ece-4aa8-f675-aef3a220574d"
},
"outputs": [
 {
  "name": "stdout",
  "output_type": "stream",
  "text": [
   "[[2 2 4]\n",
   " [5 5 7]\n",
   " [8 8 10]\n",
   " [11 11 13]]\n"
  ]
 }
],
"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)"
]
},
{
"cell_type": "markdown",
"metadata": {
 "colab_type": "text",

"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",
"execution_count": 77,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 86
},
"colab_type": "code",
"id": "vS7UwAQQL9i-",
"outputId": "8621e502-c25d-4a18-c973-886dbfd1df36"
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"[[1 0 1]\n",
" [1 0 1]\n",
" [1 0 1]\n",
" [1 0 1]]\n"
]
}
],
"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]]\""
]
},
{
"cell_type": "code",
"execution_count": 78,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 86
},
"colab_type": "code",
"id": "N0hJphSIL9jA",
"outputId": "def6a757-170c-43bf-8728-732dfb133273"
},
"outputs": [
{
"name": "stdout",

```
      "output_type": "stream",
      "text": [
       "[[ 2  2  4]\n",
       " [ 5  5  7]\n",
       " [ 8  8 10]\n",
       " [11 11 13]]\n"
      ]
     }
    ],
    "source": [
     "y = x + vv  # Add x and vv elementwise\n",
     "print(y)"
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "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",
    "execution_count": 79,
    "metadata": {
     "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 86
     },
     "colab_type": "code",
     "id": "vnYFb-gYL9jC",
     "outputId": "df3bea8a-ad72-4a83-90bb-306b55c6fb93"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "[[ 2  2  4]\n",
       " [ 5  5  7]\n",
       " [ 8  8 10]\n",
       " [11 11 13]]\n"
      ]
     }
    ],
    "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)"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"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",
"execution_count": 80,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 69
},
"colab_type": "code",
"id": "EmQnwoM9L9jH",
"outputId": "f59e181e-e2d4-416c-d094-c4d003ce8509"
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"[[4 5]\n",
" [8 10]\n",

```
      " [12 15]]\n"
     ]
    }
   ],
   "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)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 81,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "PgotmpcnL9jK",
    "outputId": "567763d3-073a-4e3c-9ebe-6c7d2b6d3446"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[[2 4 6]\n",
      " [5 7 9]]\n"
     ]
    }
   ],
   "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)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 82,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "T5hKS1QaL9jK",
    "outputId": "5f14ac5c-7a21-4216-e91d-cfce5720a804"
```

```
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[[ 5  6  7]\n",
      " [ 9 10 11]]\n"
     ]
    }
   ],
   "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)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 83,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 52
    },
    "colab_type": "code",
    "id": "JDUrZUl6L9jN",
    "outputId": "53e99a89-c599-406d-9fe3-7aa35ae5fb90"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[[ 5  6  7]\n",
      " [ 9 10 11]]\n"
     ]
    }
   ],
   "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)))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 84,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
```

```
     "height": 52
    },
    "colab_type": "code",
    "id": "VzrEo4KGL9jP",
    "outputId": "53c9d4cc-32d5-46b0-d090-53c7db57fb32"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "[[ 2  4  6]\n",
      " [ 8 10 12]]\n"
     ]
    }
   ],
   "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)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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": {
    "colab_type": "text",
    "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": {
    "colab_type": "text",
    "id": "tEINf4bEL9jR"
   },
   "source": [
    "## Matplotlib"
   ]
```

```
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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",
   "execution_count": 85,
   "metadata": {
    "colab": {},
    "colab_type": "code",
    "id": "cmh_7c6KL9jR"
   },
   "outputs": [],
   "source": [
    "import matplotlib.pyplot as plt"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "jOsaA5hGL9jS"
   },
   "source": [
    "By running this special iPython command, we will be displaying plots inline:"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 86,
   "metadata": {
    "colab": {},
    "colab_type": "code",
    "id": "ijpsmwGnL9jT"
   },
   "outputs": [],
   "source": [
    "%matplotlib inline"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "U5Z_oMoLL9jV"
   },
   "source": [
    "### Plotting"
   ]
```

},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"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",
"execution_count": 87,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 282
},
"colab_type": "code",
"id": "pua52BGeL9jW",
"outputId": "9ac3ee0f-7ff7-463b-b901-c33d21a2b10c"
},
"outputs": [
{
"data": {
"text/plain": [
"[<matplotlib.lines.Line2D at 0x121741990>]"
]
},
"execution_count": 87,
"metadata": {},
"output_type": "execute_result"
},
{
"data": {
"image/png":
"iVBORw0KGgoAAAANSUhEUgAAAYIAAAD4CAYAAADhNOGaAAAABHNCSVQICAgIfAhkiAAA
AAlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9
uMy4xLjMsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy+AADFEAAAgAElEQVR4nO3dd3hU95Xw8e
8ZVVRBSEINAaIXgQCZYjvFNsYUG9yDnTgkcdbObuxN22zKm02yfpNNNptN2WyaY8d2bMfYw
YVujFtcAIMokugIVRRAYSEunTePzTkVbCompk75XyeZ56ZuXPv3KNhmHPvub8iqooxxpjQ5X
I6AGOMMc6yRGCMMSHOEoExxoQ4SwTGGBPiLBEYY0yIC3c6gCuRnJJysw4cPdzoMY4wJKNu2
batT1ZRzlwdkIhg+fDgFBQVOh2GMMQFFRI72tdxKQ8YYE+IsERhjTIizRGCMMSHOEoExxoQ4
SwTGGBPiPJIIROSPIlIjIrvO87+87qIyP+ISImIFInItF6vLRWg+7bUk/EY4wx5tJ56ozgSWDeBV
6fD4x23x4AfgsgIknA94CZwAzgeyIyyPALLrLIY+JP2jHm9WUQGi
kg68HFgg6qeABCRDfQkl0Oc8EVewae/sZsvhE9Q0tnK6pYPG1k7SBw5g5ogksgYNQEScDtEYv3
H8dCvvHqzjTFsnXd1KtyojU+OYnTOY6Igwp8PzK77qUJYJlPV6Xu5edr7lHyIiD9BzNkF2drZ3ov
RDqkpReQMvbS9nZWElJ5s7+lwvSGaORNSefj60Qxjil
tER7i4ZmQyi/IyuGVyBi6XHUD5KhH09UnrBZZ/eKHo8CjAPn5+SExm05NYvfeXkXr+05TlS4
ixsnDOHWvExGpsaREBa1qa2XLkBJtL63l+axnLt5z+WtzePBjOcRHRzj9JxjxjE6rKyz
sq+OGavdSfaSc3M3/mTuGOROGkBwXXRZj7bLmw/BRv7avhjX01fGnZr6pYJ7IxIxEh/8
CZ4mnZihzl4ZWq+qkPl77PfC2qj7nfr6rLQx4GPq+qDfa13Pvn5+RrMQ0yoKisLK/neyt3t3f
x5Tmj+dSsYSRc5P/WH0zP31tPys1Mc44UneGb71UzKbSeqZmD+Q6/CZ93chj0e1cv1jC+vXA8UeHBXTISkW2qmv+h5T5ZKBAuBh4/h9V
Ht0666A97d7fy4vZyfrxuHyeb+yeb27lv1jC+vXA8UeHBXTISkW2qmv+h5T5KBAuBh4AF9FwY/h9V
neG+WLwNONuKaDsw/ew1g/MJ5kTQ1a18+6Vini8oY9qQRPztz2ZxU3YlBixKzAJ5ZmJpO3n

 "mNmGhl"Ð[Ô¾ÀÈ£'Ö¯Ã+f

Zwo9uz+WO6VleitYYZ+0sO8Vnn9hCV7fyjfnjuOeq7Msq9TS0dPDzDQd4cuMRZuUk8fv78kkc
ELxn0l5NBCLyHD1H98nAcXpaAkUAqOrvpOcq5v/ScyG4Gfisqha4t/0c8G33W/1QVZ+42P6CN
RF0dHXzled3srqoii9eN5KvzBlDeNiVNexqaO7gH5/dxsZD9fzTx0fyL3PHWi3UBJV3DtTyhWe2
kRwXxdP3z2DY4Ngrfq9XdlTw9eWF5CTH8eTnriI9cYAHI/UfXj8j8KVgTARtnV089OcdbNhznG
/NH8eDHxvZ7/fs6Ormuyt289yWY3zumhF895YJHojUGOetKqzkqy/sZFRqPE999ipSPdBA4r2
DdXzhmW3ER4fz3D/MYnjylScWf3W+RGA9i/1AV7fyT89sZ8Oe4/z7ookeSQIAEWEu/uO2S
Xz2muH88f3DPPZuqUfe1xgnbTxUx1ee38nUoYNY9sAsjyQBgGtHJ/P8g7No7ejic09upeE8LfS
CkSUCP/CT9ft4Y18NjyyeyNKrh3v0vUWE7yycwPxJafxgzV5WF1V69P2N8aXDdWf4x2e2Mz
w5lsc+4/l6/sSMRH5/Xz5lJ5v5wjPbaO/s9uj7+ytLBA5bVVjJ7/9ayidnZvPp2cO9so8wl/DzT+
Rx1fBBfPX5QrYeueC1eGP8UkNLB/c/tRWXwONL8y/aiu5KzRiRxH/eMZlNpfX82yu7CMTy+e
WyROCgPZWn+dflReQPG8T3bpno1X1FR4Txh0/nkzloAA//eQenmtu9uj9jPKmrW3n4uR0cq2
/mt5+a3q8Lw5fi9mlZPHz9KJ4vKOPJjUe8ui9/YInAIadbO3jwmQISBoTzm09NIzLc+/8UA2Mi
+Z8lU6lrauNbLxWHxJGOCQ6Pv1fKOwdqeWTxJGblDPbJPr8yZwxzxqfyo3X7OHi80Sf7dIolA
of8aO0+Kk628JtPTic13ndDQuRmJfK1uWNZt6uav2wr99l+jblSB4438tP1B5g7YQj3zBjqs/26
XMKP75hMfFQ4X3lhJx1dwXu9wBKBA94vqeO5Lcf4h4/kMH2Y7wdbfeCjOczKSeL7K3dzpO6
Mz/dvzKXq6Ormay8UEhcdzg9vy/X5wIrJcVH88LZcdlWc5ldvHPTpvn3JEoGPnWnr5BsvFpGTH
MtXbhzjSAxhLuFnd+cR7hK+9pdCurutRGT802/fPkRxRQM/uHUSKfFRjsQwb1Iad0zL4tdvH2L
HsZOOxOBtlgh87Cev7qPiVAs/uXOyo0PhZgwcwL/dPIFtR0+yfLuViIz/2VN5mv954yCLpmSwI
Dfd0Vi+t2gCaQnRfH15UVCWiCwR+NC2oyd4atNRls4eTr4fDAZ3x7QspmUP5D/X7QupzjPG/6
kq31+1m4QBEfz7Iu+2qLsUCdERfH/RREpqmnh601Gnw/E4SwQ+0t2tPLJqD2kJ0fzrvLFOhwP
0XAx7ZPEkTja3898b9jsdjjF/s7a4mi2HT/C1uWMYFBvpdDgAzBmfykdGJ/Pz1w9Q39TmdDge
ZYnAR1YUVlBY3sDXbxpLTKSvpoG4uEmZiXxq1jCe2XyU3ZUNTodjDK0dXfzH2r2MS4tnyVX+
MwmViPC9WybQ0t7FT18LrgMnSwQ+0NLexU9e3U9uZiK3Te1zAjZHfe3GsQyKieS7K3Zb3wL
juD+8U0rFqRa+d8tEwvxsxNxRqfEsvXo4y7aWUVwePAdOlgh84LF3S6lqaOU7C8f75VDQiTERf
P2msWw7epL1u487HY4JYVUNLfzm7UPMn5TG7JG+6Th2uf75htEkxUTy/VXBc+BkicDLjp9u5
bd/PcS8iWnM9FGPyCtx5/QscpJj+dmG/XRZc1LjkP9+7QBdqnx7wXinQzmvxAERfG1uz4HTG
3trnA7HIywReNkv3zhIR1c335w/zulQLig8zMVX547hwPEmVhZWOB2OCUGH687w0vZy7ps1
jKFJMU6Hc0F35WeRnRTDzzYcCIp+OB5JBCIyT0T2i0iJiHyzj9d/LiI73bcDInKq12tdvV5b6Yl4
/EXFqRb+UlDGJ64aGhCTXCyYlM749AR+vuFgULaVNv7tV28cJDLcxRc8NB+HN0WEufjSDaPZ
U3Wa9burnQ6n3/qdCEQkDPg1MB+YANwjIn83FZaqfkVV81Q1D/gV8FKvl1vOvqaqi/objz/5z
VslAPzjx0c5HMmlcbmEr980hmMnmnmhoMzpcEwIOVTbxCs7K/j07OGO9SC+XLdOzSQnJZaf
v34g4MupnjgjmAGUqGqpqrYDy4DFF1j/HuA5D+zXr1WeauGFgjLuyh9K5sDAmf/0urGpTB82iF
+9UUJrR5fT4ZgQ8T9vHCQ6IowHP5rjdCiXLMwlfHlOTzl1TXGV0+H0iycSQSbQ+/Cx3L3sQ0R
kGDACeLPX4mgRKRCRzSJy6/l2IiIPuNcrqK2t9UDY3vXbtw8B8E8f9//T3N5EhH+ZO5bq0612
VmB84uDxRlYWVvLp2cMZHBcYZwNn3Zybztgh8fzi9QN0BnA51ROJoK/2kOc7T1oCLFfV3oea
2e7JlO8FfiEiff5yquqjqpqvqvkpKSn9i9jLqhpaeH5rGXdOzyJrkH9f9OrLrJwkpmUP5NF3SgP6y2
0Cwy/fOEhMRBgPBNDZwFkul/CVG0dTWnsmoM8KPJEIyoHeg4RnAeebGHcJ55SFVLXSfV8Kv
A1M9UBMjvr9X0vpVuWfAuTawLlEhC98bCTlJ1sC+stt/N/R+jOsLa7ivtnDSfKToSQu19wJaYx
MieXRd0oDtl+BJxLBVmC0iIwQkUh6fuw/1PpHRMYCg4BNvZYNEpEo9+Nk4BpgjwdicsyJM+0s
23qM26Zm+n0TuAuZM34Io1Lj+N1fA/fLbfzf4+8dJswlfO6a4U6HcsVcLuGBj+awu/I075fUOx
3OFel3IlDVTuAhYD2wF3hBVXeLyCMi0rsV5XXQ3AMv37X5XxQIGIFAJvAT9W1YBOBM9uPkprR
zf/EICnub25XMKDH81hb9Vp/nrA/6/JmMBz8kw7LxSUcWteJqkJvpulzxtunZpJSnwUv3/nkN
OhXBGPjH6mqmuBtecs++45z7/fx3YbgVxPxOAPWju6eGrTUT42JoUxQ+KdDqffFudl8rMNB/j
dXw/x8bGpTodjgszTQXLQBBAVHsZnrh7Of63fz+7KBiZmJDod0mWxnsUetLKwkrqmNv7hI4H
/xQaIDHdx/7Uj2Fx6ImhnZjLOaO3o4qmNR7hubHAcNAF8auYwYiPD+MM7pU6HctksEXiIqvL4
u4cZlxbPNaP8d0yhy7VkRjYJ0eE8/t5hp0MxQeSl7RXUn2nngY8GVvPqC0mMiWDJjGxWFVV
RfrLZ6XAuiyUCD3nnYB37jzfy+Y/k+HyCbW+Kiwrn7vyhvLqrmuOnW50OxwSB7m7lsXdLyc1M
ZFaO8zP1edLnrh2BAE9tPOJ0KJfFEoGHPPZuKanxUSyakuF0KB736dnD6VLl2c3BN0Wf8b13D
tZSWneGz39kRFAdNAFkDhzATRPTeKGgnJb2wOmZb4nAA0pqGnn3YB2fnj2MyPDg+0izB8dw
/dhU/rzlGG2dgfPlNv7p6U1HSY6LYv4kZyek95ZPzx5GQ0tHQI3iG3y/Wg54ZvMxIsKEJTP8Z
1o9T1t69XDqmtpZax3MTD+UnWjmzf013DNjaFAeNAHMGJHE2CHxPLXxaMD0wQnOfwkfam
7v5MVt5SzITSc5wMZJuRzXjkomJyWWJzdaechcuWc/OIZLhHtnBu9Bk4hw3+xh7Kk6zfYAa
W1niaCfVu6spLGtk0/NGuZ0KF7lcglLZw+nsOwUO8tOXXwDY87R2tHF81uPMWd8KumJgTMi
75W4bWom8VHh/GlTYBw4WSLoB1Xl6c1HGZcWT/6wQU6H43V3TM8iLiqcPwVYiwjjH9YUV
XGyuYNPzx7udCheFxsVzh3Ts1hbXEVtY5vT4VyUJYJ+2Fl2it2Vp/nkrGFB1/qhL3FR4dw2NZP
VxVWcam53OhwTYP60+Sg5KbFc7aeT0nvafbOH0dGlLNtyzOlQLsoSQT88s/kYsZFh3Da1z+kX
gtKSGUNp7+zm5R2B0yLCOK+4vIHCslPcFyIHTQAjU+K4dlQyy7aW+f0MZpYIrtDJM+2sKqrkt

mmZxEV5ZMimgDAxI5HJWYks21IWMC0ijPOWbT1GVLiL26dlOR2KTy2ZMZSKUy28V1LndC
gXZIngCr28o4L2zm4+OTO4LxL3ZclV2ew/3sgOu2hsLkFLexcrd1ayMDedxAERTofjUzdOGMK
gmAie3+rf5SFLBFdAVXmhoIwpWYmMT09wOhyfW5SXQUxkWEDUPo3z1hZX0djWyd1XDb3
4ykEmKjyM26dlsWHPceqb/PeisSWCK1Bc0cC+6kbuyg+9Lzb0XDS+ZXIGqwqraGztcDoc4+ee3
1rG8MExzBwRXOMKXapPXDWUji716+tqHkkEIjJPRPaLSImIfLOP1z8jIrUistN9+3yv15aKyEH
3bakn4vG2FwrKiAp3sSgv+MYVulRLZgylpaOLlYXnm5XUGCitbWLLkRPcfdXQkLlIfK4xQ+KZlj
2QZVv997pavxOBiIQBvwbmAxOAe0RkQh+rPq+qee7bY+5tk4DvATOBGcD3RMSvG+S3dnSxY
mclC3LTSYgOrXpnb3lDBzIuLZ5lW8qcDsX4sRcKyglzCXeG2EXicy25KpuSmia/7WnsiTOCGUCJ
qpaqajuwDFh8idveBGxQ1ROqehLYAMzzQExe8+quahpbO7k7RMtCZ4kIS64aSnFFA3urTjsdjv
FDHV3dLN9WznVjUwN+Ksr+Wjg5ndjIML89cPJEIsgEev915e5l57pDRIpEZLmInP0VvdRt/cbz
W8vITgrdemdvi/IyiQgTXtxW7nQoxg+9ta+GuqY2loTgReJzxUaFsygvg9VFVTS1dTodzod4IhH
0Vfg7txC2ChiuqpOB14GnLmPbnhVFHhCRAhEpqK11ZjL1Y/XNbCqt5+78LFyu0Kx39pYUG8l1Y
1N5ZWclnV3dTodj/MzybeWkxEfx8bEpTofiF+6cnkVLRxev7qp2OpQP8UQiKAd6p/ws4O+uIKp
qvaqebTv1B2D6pW7b6z0eVdV8Vc1PSXHmi7V8ezkiPWPumB53TM+irqmNdw46k5yNfzpxpp2
39tdwa14G4WHWOBFgWvYghg2O4aXt/ncG7Yl/oa3AaBEZISKRwBJgZe8VRKT3DBSLgL3ux
+uBuSIyyH2ReK57md9RVV7eUc61o5KDfuTEy3Hd2FQGxUTw4jb/bRpnfG91USUdXRpyPYkvR
ES4fWoWm0rrqTjV4nQ4f6ffiUBVO4GH6PkB3wu8oKq7ReQEVnkXu2fRWS3iBQC/wx8r3t
CeD/0pNMtgKPuJf5nYKjJyk70RJS4wpdishwF4vzMtmw5zgNzdanwPR4cXsF49MTQrLD5YXc
Pi0TVXjFz/oUeOScTVXXquoYVR2pqj9oL/uuqq50P/6Wqk5U1Smqep2q7uu17R9VdZT79Rn4Qn4
vGGl7ZXMCAijJsmpjkdit+5c3oW7V3drCqyPgUGSmqaKCw7xR3T7KDpXEOTYpgxIokXt5f7VZ
8CK95dgtaOLlYXVTJvUhqxITTA3KWamJHA2CHxLLfWQwZ4eUc5LiGkO1xeyB3TMimtPeNXE
zxZIrgEb+6robG108pC5yEi3DE9k51lpzhU2+R0OMZB3d3Ky9sr+OiYFFLjQ7vvwPksyE0nKtzFS
9v9pzxkieASvLS9gtT4KK4Zlex0KH7r1rxMRGCFn9U+jW9tPlxPZUOrXSS+gPjoCG6amMaqokr
aOrucDgewRHBRJ8608/b+GhbnZRBmfQfOKzUhmqtHDmZFYaVf1T6Nb720vYL4qHDmThjidCh
+7fZpmZxq7uDt/f7R7NoSwUWsLqqks9uawV2KxXmZHK1v9qvap/GdVndnqXmT0oiOCHM6HL
927ahkBsdGsnKnfzSwsERwES/vqGBcWrw1g7sE8yalERnuYoWffLmNb725r4amtk5utWtpFxU
e5uLmyem8vve4XwzlbongAo7VN7Pj2CkW59kX+1IkREdww7jUnrMoG3Ii5KzYWUFKfBSzckJj
cvr+WpSXSVtnN6/tPu50KJYILuRsu/hbpqRfZE1z1uK8TOqa2v1+jlbjWQ0tHby1v5ZbJtu1tEs
1LXsgWYMGsMIP5vSwRHABK3ZWkD9sEFmDYpwOJWBcNy6FhOhwv6l9Gt9Yv7ua9s5u6ztw
GUSExXkZvHewltpGZ6extERwHvuqT3PgeJN9sS9TVHgYC3LTWb+7mpZ2/2gaZ7xv5c5Khg2O
YUpWotOhBJTFeZl0K6xxuFe+JYLzWLmzkjCXsCDXykKXa1FeBmfau9iw1/nap/G+mtOtbDxU
x+IpGSE7HeWVGjOkpyGK0+UhSwR9UFVWFlZyzahkkuOinA4n4MwaMZghCVGs8oPap/G+1U
VVdKsNKXGlFudlsOPYKY7VNzsWgyWCPmw/doryky0snmJf7CvhcgkLczP46/5aTvtB0zjjXSsK
K5mYkcCo1HinQwlIt7h/Z1YWOtcr3xJBH1YVVhIV7mLuROsdeaVunpJOe5d/NI0z3nOsvpnCsl
MssoOmK5Y5cADThw1idVGVYzFYIjhHZ1c3q4uquH5cKvHREU6HE7CmDh1I5sABrLahqYPa2Sb
WCyfbtbT+uGVyOvuqGympaXRk/5YIzrHl8Anqmtr+drpmroyIcMuUDN47WMfJM+1Oh2O8ZH
VRFVOzB1oT635akJuOCKwqdOaswCOJQETmich+ESkRkW/28fpXRWSPiBSJyBsiMi0ist
N9W3nutr62qqiKmMgwrhub6nQoAe/myel0diuv7va/ybpN/x2qbWJv1WlunmwHTf2VmhDNzB
FJrC5yZtDGficCEQkDfg3MByYA94jIhHHnW2wHkq+6vtahqnvu2CAd1dnXz6q4q5ow
fwoBIGzSrvyZmJJCTHGuth4LUmqIqRGChNbH2iJsnZ3Co9gz7qn1fHvLEGcEMoERVS1W1HVg
GLO69gqq+papn20ZtBvxyKM+Nh+o52dxh9U4PERFunpzO5tJ6ahpbnQ7HeNjqokquGpZEWqJ
NQOMJ8yelEeYSRw6cPJEIMoGyXs/L3cvO535gXa/n0SJSICKbReTW820kIg+41yuorfXOGN5
riqqIiwrnY2NSvPL+oeiWKRl0K6wrtvJQMDlwvJEDx5u42cbh8pjBcVFcPXIwq4uqfF4e8kQi6Ksr
YZ9/hYh8CsgH/qvX4mxVzQfuBX4hIVKFXNT0nx/A91e2c3r+6u5sYJQ2wsdQ8aP
SSesUPirfVQkFldWIlLYP4kSwSedPPkdI6daKa4osGn+/VEIigHhvZ6ngV86H+9iMwB/g+wSFX/
NsKSqla670uBt4GpHojpsr1fUkdSwc3W1nI4xZOTqfg6EmqG6w8FAxUldVFVczKGUxKvPW89
6SbJqYR7hKf9ynwRCLYCowWkREiEgksAf6u9Y+ITAV+DiyIWSqOm1fJCIRLkfJwPXXAHs8ENNl
11URXx0ONeOtnmJPW1BbjqqsG6Xcx1mjOfsrWqktO6MtRbygoExkXxkdDJrfFwe6nciUNVO
4CFgPbAXeEFVd4vIyIyVtJyXTkZ67ZehPTBkS1bgbWkkT6sHRx8eHT9JmKbf5e2+6DRS2aLyyYS9mJ0Z
FvK0UalxjEuLZ22xJYJgsKa4Z0DGm6znvVcsyE2n4lQLheW+Kw+Fe+JNVHUtsPacZd/t9XjOeb
bbCOR6Iob+ePdAHY2tndZayIsW5Kbzsw0HqG5otVYmAUxVWVtczeycwQy2ARm9Yu6ENL4d
Vsza4iryhg70yT6tZzGwtriKhOhwrhlpZSFvOTuct5WHAtveqkYO152x4dm9KDEmgmtH+bY8FP
KJoK2zZ9z8uRN7Jl433mHloeCwtrjKykI+4OvyUMj/8r1f4i4L2RGO1y3MTWfrEWs9FKh6ykJ
VzMpJsrKQl82dkEZEmPjswCnkE8GaouqestAoKwt524LJVh4KZPuqe1oLWVnI+xJjIrjGh+Whk
E4E7Z3dbNhTzY0TrCzkCyNTespDaxwcd91cubXFVbikp6278b6z5aEiH5SHQvrX7/2SOk63drJ
wsn2xfWVhrnUuC0Sqyprink5kNn2rb9zkLg+t8UF5KKQTwZriKuKjrCzkS/PdZYVXrTwUUPZVN
1Jaa2UhX/JleShkE0F7ZzevuccWsk5kvjMqNY6xQ+JZu8sGoQsk69xloXmT7OzZl86Wh7w99l

DIJoKNh3rKQnaE43vzc9PYeuSEDU0dQNbuqmbmCCsL+drcCUMIdwlrvTx6b8gmgnXF1cRFhfO
RMVYW8rWzYw+tt7OCgHDweCMlNU0syLWzAV8bGBPJ7JGDWbfLu+WhkEwEHV3drN9TzZz
xqVYWcsDo1DhGpsR6/SjHeMba4mrEWgs5ZkFuOkfrm9lTddpr+wjJRPBB6QlONXf87cKl8S0R
YWFuOh8crqeuqe3iGxhHrdtVxVXDkkhNsDGinDB3whDCXOLVyZ1CMhGs3dUzQb3NROac+bn
pdCust4nt/dqh2ib2VTcy38pCjhkcF8WsnCTWFnuvPBRyiaCrW1m/q5rrx6XaTGQOGpcWz4jk
WJvC0s+96r6OY62FnDV/UjqldWc4cLzJK+8fcolgy+ET1J9pt9ZCDhMR5k9KY1NpPSfOtDsdjj
mPtcVVTMseSHriAKdDCWk3TUxDBK+NPRRyiWDdriqiI1x8fKyVhZy2IDedrm5lwx47K/BHR+v
PsLvytBO0+YGU+ChmDE/y2jhdHkkEIjJPRPaLSImIfLOP16NE5Hn36x+IyPBer33LvXy/iNzkiXj
Op7tbWbermuvGphIT6ZE5eUw/TMxIIDspxloP+al1VhbyKwty0zlwvImSmkaPv3e/fw1FJAz4
NXAjPRPZbxWRledMOXk/cFJVR4nIEuA/gU+IyAR65jieCGQAr4vIGFXt6m9cfdl27CS1jW3W
WshPiAjzc9N4/N3DNDR3kBgT4XRIppd1xVVMzkoka1CM06EYejpixkaFk+aFMp0nzghmACWq
Wqqq7cAyYPE56ywGnnI/Xg7cICLiXr5MVdtU9TBQ4n4/r1hbXEVkuIvrx6V6axfmMs2flE5nt7
Jh73GnQzG9lJ9sprC8wcpCfiQ1Ppo7p2cRF+X5aoYnEkEmUNbrebl7WZ/ruCe7bwAGX+K2AIjI
AyJSICIFtbW1VxRoV7cyb2KaVz5Ic2WmZCWSkRhtg9D5mbOtheZbWSgkeOIXUfpYdm5j1/
Otcynb9ixUfRR4FCA/P/+KGtM+sniSz+YANZempzyUztObjtLY2kF8tJWH/MHa4iomZiQwbHC
s06EYH/DEGUE5MLTX8yyg8nzriEg4kAicuMRtPaqnImX8yYLcNNq7unlzX43ToRigqqGF7cdOW
VkohHgiEWwFRovICBGJpOfi78pz1lkJLHU/vhN4U3sOzVcCS9ytikYAo4EtHojJBJCpQwcxJCHK
Jrb3E1YWCj39Lg2paqeIPASsB8KAP6rqbhF5BChQ1ZXA48DTIlJCz5nAEve2u0XkBWAP0Al80
Vsthoz/crmE+ZPSeW7LMc60dRJr13Acta64mnFp8eSkxDkdivERj/QjUNW1qjpGVUeq6g/dy7
7rTgKoaquq3qWqo1R1hqqW9tr2h+7txqrqOk/EYwLP/ElptHV289Z+Kw85qeZ0K1uPnvD8HPc
AABUuSURBVGD+JCsLhZKQ61ls/FP+8CSS46Js7CGHrd9djSo290CIsURg/EKYS5g3aQhv7quh
pd2qg05ZW1zNqNQ4Rg+JdzoU40OWCIzfWJCbTktHF29becgRdU1tfHC43i4ShyBLBMZvzBie
xODYSJvY3iHrd1fTrViz0RBkicD4jfAwF3MnpvHm3uO0dlh5yNfWFleRkxzLuDQrC4UaSwTGry
zMTedMexd/PXBlw4iYK1Pf1Mbm0hPMz02zTpchyBKB8Sszc5IYFBPBOutc5lOv7TlOV7daWSh
EWSIwfiUizMXcCWm8vrfGykM+tLa4iuGDY5iQnuB0KMYBlgiM31kwOZ2mtk7eO1jndCgh4eS
ZdjYeqmd+brqVhUKUJQLjd64eOZjEARE29pCPvLanmq5uZaGVhUKWJQLjd3rKQ0PYsOc4bZ1
WHvK2NcXVZCfFMDHDykKhyhKB8UsLJ6fT2NbJuwesPORNp5rb2VhSxwIrC4U0SwTGL10zKp
nEARGssfKQV722+zid3WpjC4U4SwTGL0WEubhp4hBe32Ody7xpdXEV2Ukx5GYmOh2KcZAlA
uO3Fk7O6CkPWeshrzh5pp33S+pYONnKQqHOEoHxW1ePHMzAmAjWFHl19tKQtX63tRYyPfq
VCEQkSUQ2iMhB9/2gPtbJE5FNIrJbRIpE5BO9XntSRA6LyE73La8/8ZjgEhHmYt7ENDZYecgr
1hRXMSI51loLmX6fEXwTeENVRwNvuJ+fqxn4tKpOBOYBvxCRgb1e/7qs5rlvO/sZjwkyC2zsIa
+ob2pj46F6FlprIUP/E8Fi4Cn346eAW89dQVUPqOpB9+NKoAZI6ed+TYiYPXIwg2Ksc5mnvXq2
LDTZykKm/4lgiKpWAbjvUy+0sojMACKBQ70W/9BdMvq5iERdYNsHRKRARApqa+3oMFREhLmY
N6mnPGQzl3nOmqIqclJsyGnT46KJQEReF5FdfdwWX86ORCQdeBr4rKp2uxd/CxgHHAhHXAUkAd84
3/aq+qiq5qtqfkqNVCEklsmZ9Dc3mUT23tIbWMbm0vrudnKQsYt/GIrqOqc870mIsdFJF1Vq9
w/9H3+TxWRBGAN8B1V3dzrvc+e77eJyBPv1xW9CYkzMwZTHJcFKsKK22YZA94dVcV3drT
PNcY6H9paCWw1P14KbDi3BVEJ4GfiTqv7lnNfS3/WFXf2MxwShMJdw8+R03txXQ2Nr
h9PhBLyVhZWMGRLHWCsLGbf+JoIfAzeKyEHgRvdzRCRfRB5zr3M38FHgM300E31WRIqBYiA
Z+EE/4zFB6pYp6bR1dvP63uNOhxLQKk61sPXISRAapaD9zQx/IC4PPux88
Az5xn++v7s38TOqYOHUTmwAGsKqzitqlZTocTsM52zrvZykKmF+tZbAKCy10eeudALaea250O
J2CtLKxkSlYiw5NjnQ7F+BFLBCZg3DIlg85u5dVd1U6HEpBKa5vYVXkAEjIkZCY
xIjmWVjT10RVYWViKCJQLzIZYITMAQEW6ZnM6mQ/XUnG51OpyAoqqsLKxk5ogkhiREOx2
O8TOWCExAWZSXSbfCqiIbcuJy7K48TWntGRZNyXQ6FOOHLBGYgDIqNY5JmQms2FnhdCgBB
ZVVhJeEuYf4km4nMfJglAhNwbs3LpKi8gUO1TU6HEhC6u3vKQh8dk8kwzF+yBKBCTi3
MnAJbBih50VXIrNh+upamjltqlWFjJ9s0RgAs6QgUhM3MKzsrUVWnw/F7r+yoIC4qnDnjhzgd
ivFTlghMQFqlo8GxE81sP3bk6VD8Wm0WFOh2P8lCUCE5DmTUojjGpkXlF4Lb06x+uKq5k3Kv
W9x2ls6+R2KwuZC7BEYAJSfHQEcyYMYGi4wM9EBYVRFRFR1d3RffIES9sqOCtIRoZuYMdjoU48
teJifOtPOOzWfcpxNn2nl7fy2L8zIlC9kENOb8LBGYgPWxD/VldVElnd3rKrVY
WMhdhicAErMhwF4vzMtiw57iNSNqHl3dUMC4tnvHpCU6HYvxcvxKBiCSJyGYRKYjARXhOei+H3Se9bp6
TUqzstfyESLygXv7592zmRlzye6cnkV7VzcrC20gRm2ut5KaprYcewUt0+zswFzcf09I/gm8Iaqjgbe
cD/vS4uq5rlvi3ot/0/g5+7tTwL39zMeE2ImZiQyPj2B5dvKnQ7F+xlWxlhLrFJfMwl6W8iWAw
85X78FD3zDl8S9zzF1wPLr2R7Y866c3oWReUN7K9udDoUv9DZ1c1L2yu4bmwqKfFRTodjAkB/
E8EQVa0CcN+nnme9aBEpEJHNInInw34Py4HznseKyAPu9yiorbV7GXSwRGw5yXQbhL
eHG7nRUA/PVALbWNbdydb2cD5tJcNBGIyOsisquP2+LL2E+2quYD9wK/EJGRQF/t2c47XoCq
Pqqq+aqan5KSchm7NsEuOS6K68al8sUYbQ6S68al8tL2CjqtTwF/KSgnOS6S68ad77jMmL930USgqnNUdVIftx
XAcRFJB3Df15znPSrd96XA28BUoA4YKCLh7tywGtq452DoX22WN/Uxut7j3
Pb1EwiwqxRoLk0/f2mrASWuh8vBVacu4KIDBKRKPfjZOAaYI/2jBb2FnDnhbY35lJcNzaVpNhI

nt9a5nQojnplZ0/fgbvyhzodigkg/U0EPwZuFJGDwI3u54hIvog85l5nPFAgIoX0/PD/WFX3uF/7
BvBVESmh55rB4/2Mx4SoyHAXd07P4o29NSE7jaWq8peCMqYMHciYIfFOh2MCSL8SgarWq+o
NqjrafX/CvbxAVT/vfrxRVXNVdYr7/vFe25eq6gxVHaWqd6lqW//+HBPKllw1lM5u5S8h2pS0q
LyBfdWN3DXdLhKby2NFRBM0clLimJWTxLKtx+juDr15Cp794CgxkWEszstwOhQTYCwRmKBy
z4xsyk608F5JndOh+FRDSwcrCytZnJdJfHSE0+GYAGOJwASVeZPSGBQTwXNbjjkdik+9vL2c1
o5uPjkz2+lQTACyRGCCSlR4GHdOz2LDnuPUNIbGRWNV5dkPjjElK5FJmYlOh2MCkCUCE3SW
zMims1tDZvyhrUdOcrCmiU/OHOZ0KCZAWSIwQWek+6Lxnz84RlcIXDR+9oOjxEeHc/OUdKd
DMQHKEoEJSp+ePZzyky28sfe406F4VX1TG+uKq7ljWhYxkeEX38CYPlgiMEFp7oQhZCRG8+T
GI06H4lXPF5TR3tXNvXaR2PSDJQITlMLDXNw3ezgbD9Wzr/q00+F4RUdXN3/aeJRrRg22nsS
mXywRmKC15KqhREe4eCpIzwrWFldRfbqV+68d4XQoJsBZIjBBa1BsJLdNzeSl7RWcPBNccxqr
Ko+/d5iclFg+PsaGmzb9Y4nABLWlVw+nrbObZUE2KmnB0ZMUlTfw2WtG4HL1NbWHMZfOEoE
JauPSErh65GCe3nSEjiCatObxdw+TOCCCO2xyeuMBlghM0PvcNSOobGhldVFwzHtUdqKZ1/ZU
c+/MbGsyajzCEoEJetePS2XskHh+89ahoBiV9In3j+ASYens4U6HYoKEJQIT9Fwu4Z+uG8nBmiZ
eD/AOZnVNbfx5y1EW5WWQlhjtdDgmSPQrEYhIkohsEJGD7vtBfaxznYjs7HVrFZFb3a89KSK
He72W1594jDmfhbnpZCfF8Ou3D9EzS2pgeuzdw7R1dvPF60Y5HYoJIv09I/gm8IaqjgbecD//O
6r6lqrmqWoecD3QDLzWa5Wvn31dVXf2Mx5j+hQe5uLBj+VQWHaKTYfqnQ7nipw8087Tm45
w8+QMRqbEOR2OCSL9TQSLgafcj58Cbr3I+ncC61S1uZ/7Neay3TEti5T4KH79donToVyRJ94/
zJn2Lh6yswHjYf1NBENUtQrAfX+xni1LgOfOWfZDESkSkZ+LSNT5NhSRB0SkQEQKamtr+xe1
CUnREWH8w0dG8H5JPduPnXQ6nMtyurWDJzYeYd7ENMam2XASxrMumghE5HUR2dXHbfHl7
EhE0oFcYH2vxd8CxgFXAUnAN863vao+qqr5qpqfkpJyObs25m8+OXMYg2Mj+a9X9wfUtYI/bT
xCY2snD11vZwPG8y6aCFR1jqpO6uO2Ajju/oE/+0Nfc4G3uht4WVU7er13lfZoA54AZvTvzzH
mwmKjwnno+lFsKq3n3YOBMa9xQ3MHf3j3MNePS7UZyIxX9Lc0tBJY6n68FFhxgXXv4ZyyUK8
kIvRcX9jVz3iMuah7Z2aTOXAAP1m/LyD6Ffz67RJOt3bw9ZvGOh2KCVL9TQQ/Bm4UkYPAje
7niEi+iDx2diURGQ4MBf56zvbPikgxUAwkAz/oZzzGXFRUeBhfvXEMuypOs3ZXldPhXFDZiWae
fP8Id0zLYnx6gtPhmCDVr/7pqloP3NDH8gLg872eHwE+NCiKql7fn/0bc6VunZrJo++U8tP1+7lp
YhoRYf7Zt/Knr+3H5YKvzR3jdCgmiPnnt98YLwtzCV+/aSxH6ptZtuWY0+H0qaj8FCt2VnL/tSN
ITxzgdDgmiFkiMCHrhvGpzByRxE9fO0BdU5vT4fwdVeVHa/eRFBvJgx8b6XQ4JshZIjAhS0T4w
a2TONPWyY/W7nM6nL+zqqiKTaX1fHnOaBKiI5wOxwQ5SwQmpI0eEs8DH83hxe3lbC71j6En
Tp5p599X7mZKViKfnDnM6XBMCLBEYELew9ePJmvQAL7zyi7aO52fvOaHa/fS0NLBj26fTJjNP
mZ8wBKBCXkDIsP490UTKalp4g/vljoay3sH61i+rZwHP5bDhAxrLmp8wxKBMcAN44cwf1Iav3z
9ILsqGhyJoaW9i2+9XEROciwPXz/akRhMaLJEYIzbf9yWS1JsJA8/t4Omtk6f7/97K3dRdqKF
/7g9l+iIMJ/v34QuSwTGuA2KjeSXS/I4Wn+G777i29FOnt96jBcKynn4+lHMyhns030bY4nAmF
5m5gzmSzeM4aUdFby4rdwn+9xV0cC/rdjNtaOS+fIc60FsfM8SgTHneOj6UcwckcR3Xtnl9XkL
TjW384VntjHYfTZirYSMEywRGHOOMJfwv/dOIzUhis8+sZUDxxu9sp/m9k4eeHobx0+38utPT
mNw3HnnZTLGqywRGNOHlPgonrl/JlHhLu57/APKTnh2dtWW9i4+9+RWCo6c4L/vzmNa9iCPvr
8xl8MSgTHnMTQphj/dP4OW9i7ue/wDyk96Jhm0tHdx/1Nb2XL4BD+7O49FUzI88r7GXClLB
MZcwLi0BJ747Azqm9pZ9L/vs7Gkf7OaVTW0sPSPW9hUWs9P75rCrVM/NDq7MT5nicCYi5g+
bBArHrqGwbGRfOrxD3j0nUNXNN/x+t3VzP/lu+yqbOAXn8jj9mlZXojWmMvXr0QgIneJyG4R
6RaR/AusN19EPhGRQyLylXNLP5lMLcxm/EY4y05KXG8/MVrmDcpjf9Yu4+7freJdw/
WXlJCOFbfzDeWF/Hg09sYOiiG1Q9fy+I8OxMw/kOu5MjmbxuLjAe6gd8D/+KemezcdcKAA/R
MZVkObAXuUdU9IvIC8JKqLhOR3wGFqvrbi+03u0Pz9fCwo+tCtjvE5VeW5LGb968yBVDa1MzR7
IvTOymZSZyKjUOCLCXHR3K7VNbeyqaODZD47x1v4aXCJ8/iMj+NqNY4kMtxNx4wwR2aaqH
zpo7+9UlXvdb36h1WYAJapa6l53GbBYRPYC1wP3utwtd7Cvg+cNFxEYIxTRIR7Z2Zzx/RMlm8r5z
dvHeLry4sAiAx3kRIXRU1jKx1dPQdYyXFRPHzdKO6ZmW2zjm2zjBm/1a9EcIkygbJez8uBmcBg4JS
qdvZaft7zZRF5AHgAIDs7+8+e6nc7xZXTH8Ab3j8D40+Q3/qJOm1AbxSwxTGVQ+e8vcF3ty1ty9/WMbMUxSXr9IzHyBY3jMgLx9SWtWkDEK2f8bWbjkZ3UwxpNOGFwttw2Ryl4Ywsnjk6L8Ez5SFiXcCzbNM0smKlu7nKHeAJtJ5gUUIiY3KjGb0O+H1aPG5y5wq9+mGNgWm0Zd6v0cM0TzKhhJrCAKbQtFRcFHN6ZJ0Afk5KLUUSuQHgjQCIijwBXh3g+yo2CQ9Atv+TTPcpbN5oAohtwb0jx1UhjAXSnd+P6kuH0/vptsoTy/8+Tdg5Z4Z+X2S5aO+/+QmDbwH+UJOiEuanqI/Z2LwB3hYH7iaGFIsy+oHvk4Wl1ldKCSe3rzjrc3wOA0Mf4c3MHoS88Pc5QwJaCmXfK/u7L2wvIsMI0yAf8hjyDAzMoHMqSiAcaXJRh1KTW9oSNK

VnQqcAULtetkgeqoBI4AMIFZEPuVsVM4KpURQDgzt9TyLEDsdFJEIepLAs6r6ktPxOOAaYJGI
HKGnNHi9iDzjbEg+Vw6Uq+rZs8Hl9CSGUDIHOKyqtaraAbwEXO1wTI4KpUSwFRgtIiNEJJKei0
MrHY7JZ0RE6KkL71XVnzkdjxNU9VuqmqWqw+n5939TVUPqSFBVq4EyERnrXnQDsMfBkJxwD
JglIjHu/xc3EGIXzM8V7nQAvqKqnSLyELCenlYCf1TV3Q6H5UvXAPcBxSKy073s26q61sGYjDM
eBp51HxCVAp91OB6fUtUPRGQ5sJ2e1nQ7CPHhJmyICWOMCXGhVBoyxhjTB0sExhgT4iwRG
GNMiLNEYIwxIc4SgTHGhDhLBMYYE+IsERhjTIj7fwNC64VTR4WPAAAAAElFTkSuQmCC\n",
"text/plain": [
"<Figure size 432x288 with 1 Axes>"
]
},
"metadata": {
"needs_background": "light"
},
"output_type": "display_data"
}
],
"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)"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"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",
"execution_count": 89,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 312
},
"colab_type": "code",
"id": "TfCQHJ5AL9jY",
"outputId": "fdb9c033-0f06-4041-a69d-a0f3a54c7206"
},
"outputs": [
{
"data": {
"text/plain": [
"<matplotlib.legend.Legend at 0x121939950>"
]
},
"execution_count": 89,
"metadata": {},

"output_type": "execute_result"
},
{
"data": {
"image/png":
"iVBORw0KGgoAAAANSUhEUgAAAZAAAAEWCAYAAABIVsEJAAAABHNCSVQICAgIfAhkiAAA
AAlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9
uMy4xLjMsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy+AADFEAAAgAElEQVR4nOydd3hU55X/P0d
dQgXUCyCaKCpIdONuY0wzCBv3EjvNSTb5ZRNnk3WSXcebrHednmzijjddJHDtxN7bp2OBeMA
ZRhApVNElIQkggVFA/vz/uyFGwJAZNuXNH9/M895mZW78qd859z3uKqCo2NjY2NjYXS4DZ
AmxsbGxsrIltQGxsbGxsBoVtQGxsbGxsBoVtQGxsbGxsBoVtQGxsbGxsBoVtQGxsbGxsBoVtQ
GyGBCJyl4hsMlvHhRCRd0XkS16+5g9E9E/evKaNf2AbEBu/QUQuF5EEtItIgIvUi8pIGpIzAJQ1Wd
V9XqzNbqKiEwUkZdF5JTj59wjIg+ISOSBBz6mq/ABo/6WqXjVaNv6BbUBs/AIRiQoRiREoDs1D+
ANjN1uRMRGQ98ApQDOOaoaARAaA9wCzASizNRmMzSxDYiARQFWfV9UuVT2nqptUdQQ+AiNwn
Ih/27CwiKiJfFZGDInJaRB4TEem1/Qsistex7Q0pRiGcf5lolIiYyiccbjDpvS61r+KSKVDx34
RmedY/7CIPON4P8P8bxu7lXRI47Rjo/7HWOABF5UETKRKROF4Dsdgc/DsewdFDP4GnABsgsARGQ58APg3AgsARQ58APgJiAB+AB4foDzbAQygEQRJ/DsedvvwPjyHwEcAh5
xXCceeAX4NyAeKAMuG+A61wEr+9soIhMdOr/l0LOL/WBCsiISIyCfgGMEtVoxw/69EBrnU6U5MAm
YBzzUyxB9E1gOXAWkAqeBxwY4j40fYXsG4AAc9AVc9ifOkp8EegIg/aqqpnVPU48A6A
Q51j/FeC/VXWvqnYC/wXkUbVYC/wAKqqVtVtQP4/FcEYEqvgcnne11nMVCqqqitVtQP4
DVA9gN44AGcBzz4oGqA7bcB61V1sgsMs4WzGgEAA+vTBugdwYBpjE9twkhwIXuYO2kvvuIPIYYHB
+eYPBGXKj6Hzh/pgaQcARwm5akiKYRM/zN7hGdStsmUcU+FrXaJyRq6KyyYIZbTBdtIjB3T0q
WI5B+Na7mvXMVsJWfFgtsPKMExWKqHnlhuS2JlTS7Jg1ZYmU6iI+Wnp9WmgDp0t0Y8kd
7bvVknPWbGSfDPubzOctECFWByEXAf0ZoE4v8BwqN2lL/iCsp0X4FeV+IOUQGNGbOBBszQpKlx2PjPIJpp0GP7nm0r1xpL6Y7r5i+R3fCoHpcsJ9vkX5gLF1BmV6FULhU6kJ5wcBn0/AkzQKJ
+XShcoGN1hFNGsK2S+HGNFn9Vu9QXKVq57CHVcA79XyDnRn2f/1OvAtbkOmpjhMT/9a8/b4hWAjXY9WmB03/a+BaAb0Yf5IbVdn3VzDJFO8/XkrPjHZWbV7kXdjIOZDvBh9gfPzn9g+dWiWwJRxA5+5uSuPBD
pYHAVeFwPPj6+lk/+U5N6ZT6bwk9P9Xan0/EwCRm5X/gegJyALkMBwD1k8FAKV5Sbu9ew8
CsrC7njzHTdkSXKrhT4hLg6u1Tz2dAz6GiIbzPLJrdtnAxQ+FOiDcDZBuI5uU7hSD30xwaRiNGa5Q
LLKcG9P07sQJbspu2r/9s8v4jJ1nLw0RPf1Tw/mwxqWDcOP1Ip8hWs3DbgXWuKKw8fHMAxIKNB
EpA/60AOcBjrVzW1NtY8SMiXy4cfL4F6SzA1CLZ3BHMDfRa16EfvzM6+gHrvFhjjXRx84s6xcfShB
6T9EJc7eSeb19aI16yeRpJJlmTWvaRWjc9Qi8xB4IrQFMKjnffH+DnKx4QHfXRXpzX1xhqxfPPe
fbrTbFtY7XbBe9uUp1kIDG4Rr0u+Cj8mtM9aeAXhuJWFu6+tQJlRdaXjGe0wDc1Rv0y7VA6+v1V
s6Xr0OsWWi8WWQKLwtjQm7YGNexh7iP2aBHZuCxJp1YQaq1cTvYK/tpyrrr7cVJHv3+g2ep6PuHm
Zp0g7KWnZ3wh3lKqEBbZRTD9rXj8bz1CbA3Cr0q3Hpw9KGU7ojx2gezuBQIYFvFmYuu1kVVHmLY3
2Fb3XQLFrl6vfB7qwFe7/0Sv2Zo1WdHQZNPfjEllW6Jy8Wh+v9wQlhZjtF0tshpCDFGGnwzUGwwA
/MfhqSHvjTqR4mgr9YjkPK1zkPfvGaBv/uoNjSrGK/aGpXxS5U9Brzn/ZP3xYDbF7X8j1HrZeM4LWk
mX3HxUaOj4BtqdORXMmc7v+Jhv7zNnZRhPrGCDoZEDEYPRzV+jAohGeW8lgCdZvPsNtmBcW2idmwW0FsLP8jkh6eSrrWuqb4y5y8aHTSGhKHHwZJxdwx0x9Od4yTBWNTLr3mH8WzxTQ2nGxpXY89KOl6V6bS/+HzxlBxW06pmmEEhQTZTKMOELUnvuqbStx7/eZpk6Gxi0TU0DzsTOoDhzEeyZUw7pnVlvVc7vbQl1CMtj1FMJZqbV1TUXoJ4ti9LFcahE6+EUv+iBzTKyXp7Hsvgwk+XgM1SvGqzxFWsY0T3KsmyVTHLX62FTuX/2RxyYiTP8S7LTpHqCCuiNUelHUt9OaTvdx7AC/xAg8Bt4eo8XrJvT4AAgq
Bi8D+sG9MgdLGeAKxeA86g2Vy0NP3vdUEAk+P8c0

eYbz+9i1Ihw/vbFOaQOD3f5nG/treHrz+0kMSqM5748h5EjItyg1Lfoz4D4t+POm4jAVd81JhN
3PAVv/NAwKh6ku1v51gu7WVt4ggcXTebb8ye6ZbQQFhzIr27N5Y7Zo3nsnTKe2dpnJWeboYA
njAdAUiZ8bhUEBMIzN0FDpXvOOwAFR+v5xnO7mJISzcqvXuoW4wEwb0oSz3/5Ek63tPPFpw
pobO1wy3mtgG1A3M21/w5zvgpbH4P3fubRS/32rYOsL6riB4sn89Wrxrv13CLCT/KzmDc5kYd
WF7O5tMat57exAC318OwthvG4b737Rwlx4+GulXDuDDyzwriehyivb+Erf9tB6vAwnv78LEa4
OWR92ugR/OGuGRyqbeL/Pb+Lzq5ut57fV7ENiLsRgYWPQu4d8O5/GSG/HmBTSTW/fesgK6a
P5MtXjPPINYICA/jdndPISYvh/z2/k93lZzxyHRsfpLsLXvkSNFXD7c8bCbSeIDUP7ngO6svgpc
8Z13UzTW2dfPmvBbR3dfOne2cxPMIz+U6XZ8Tzk/xs3t1fy0/WlXrkGr6GbUA8gQjc8GtIyo
ZX74eGCree/tDJRr794m5yR8bwyI3ZHp3kjggJ4s+OycGvP7uTs0NoeD6kee+nUPYWLPoZjJz
h2WuNvRKW/haOfmBEaLmRHjfvwZNN/O9d05mQGOnW85/PnXNG8+UrxvL0x8d49hP/d/3a
BsRTBIfDLU9BVzus/IIRC+8Gmts6uf+vOwgPCeTxe2YQFuz5pL/4yFB+e/s0qs+28qPVJR6/no3
J7H/dMCB5dxsBId4g9w7IuRXe/W84vtVtp/3rx0d5c28N/7ZkCldkfKalt0d4cNEUrsiI5yfrSjly
qtkr1zQL24B4kvgM48mq/BN4+yduOeXP39jPkbpmfn/ndFJi3DMJ6Awz0kfwzWszeG1XJat3e
37C08Ykmk7Cqq9C8lRY8gvvZYyLwJJfGnXmXvkSnDvt8ikP1zbx6Ov7uHpSAvddOsZ1jU4SGC
D84pZcQoMC+faLu/16PsQ2IJ4m52bjKe6j30L5NpdOte1IPU9tOcq9c8dwybg49+i7CL5+zXhm
pI/g314rpry+xevXt/ECG78H7c2w4s/GKNqbhEUbFR0aq2DNN12KYuzqVv7l5UJCgwL56YqpXs
9lSooO4z+XZ7O7/AyPv1fm1Wt7E9uAeIPrH4HokbD2nwftyjrX3sX3VhYyKjac7y2c5GaBzhEU
GMBvbstDge+uLGQo5RANCfZtMAqHXvU9r+VlfIa0GUZFh71rYP+GQZ/mjx8cZufxM/w4P4uk
6IvIlncjS3NTWZqbym/ePEhxZYMpGjyNbUC8QWgkLP45nCw1ijAOgl9t3s/RuhZ+umIqESHml
UwYFRvBDxZPYevhetYUnjBNh42baW2A9Q8YgR+XfctcLZf+P0jMgo3/aoyGLpKDNY38atMBF
mUnsyw31QMCnecn+VnERYbwLy8X+qUryzYg3mLyYpiy1JicrD9yUYcWlp/hzx8e4a45o7l0fLy
HBDrPbbNGMXVkDP+5fu+QSpryazb/CJpqYNnvBi6M6A0Cg435kIbyi86lUlUeXltCeEggP1nu2
QhFZxgeEcLDS7PYV93I89uOm6rFE9gGxJss+pnRH2H9A077d1WNIomxw0J5cJFvlBQJDBB+n
J/NqaY2fvvmQbPl2LhKRQHs+Atc8k+QNv3C+3uD9LlGFNjHvzey4Z1kc2kNHx2q49vXZfhMXa
qF2cnMHRfHLzcf4HRzu9ly3IptQLxJdKpRwbTsbdi3zqlD1hdVsePYab67YCJRYSY/GfYib9Rwb
p81ir9sOcr+6kaz5dgMFlWj7M6wRKM0uy8x/8cQGgXrv+PUA1dbZxePbNhLRmIkd12S7gWBz
iEi/GhZJmfPdfCrzQfMluNWbAPibWZ+AeInwZv/AV2dA+7a2tHFoxv3MTk5iptnjPKSQOf57oL
JRIUF8dDqYntC3arsXQvlW+HaHxpzdb7EsDijWduxD6F01QV3/8tHRzlW18JDSzN9rj/H5ORo
7rkknWc/OcbeqrNmy3EbvvVbHgoEBhk3Rd1B2PXXAXf9y0dHqTh9jn9bkvlpZzRfInZYCN+ZP5
FPjtTz9r6TZsuxuVg62+HNH0HCFMNd5ItMu8fQ9/YjAz5wnWxs5XdvHeS6KYleSxi8WL49fyI
x4cE8vKbEbx64bANiBpMWwahL4N1H+40yOdXUxmPvHGLe5EQuzzB/4rw/bp89mtGxEfxi0w
G6++gjbePDFDxpNEKb/2PjwcYXCQiEef9uPHAVPtfvbr/efID2rm5+uCTTi+IujuERIXzb8cD13o
Fas+W4BduAmIGIcdM21Rh91fvgd28dpLWjix8smeJlcRdHcGAA356fwd6qs2worjJbjo2znDtjR
ASOvQoy5putZmAmLTY6GL77qNFS9zzK61t4uaCCO2ePZmz8MBMEOs/ts0aTNjycX20+4Bej
EFMNiIgsFJH9InJIRB7sY/uvRWS3YzkgImd6bevqtW2Nd5W7gdFzYPINRoZ686l/2FTd0Mrz2
8q5ecZIxif4mF+6D5blpjExKZJfbTrgl7HufsnHv4dz9XD9T7xXrmSwiMC8H8HZStj+p89s/t3bB
wkIEP7pmgkmiLs4QoIC+Oa8CeypaODNvdZ3+5pmQEQkEHgMWARkAneIyD+MP1X126qap6p
5wO+AV3ttPtezTVWXeU24O5n3I+hohg9//Q+rH3+vjG5Vvm6BGwKMsN7vXD+Jw6eaeXWXX
SfL5zl3Bj75PyMvKSXXbDXOMfYKGD8PPviVkfTo4OipZl7ZWcldc0ablnF+sdw0fSTpcRH8arP1
3b5mjkBmA4dU9bCqtgMvAPkD7H8H8LxXlHmLhImQfTMU/OXTZjo1Z1t5bttxbpqexqhY67TG
vD4zidyRMfz2zYO0dbq/p4ONG9n2BLSdhSu/Z7aSi2PeQ8aoaesfPl31P28fJDhQ+NrV7m2o5k
mCAwP453mG2/f1kmqz5biEmQYkDSjv9bnCse4ziEg6MBZ4u9fqMBEpEJGtIrK8v4uIyP2O/Qp
qa31w4uqKB4xRiOOmePy9Mrq6lW9c46EGPh5CxBiFVJ45xys77FGIz9LWCB8/BhMXQcpUs9
VcHKl5hu5PHof2Zg7XNrFqVyX3XJJOYpQ1Rh895OelMT5hGL/efIAuC49CzDQgfTle+/tN3g6
sVNXej7ajHU3e7wR+IyJ9PoKo6hOqOlNVZyYk+GB4X+IUYy5k2/9RW1vLc58c56ZpaYyOs87
oo4crMuKZOjKGJ94vs/RN4dds+yO0noGrvmu2ksFxxQNGqfcdT/O7tw8RGhTIV9zcztkbBAYI3
7puIgdPNvF6sXVHIWYakAqgd3bcSKC/6ny3c577SlVPOF4PA+8C09wv0Utc+S/Q2sCeVb+ks1v
5xrXWmPs4HxHha1eN52hdi6VvCr+lvdmYPJ9wnVH11oqMmg3pl9P50e/YWHicuy8Z7TMlSy6
WxTkpjImL4In3yywbkWWmAdkOZIjIWBEJwTASn4mmEpFJwAjg417rRohIqON9PHAZYN0m
xKnT6Bh7LXkVz3JzTizpcb4dijgQ12clMzZ+GI+/Z92bwm/Z8RS01Flv7uN8rvg2QU0nWB7wA
V+4fKzZagZNYIDwpSvGUVjRwNbD9WbLGRSmGRBV7QS+AbwB7AVeUtUSEfmxiPSOqroDeE
H/8dtoClAgIoXAO8CjqmpdAwKsjbmTODnLt+M+vvDOPkxggHD/leMoqmxgS1md2XJseujqNHK
OxlxhhJBbmIaUKynVMTwQsZGUqBCz5bjEzTNGEjcshCfet2bTKVPzQFR1g6pOVNXxqvqIY91
Dqrqm1z4Pq+qD5x23RVVzVDX8fpnb2t3J+2d3fy0dAT7QrJJLnnygjWyfJ2bpqeRGBXKH961
5k3hl+xdDWcrYO7XzVbiMs9sO85jHctIbC83anlZmLDgQO69dAzv7K+1ZFFSOxPdB1i35wQ1Z
9vonP01aDjuUic2XyA0KJAvXD6WDw+doqjCPzuxWQpVI/IqdjxkLDBbjUu0dXbx1JajNI5bZP
w8W/7HbEkuc88l6YQHB/LE+4fNlnLR2AbEZFSVP35whIzESLKuuR2Gj/6HOHerctec0USFBvG

nD613U/gd5dugcgdc8jUIsPYtv3rXCWob27j/qokw56vGz1VRYLYslxgxLITbZo1iTWElVQ2fLd
Xiy1j7v8kP2FJWx96qs3zpirFIYBDM/goc3wIndpstzSWiwoJZMWMkG4qqONnYaracoc3Hv4e
w4ZB3p9lKXKK7W/njB4fJTInmsglxkHcHhEQZWfUW54uXj6WrW3lqy1GzpVwUtgExmT9+cJ
j4yBDy8xw5lNPvgZBII1nK4nxubjodXcpzn/hfK0/LcPqo0bxsxn0QYt3oPoCPyk5x8GST8bAlYj
SbmnY3lLwGjdYOGx8VG8H1mcm8tL2c1g7rVHKwDYiJHK5t4t39tdx9STphwYHGyrAYyLsLilZC
Y425Al1kXEIkV01M4NlPjtPeaRdZNIVPngAJgNn3m63EZf728TFih4WwZGrK31fO/jJ0dxql6S
3O5+amc7qlg3V7rFPV2jYgJvLsJ8cJChDunDP6HzfM+Yrf3BT3XTqG2sY2y9f8sSTtzbDrb5C5H
GL6rBJkGSrPnOPNvTXcNmsUoUGBf98QNx4mLjDulc428wS6gbnj45iQGMlfPz5qthSnsQ2ISZ
xr72LljgoWZCd/to7PpzfFn42ucRbmqokJjImL4GmL+Xb9gqKVRtHE2V82W4nLPPfJMcAIzvg
Mc74CzbWGK8vCiAifm5vOnooGdpefufABPoBtQExi7Z4TNJzr4J5L0vveYdaXjJti3zrvCnMzAQ
HCPXPHsOPYaYor7ZBer6FqPIAkZsIoaycOtnV28cK2cq6dnMTIEX3UiBt3DcRP8ot5wxunpTEsJ
NAyoxDbgJjEs1uPkZEYyZyxsX3vMP5aI6R3x1+8K8wD3DJzJBEhgZaLMLE0J3ZCVSHM/ILvN
4y6ABuLqqlrbueeuf08bIkYD1wndhmLhemJXlxXWEVdk++75GwDYgJ7Ks5QWNHA3ZekG9Ekf
REQCNPvhSPvw6lD3hXoZqLDglk+LY11jlGXjRfY/iQED4Opt5mtxGX+tvUYY+IiuGJCfP87Tb0V
gsJhx9PeE+Yh7rkknfaubl7YXn7hnU3GNiAm8MzWY4QHB3Lj9AtMbE67BwKC/GIUcses0bR2d
LN6t90rxOOcOw3Fr0DOzRAWbbYalyg50cCOY6e5+5J0AgIGGEmFD4fsm6DoZWhr8p5AD5C
RFMXccXE8v+24z3cstA2Il2lo6WBN4QmWT0slOix44J2jkmDSYtj9HHRYOxkvZ2QMWanRPL
+t3K7S62kKX4TOczDri2YrcZkXt5cTEhTAzTNGXnjnGfdBe5NhPC3O7bNHUXH6nM8XJLUNiJd
ZtbuS1o5u7prTjz/3fGZ+3mjjafGicQC3zx7N3qqz7LHrY3kOVSOkNW2Gdfqd90NrRxev7apkY
VYywyOcqLo7chYkTDHK1lucBVnJxIQH82KBb7uxbAPiZV7cXk52WjTZaTHOHTD2ahgxxi/cW
Pl5qYQHB/LCdjsz3WMc3wqn9sOMz5utxGVeL66msbWT22eNuvDOYEymz7jPEUCwx6PaPE1Y
cCA3TkvjjeJqTjf7bii/bUC8SHFlA6VVZ7l1ppM3BBjF72bcB8c+gtoDHtPmDaLDglkyNYU1u0/Q
3GbtkvU+y65njFI4WTearcRlXth+nNGxEVwyLs75g6beCkFhsNP6k+m3zRpFe1c3r+3y3XlD24B
4kZcLDH9ufu5FZgXn3gkSCLuf8YwwL3LH7FE0t3extrC/7sU2g6atyUimy1oOoZFmq3GJY3XN
bD1cz60zRw48eX4+EbFG5v2el4xMfAszJSWa3FHDeXG7784bmmpARGShiOwXkUMi8mAf2+8
TkVoR2e1YvtRr270ictCx3Otd5RdPa0cXq3afYGFWMjERF5g8P5+oJMi43pgctXizqemjR5CRG
MnzFghRtBylq6Cj2YjeszgvFZQTIHDzjIsYrfcw414jA98f5g1njWJ/TaPPZqabZkBEJBB4DFgE
ZAJ3iEhmH7u+qKp5juVPjmNjgR8Bc4DZwI9EZISXpA+KTaU1NJzr4DZn/bnnM+0uaKqGsrfdK
8zLiAi3zRpFYfkZDtRYrwObT7PrGYibYPnM886ubl4uqODqSYkkx4Rd+IDzGT3XmDfc/azbtXm
bpbmpRIQE8qKPPnCZOQKZDRxS1cOq2g68AOQ7eewCYLOq1qvqaWAzsNBDOt3CywXlpA0P
Z+7F+HN7k7EAIuL8wo21fFoaQQHCKzsqzJbiP5w6BMc/NsqbWzzz/L0DtZxsbBv8w5aIUdH6y
PtwxtoBG5GhQdwwNYW1hSdoafc974OZBiQN6G1WKxzrzmeFiOwRkZUi0vMf5eyxiMj9IlIgI
gW1tbXu0H3RVJxu4cNDp7jlYv25vQkKgZxbYf9GaKl3r0AvEx8ZytWTEnhtVyWdXXaZd7ew+
1ljniz3DrOVuMzKHRXER4Zw7eTEwZ8k93bjtfBF94gykZtnGPOGb/hgRWszDUhf36TnzxStBc
ao6lTgTaAntMKZY42Vqk+o6kxVnZmQkDBosa7wyg4jisKpZKiBmHYXdLUbVVYtzorpIznZ2Ma
Hh06ZLcX6dHdB4fMw4TqISjZbjUucaWnnrb0nWZabRnCgC19Pw0fD2CsNw+qjE9DOMjN9BK
Niwz/9HvElzDQgFUDvMepI4B9Cc1S1TlV7Kor9EZjh7LG+gqry6q4K5o6L67uS6MWQnAPJU/3
CjXXtlERiwoN5Zafv3RSWo+xtaKwy3FcWZ92eKtq7urnpQmV+nCH3Tjh9xMiNsTABAcKN00by
Udkpn+uZbqYB2Q5kiMhYEQkBbgfW9N5BRHq1HmMZsNfx/g3gehEZ4Zg8v96xzufYefwMx+p
auHGamxr6TLvbqLJaXeye85lEaFAg+XmpbCqp5myrXWDRJQpfgPARMNGnpwGd4tWdFUxKiiI
r1Q01vDKXGTkxfjCZvmJ6GqqwapdvPSebZkBUtRP4BsYX/17gJVUtEZEfi8gyx27fFJESESkEv
gnc5zi2HvgJhhHaDvzYsc7neHVnBWHBASzKSbnwzs6QcwsEBMOeF9xzPhNZMX0kbZ3drLdQC
0+fo60R9q2HrJuMeTILc+RUMzuPn+Gm6Wn9V6m+GEKGGTkhJaugvcX185lIetwwZqaP4JWd
FT6VE2JqHoiqblDViao6XlUfcax7SFXXON5/X1WzVDVXVa9R1X29jn1SVSc4Fp+s89r6KGcAA
CAASURBVNHW2cW6PVUsyEomMjTIPSeNiDVyQopWGr5vCzN1ZAwTEiNZaUdjDZ69a43CiX5
Qtv21nRUEiBGl5zby7oT2Rss3ZgO4afpIDp1sosiHGrPZmege5J19tTSc63Cf+6qHqbcaPu+jH7j
3vF5GRFgxfSQ7jp3myClrZw2bRuELMGIsjJptthKX6O5WXt1VyWUT4kmKHkTuR3+Mngsxo2
GP9aOxlkxNISQogFd9aN7QNiAe5LVdFcRHhnL5QI1wBsPEhRAabZRrsDjLp6Uigt0nZDA0VBq5
DlNvs3zux/aj9VScPseK6S5GKp5PQIDRF6XsHWgyJ4zfXcSEBzM/M4nVuytp7/SN8HfbgHiIM
y3tvL3vJPl5qQS5Eo7YF8FhxgRh6RrL+3ZTYsKZMzaW1btP+JRv1xIUrwTUGJFanFd3VjIsJJD
rs5Lcf/Kpt4F2Qcmr7j+3l1kxPY3TLR28d8A3jKFtQDzEuj1VdHSp+91XPUy9zfDtHtjomfN7ke
V5aRw51Wz3CblYCl80emDEjTdbiUu0dnSxobiKhdkpRIS4aa6wN4mTjRB4P3BjXZGRwIiIYNb4
SDFS24B4iFW7KpmYFOmecMS+SL8cotP8wo21KCeFkMAAVu/2jZvCElQXwckSv5g8f3d/LY2t
neTnpXruIjm3QuUOqCvz3DW8QHBgAEumprC5tJomH2iJYBsQD1Be30LBsdPk57kpHLEveny7h
96EZmtnc8eEB3PN5ATW7jlBl4/3gPYZ9rwIAUFG+K7FWVNYSXxkCJeOH2SdOGfIuRkQo2e6
xcnPS6O1o5vNpeaXNrENiAdYu8d4kl6W68EnKjCePrs7jR4QFmd5Xhq1jW1sKbO2MfQK3d1Q
/KpRumSYB79OvUBjawdv7j3JDVM9MFfYm+hUGHuFYXgtPtc2Y/QI0oaH+8SI3TYgHmDN7hN

MHz2cUbEuli65EElZkJjlF26sayYnEhUa5HOZtj5J+VY4WwnZK8xW4jJvlNTQ3tnNMk+6r3rIu
RXqD0PlTs9fy4MEBAjL8lL54OAp6praLnyAJ7WYenU/5EBNI/uqGz0/+ughZwVUbLN82eqw4E
AWZifzRkk1rR3WTpD0OEUrISgcJi02W4nLrN5dyajYcKaNGu75i2Uug8BQKLL+A1d+Xipd3cq
GInOrONgGxM2s2X2CAIElU71kQHqeQoutH6K4fFoaTW2dvLm3xmwpvktXh9F5cNJCy7etrW
1s46NDp8jP9eBcYW/CYmDi9YbL1+JVHCYnRzMpKYpVJruxbAPiRlSVNYUnuGxCPAlRod656Ig
xkDbTkRNgbS4ZF0dCVCjrCu3aWP1y+D1oqYPsm81W4jIbiqroVjwbfXU+2SugqQaOfeS9a3qI
ZXmp7Dh2mvJ683LBbAPiRnaXn+F4fQtLveW+6iF7hRHWWXvAu9d1M4EBwpKcFN7ef5JGu0J
v3xS/AqExkDHfbCUus3p3JZOTo8hIivLeRTMWQPAw4/docXrc5GbmhNgGxI2sKTxBSFAAC7O
93NQn60ZA/OKmWJqbQntnN5tLbTfWZ+hoNYoCTlkKQV4a4XqI8voWdh4/453J896ERMDkx
VC62nAHWphRsRFMGz2cdSZWs7YNiJvo6lbW7animkkJRIcFe/fi0Skw5nLDgFg8RHHaKCNE0c
ybwmc5uAnazhqBExZnvWPyd6m35gp7k70Czp2Gw+96/9pu5oapqeytOktZbZMp17cNiJvYdqS
e2sY277uvesi+CeoOGq4sCxMQICyZmsL7B2o509JuthzfonglDEuAMVearcRl1u05Qe4oL4S698
X4a40JdT8YsS/JSUEE0+YNTTUgIrJQRPaLyCERebCP7Q+ISKmI7BGRt0Qkvde2LhHZ7VjWnH
+st1m35wThwYFcOznRHAFT8o3MZD+YTF86NZXObuWNEvMzbX2GtiY4sMlokBTogXpRXuToq
WaKK8+ydKqbmqxdLEGhhhtw7zrDLWhhkmPCmJUey7o95syDmGZARCQQeAxYBGQCd4hI5n
m77QJmqupUYCXws17bzqlqnmNZhol0dnXzenE1105J9EwxOGcYFgfjrjHCeS3uxspOiyY9LoK1
djTW3znwutE4Ktv6pUt6vuwWu6tL52DIXmEUIz202TwNbuKG3BQOnmxif3Wj16/drwERkd+
JyP/0t7jh2rOBQ6p6WFXbgReA/N47qOo7qtoTo7YVcHOzAPew9XA9dc3t5j1R9ZB9EzSUG0
XjLIyIsHRqKlvKTnHK5Exbn6HkNYhMhlGXmK3EZdbtqWJm+ghSh4ebJ2LMlRAR7xdurEXZKQ
QIpoxCBhqBFAA7BlhcJQ0o7/W5wrGuP74I9K5dHiYiBSKyVUSW93eQiNzv2K+gttYzNfTXF51
gWEggV08yyX3Vw6TFRr90P6iNdUNuCt0KG03OtPUJWs/Cwc2QtdwoomlhDp00KjXcYPbDVm
AQZObD/teh3drdMBOiQpk7Po51e6q83lOn3/9GVX269wKsPO+zq/SVetrnTy8idwMzgZ/3Wj
1aVWcCdwK/EZE+myKo6hOqOlNVZyYkJLiq+TN0dHWzsbia6zKTCAsOdPv5L4rw4TBhHpSsMg
ruWZhJSVFkJEay1o7GMtxXXW1+UXl3bWEVIia7r3rIutFwCx7cZLYSl7lhaipHTjVTcuKsV697
wccZEZkrIqXAXsfnXBH5XzdcuwIY1evzSOAzYzARuQ74IbBMVT/1Z6jqCcfrYeBdYJobNF00
W8rqONPSwQ1mhCP2RdaNcLYCKgvMVuISIkY01vaj9Zw8a+2JTpcpftXo/TJyltlKXEJVWbfn
BHPGxpLozr7ngyX9UhiW6Bcj9oVZyQQFyKeVwL2FM+Ph3wALgDoAVS0E3BFHuB3IEJGxIhIC
3A78QzSViEwD/g/DeJzstX6EiIQ63scDlwGlbtB00awrPEFUaBBXTnRz3/PBMmkRBIb4xU2xJ
CcFVdhYPISjsc6dgbK3jOgri7uv9tc0Ulbb7L06cRciINAosHhgk+XdWCOGhXDZhHg2FHnXjeXU
f6Sqlp+3yuVKZKraCXwDeANjdPOSqpaIyI9FpCeq6udAJPDyeeG6U4ACESkE3gEeVVWvG5D2
zm7eKKlmflYSoUEmu696CIuBCfP9wo2VkRTFxKTIT5POhiT7N0BXu6PagLVZv6eKAIFF3q7UM
BA9bqwDb5itxGWW5KRQXn+OokrvtYZ2xoCUi8ilgIpIiIj8Cw53lquo6gZVnaiq41X1Ece6h1R1
eP9daqadH64rqpuUdUcVc11vP7ZHXOulo8OneJsa6f5E4Lnk3UjNJ4wyrxbnMU5Q9yNVfIaxIy
CkTPNVuISqsr6oiouGRdHfKQPlWEZPRcik/xixH59VhJBAeLVBy5nDMhXga9jREhVAnmOz0Oe
9UVVRIUFcfkE90/Ou8SkhUbfAz+4KYa0G+vcaSh7x4gW8ka5cw+yv6aRw7XNvjF53puAQOP3
e3CTkaxpYYZHhHCpl91YFzQgqnpKVe9yjAQSVPVuVa3zhjhfpr2zm00l1czPTCIkyMd806FRRr
XW0tW2G8vK7NsA3R1+EX21weG+8nqhUWfIXA6drXDQ+m6sG7zsxnImCmuciKwVkVoROSki
q0VknDfE+TIflRnuqyW+9kTVQ9aN0FhltD+1OEtyUoemG6t0FcSMhrTpZitxiR731ZyxPua+6m
H0JUaSph+M2L3txnLm0fk54CUgBUgFXgae96QoK7BhTxVRoUFcnuEj0VfnM3GB4cYqXW22Ep
dZMjV56Lmxzp1xuK+WWd59daCmibLaZhb72lxhD5+6sTZDm/fLgbgTb7uxnDEgoqp/U9VOx/
IM/ST8DRU6urrZVFrD/Ewfir46n9AomHAdlK6xvBtrQmIUk5KiWD+Ukgr3bzTcV5n9FlmwDOu
LHO6rLB90X/WQ1ePGsn5S4ZKcZMrrz1Fc6fmkwoFqYcWKSCzwjog8KCJjRCRdRL4HrL4Pe4Mh
/mo0OnaDjX4XsTgueTtdyIxrJ4UiE4orGODSE3VukqiB7pF9FXG4qqmD021nttngfDqDmOaKxV
ZitxmeszjaTCdUWeTyocaASyA6Me1m3AVzDyLd4FvgZ83uPKfJiNRdVEhQZxha8kD/bHxAW
OpELr3xQ9bqzXh0KJ99YGKHvbL6KvDtQ0cehkk+/OFfYQEAhTlhluLD9IKvSWG2ugWlhjVXW
c4/X8ZchOond0dfNGqVH7ymfdVz2ExcD4ecY8iMVLvE9INGpjbRgK0Vj7XzeSBzPzL7yvj7Ohy
Kh9tcAXo6/OJzPfURvL+iXee9xYnq6N5VT8qYhki8itIvK5nsWjqnyYjx21r3wqm3YgMvMdtbGs
XeIdYFFOyqedH/2a0lUQlWr52lcAG4urmD0mlsQoH6h9dSHSLzVKvJdaf8Q+PzOZwADx+AOX
M2G8PwJ+51iuwWjqZGoDJzPZWFxFZGgQV070seTB/pi0yCjx7gc3xeKcZLoV/+5U2HoWDr1l
GH6L1746dLKRAzVNvj9X2ENAoNGp8MAmaG+58P4+TOywEOaOi/O4G8uZ/9CbgXlAtap+Hsg
FfHg2zHN0dnXzRkkN86Ykml+63VnCh8P4a6DE+m6sSUlRjEsYxsZiP3ZjHXjDKN3uB+6rjUWGo
ffJ5MH+yFoOHc1GAUuLsygnmaN1LezzYKdCZwzIOVXtBjpFJBo4CQzJOZBtR+qpb25nUbZFn
qh6yMyHhuNwYqfZSlxCRFicncLHZXXU+WunwtJVjs6Dc8xW4jIbiquZmT6CJF8o3e4s6ZdDeK
xfBJ4syEomQDzblM0ZA1IgIsOBP2JEZu0ErF+lbxBsKK4iPDiQq6zivuph0mIICDJyQizOIocba1
NpjdlS3E9bExx600getLj76sipZvZWnWWRVdxXPQQGwZQbjCZeHdYOGY+PDGXO2Dg2eDA
B15laWP+kqmdU9XFgPnCvw5U1pOjqVl4vruHayYmEh1jEfdVDRCyMvcovorEyU6JJj4vwz2isg5
uMZDZ/cF853IyWcl/1kLkc2pv8wo21OCeZQyebOFjjGTfWQImE089fgFggyPF+SFFwtJ5TTW

0syrHgDQHGl9LpI1BdZLYSlxARFueksKWsjtPN7WbLcS+lq2FYglFi3OJsKKoib9Rw0oaHmy3l4h
l7JYQN94sR+4KsZERgQ5FnRiEDjUB+OcDyC4+o8WE2FlcTGhTANZMSzZYyOCbfABLoF7WxF
men0NWtbN7rR26s9hZjBDJlqRENZGGO17VQXHmWxVZ92AoMNu6X/Ruh09pzbYnRYcxKj/
VY4MlAiYTXDLBc646Li8hCEdkvIodE5ME+toeKyIuO7Z+IyJhe277vWL9fRBa4Q09/dHcrG4ur
uHpSAsNCgzx5Kc8xLA7GXG5M0lrcjZWdFs3IEeEenRz0OmVvQUeLX7mvLBds0pvMfGhrgMPv
ma3EZRblJLOvupGyWvf3OzFtpk5EAoHHgEVAJnCHiGSet9sXgdOqOgH4NfBTx7GZGD3Us4CF
wP86zucRdpWfpuZsm3Xi2fsjMx/qDsFJtzSUNI0eN9aHjppkfkHpaiP6J/1ys5W4zIbiarLTohkV
G2G2lMEz7ioIjfaLEfuSnBT++6Ycj9QiMzPUYzZwSFUPq2o78AJw/uNXPvC04/1KYJ6IiGP9C6r
apqpHgEOO83mEDUXVhAQGcO1ki7qvepiyFBC/uCkWZSfT0aW85Q9urI5Wo3zJlBuMKCALU3
G6hcLyM9Z/2AoKNZJw962DLms/pCRGh3HH7NFEhwW7/dxmGpA0oLzX5wrHuj73UdVOoAGI
c/JYAETkfhEpEJGC2traQQk919HFdZmJRHngD+BVIhMh/TK/MCB5o4aTGhPmsclBr3L4HWhv
9Av31euOkFFLu696yMyH1jNw5H2zlfgszpQyuUxEhjne3y0ivxKRdDdcu68yo+c75/vbx5ljjZWq
T6jqTFWdmZAwuPyN/7oxh8fu9JPAs8xlULsXavebrcQlRISF2Sm8f7CWprZOs+W4Rulqo/Dlm
CvNVuIyG4urmZISzdj4YWZLcZ3x10JIJOy1fjSWp3BmBPIHoEVEcoHvAceAv7rh2hXAqF6fR
wLnF7D/dB8RCQJigHonj3UrYvGy2p8yZanx6gejkMU5ybR3dvP2vpNmSxk8ne1G7/PJN0BQiNl
qXKK6oZUdx06z2Iq5H30RHG60RNi7Dros/pDiIZwxIJ1qVOPKB36rqr8Fotxw7e1AhoiMFZEQj
Enx8039GuBex/ubgbcdWtYAtzuitMYCGQzR7PiLJjrVKJPhBzHu00ePIDEq1NrRWIffNaJ9/M
J95Yi+svr8R28y86HlFBzfYrYSn8QZA9IoIt8H7gbWO6KdXJ4McMxpfAN4A9gLvKSqJSLyYxH
pqfb7ZyBORA4BDwAPOo4twejTXgq8DnxdVbtc1TRkyFwONUVQV2a2EpcICBAWZifzzv6TtLR
b9AmxdLUR7TPuarOVuMzG4momJkUyITHSbCnuY8J8CI7wi9pYnsAZA3Ib0AZ8UVWrMSarf+
6Oi6vqBlWdqKrjVfURx7qHVHWN432rqt6iqhNUdbaqHu517COO4yap6kZ36Bky+JEba1F2Cq0
d3by7f3ABEqbS1WFE+UxaZET9WJjaxja2Ha33j8nz3oREQMZ82LsWuu1n1PNxphZWtar+Sl
U/cHw+rqrumAOxMYvhoyBtpl/0CJk9Npb4yBBr1sY68r4R5eMH7qs3SqpRxfrhu32RmQ/NJ+H
4VrOV+BwD1cL60PHaKCJney2NIuLZPok2niczH6oKof6I2UpcIjBAWJCVzNv7TtLaYbEnxNLVR
pTPeLcUdjCVjcVVjEsYxsQkP3Jf9ZCxAILC/GLE7m4GKmVyueM1SlWjey1RqhrtPYk2HiHTMc
3kByGKi3NSaGnvspYbq6vTcF9NXGBE+1iYuqY2th6uZ1F2sv9EK/YmNBImXGfcK93dZqvxKZzJ
A7muj3X39rWvjYUYMQZS8vziqWrO2Fhih4VYq1PhsY+gpc4IaLA4b5TU0NWt/um+6iFzOTR
WQYUd7NkbZybRHxKRP4jIMBFJEpG1wFJPC7PxApn5ULkDzhw3W4lLBAUGsCAribf2WsiNVb
raiO6Z8JnnM8uxsbiKMXERZKb4sWNi4gIIDPWLBy534owBuQooA3YDHwLPqerNHlVl4x16Jm
/9ICdkUXYKTW2dfHDwlNlSLkx3lxHVkzHfiPKxMPXN7Wwpq2NxTop/uq96CIuGCfMMA2K7sT
7FGQMyApiDYUTagHTx6/+UIUTceEjO8YtorLnj4xgeEWyNaKxjW4yoHj9wX20qqfZ/91UPmfl
wttIYtdsAzhmQrcBGVV0IzAJSgY88qsrGe2TmQ8V2aKgwW4lLBAcGcH1mEm+W1tDW6eNur
NJVEOQok2FxNhRXMzo2gqxUP3Zf9TBxIQQE+8UDl7twxoBcp6pPAqjqOVX9Jo6McBs/IPNG
49Uf3Fg5KTS2dfKhL7uxuruM33XGfAixdsHBMy3tbDl0yv/dVz2EDzdCrktXW74pm7twJpHwu
IiMEJHZInKliFi/ZKjN34mfAEk5UPKa2Upc5rLx8USHBbHel91Yxz823FdZ/uC+qqGzW1kyFNxX
PWQth4ZyqNxpthKfwJkw3i8B72PUrPoPx+vDnpVl41Wy8o3wRIu7sUKCArg+K5nNvuzGKlllJK
Vl+IP7qoqRI8LJThsC7qseJi023Fglr5qtxCdwxoX1zxhzH8dU9RpgGmChjC2bC+JHbqwlU1NobP
VRN1Z3l5GMljHfSE6zMA0tHXx06BRLhor7qgfbjfUPOGNAWlW1FUBEQlV1HzDJs7JsvEr8BEj
K9ovJQZ92Yx3fCk01/hF9VVpNR9cQib46n6wbHW4sOxrLGQNSISLDgVXAZhFZjYebN9mYQ
OZyKP8EGirNVuISIUEBLMhKZnOJD7qxSh3uq4kLzVbiMuv2VDEqNpypI2PMluJ9Ji1yuLGsP2/
oKs5Mot+oqmdU9WHg3zF6dFj/EcrmH+mZ1PWD2lhLphrRWB8c8CE3Vne34SKccJ3l3Venm9s
d7qvUoeW+6iF8+N+TCoe4G8uZEcinqOp7qrpGVds9JcjGJOIzDDeWHzxVXTYhnpjwYN9yYx3/
GJqqDfeHxdlUWk1nt3LD1CHovurBdmMBF2lA3IWIxIrIZhE56Hgd0cc+eSLysYiUiMgeEbmt17a
nROSIiOx2LHne/Qn8lE/dWNaOxgp21MZ6s7TGd2pjlbzqSB70D/dVetwQSR7sj0mLIDDELx64
XMEUA4KRiPiWqmYAb9F3YmIL8DlVzQIWAr9xzMX08F1VzQiQfZPx6gftO5dMTT
XcWL4QjdXVabg7Ji6wvPuqrqmNLWV1Qy/66nzCYmD8PONeGcK1sZzJA/lGXyMEF8kHnna8f
5o+5lRU9YCqHnS8PwGcBBLcrMOmN3HjIXmqX8S4X+qojbV+jw/Eexz7EJpr/26gLUxP6fYlQ9l
91UPWjXC2AioLzFZiGs6MQJKB7SLykogsdFMhxSRVrQJwvCYOtLOIzAZCMAo69vCIw7X1ax
Hpt6G0iNwvIgUiUlBba6evXJDsFYZf9/RRs5W4RHBgAAsyjaRC091Yxa9C8DDIuN5cHW5gfdE
JxsUP8+/S7c4yaZFR4r34FbOVmIYzUVj/BmRgRF/dBxwUkf8SkfEDHScib4pIcR/LRTWAFpEU
4G/A51W1Z6z4fWAyMAtIAkKvavIWzbpPdJlTo32n5nBfglUbKb8J0CfrA2X3bXYtEWrEOSlqKoR2Nk6QDD8xUvGCNXlx5UUfNL7zYN6dU73mFO7zx1lnebw5MqE0/Jv8Lt8TBNt2tAQpMPyPAAA6/gLo1TbLwIqG0aR9+ZSYj4W2Z46ZIs3QMtG6Np0GvM6CGV04k8vB0nnVUJt6tZ0+XSk/3Xg2XvqFb7LNZ3G3J8M0n/K8zPc1yjHx4PoK/qX7C0mZW1nkoy1b4h+hC0lEbt5M8kOLt2c+ZiCD8uWnSlZLyrfEt91k5n0fPoYUdmgf0W1Zg6ok8tC4Bf0tfC6PMyLWnRh+8sv3qYTs7xCbfkVuz9s
4G/A51W1Z6z4fWAyRoJjLPCvA+h/QlVnqurMhAR7AHNBeiZ5i60/Clmam0pzexfv7DtpnoiuDi
OybdIiy3cePNXUxsdldSyZOsTdVz2ERcPE6415kG4fmWvzMk7NgaiqAtWOpROjxPtKEfnZAM
dcp6rZfSyrgRqHYegxEH3e4SISDawH/k1Vt/Y6d5UatAF/Y79dPaXJgR6ZA20y/cWJeMiy
U+MoS1ZrqxDr8H5077hftqY3E13crQTB7sj+wVRnLosS1mKzEFZ+ZAvikiO4CfYZRxz1HVrwEz
gBWDvO4aoKct7r3AZ9p8iUgI8BrwV1V9+bxtPSf4AgmVXRZPZT0IvCcwEQFGABDJ49ygVEdb
gBWDvO4aoKct7r3AZ9p8iUgI8BrwV1V9+bxtPcZHMOZPigepw6YvsldAdRGcOmi2EpcICgxgS
U4Kb+09SVNbpzkiSl6F0Gi/6Dy4dvcJJiRGMjk5ymwpvkPGAsM9OUTdWM6MQOKBm1R1gaq+
rKodAA530g2DvO6jwHwROQjMd3xGRGaKyJ8c+9wKXA3c18ez7h+KyD5gH0b4cV/cl3mhcsH5uq+U/uf2TbFpnD+GwAbG1Wx3A6y7ezaDzOM5yhWE1mT5z+DlOSoBZ6KY5Z8y0+7gKPYg6ErvXCvBs9kgmVbS3WgdGHI1udpWSpYsfH+NMPHiw5RNfgWnxRQKyVyd7ePUjdIhG4EwZ4IkW1U4KxtqE6e5L4gSqOQ53DyX1Y0Aj9BXW0JoFt5/LUxW3W0hl/Y+5AWb5JYWzXsDMaR8MEPMSIKrT3E0Vk/
rKodAA530g2DvO6jwHwROQjMd3xGRGaKyJ8c+9wKXAnc10e47rMiUgQUOfT95yB12PRF1nJ

A/MaN1dbZzZulNd6/eGcb7F0Hk5dAUL/TdJagquEc247Wsyx3iCYP9kdIhOGeLF1tuCuHGM7
MgTykqsf62bZ3MBdV1TpVnaeqGY7Xesf6AlX9kuP9M6oa3CtU99NwXVW9VlVzHC6xu1W1aT
A6bPohOhVGz/ULN9b00SNIjQljbaEJbqyyt6GtAbKs775aV2gkZS7LTTVZiQ+SvQLO1RvuyiG
GWXkgNr5O9k1Quw9qSsxW4hIBAcINuam8f7CWMy1eLqBQ9DKEx8L4a7x7XQ+wpvAEU0fG
MCbe2k2wPMKEeRAa4xcPXBeLbUBs+iZzOUggFK00W4nLLJ2aSkeX8kZJtfcu2tYE+zYY7sDA
YO9d1wMcOdVMUWWDPfroj6BQmHID7F1ruC2HELYBsembyAQYd7VhQCxeMC47LZoxcRGsL
fRibaz9G6HzHOTc4r1reog1u08gAjdMtQ1Iv2TfBG1n4eBms5V4FduA2PRPzi3QcBzKt5mtxCV
EhKW5qWwpO0Vto5eeEItehug0GHWJd67nIVSVNYWVzBoTS3JMmNlyfJexV0NEPBS9ZLYSr
2IbEJv+mXKD0b+i6OUL7+vjLMtNpVthnTdyQlrqoewtY3I1wNq32N6qRspqm2331YUIDDJGIft
fh9YGs9V4DWv/d9t4ltAoI0Sx5FXLhyhmJEWRlRrNql1eaJhVugq6O/3DfVV4gqAAsZMHnSHn
VuhqM+ZChgi2AbEZmJxboaUODr9rthKXWZ6XRmFFA4drPRz1XbQS4idCco5nr+NhuruVNbsr
uTwjnthhIWbL8X1GzoQRY/1ixO4stgGxGZgJ10HYcL+4KZbmpiICq3Z70I3VUAHHPjJGHxZP
uPvkSD0nGlq5cVqa2VKsgYjxdz/yPjR6MeLPRGwDYjMwQSGQmW9kVLc3m63GJZJjwrh0fBy
rd1einoos6wl7zh5slR/fYdWuSoaFBHJ9ZrLZUqzD1FtBu4dMaRPbgNhcmKm3QkezEZpqcfLz0j
hW18Ku8jPuP7kq7HkRRs4yeqtYmNaOLjYUVbEwO4XwkECz5ViH+AxIyYM9QyMayzYgNhdm9
KVGSOqeF81W4jILs5MJCQpgtScm06uL4GQpTL3twvv6OG/tPUljW6ftvhoMU2+Fqt2WL0bq
DLYBsbkwAQHGl+Kht6DRhKKEbiQ6LJj5U5JYt6eKji43tyItfAECgv3CffXarkqSokOZOz7ObCn
WI3sFSIBfPHBdCNuA2DhH7h2gXX4xmZ6fl0pdczsfHHRjh8quTiOJbOICiIh133lNoL65nXf3ny
Q/L43AAGsHAphCVLJRxaHwBb/vl24bEBvnSJgIaTOg8HmzlbjM1ZMSGRERzCs73ejGKnvb6H
uee4f7zmkS6/ecoLNbWZ5nu68GTe6d0FAOxz40W4lHsQ2IjfPk3gE1xYav38KEBAWQn5fG5p
IaGlrclCC55wUIH+EXfc9f21XJ5OQoMlPtvueDZvISo5HYbus/cA2EKQZERGJFZLOIHHS8juhn
v65ezaTW9Fo/VkQ+cRz/oqN7oY2nyV5h+Pj94Ka4ecZI2ru6WeOO0iatDbBvvfH7CbL2v2JZb
RM7j5+xJ89dJSTCqMRcutqozOynmDUCeRB4S1UzgLccn/viXK9mUst6rf8p8GvH8aeBL3pWrg
1g+PYnLjB8/V0mtYh1E1mp0UxOjmJlQbnrJytdDZ2tfuG+eHxcQQAAHgxJREFUermggsAA4cb
ptgFxmby7jPD3vWsuvK9FMcuA5ANPO94/jdHX3CkcfdCvBXoaVVzU8TYuknen4esve8tsJS4hI
tw8YySFFQ0cqGl07WS7n4e4CcYckYXp7Orm1Z0VXDMpgcQou/Kuy4yaA7HjYPdzZivxGGYZk
CRVrQJwvCb2s1+YiBSIyFYR6TESccAZVe15BK4A7MclbzFhvtFlzw9uiuXT0ggKEF7ZUTH4k5w
6BMe3GE+bFi9d8v7BWk42tnHLzFFmS/EPRIxR6dEP4Mxxs9V4BI8ZEBF5U0SK+1jyL+I0o1V1
JnAn8BsRGQ/0dZf2W5dCRO53GKGC2lo3hm0OVYJCjJyQ/Rugc5sNS4RHxnK1ZMSeXVXJZ
2DzQnZ9Tejc2Pene4VZwIvF1QQNyyEayf39zxnc9Hk3m68Fr5grg4P4TEDoqrXqWp2H8tqoEZ
EUgAcryf7OccJx+th4F1gGnAKGC4iQY7dRgL9zoSq6hOqOlNVZyYkJLjt5xvSTL8Hutr9IlHq5h
kjqW1s44ODpy7+4K5OI6x54gIj9t/C1De38+beGm6clkZwoB2c6TaGj4axV8LuZ/0yJ8Ss/5Q
1wL2O9/cCq8/fQURGiEio4308cBlQqkYVvHeAmwc63saDJGVB2kzY+bTl291eOzmR2GEhvLh9
EJPpBzdBUw1Mu8f9wrzMql2VdHSp7b7yBNM+B6ePwpH3zFbidswyII8C80XkIDDf8RkRmSkif
3LsMwUoEJFCDIPxqKqWOrb9K/CAiBzCmBP5s1fV28D0z0HtPqjYbrYSlwgJCmDF9DTe3FvDyc
bWizt4518hMsnyuR+qyksF5eSOjGFScpTZcvyPKUuNHKGdT194X4thigFR1TpVnaeqGY7Xesf6
AlX9kuP9FlXNUdVcx+ufex1/WFVnq+oEVb1FVb3U6NrmU7JvguBhfnFT3D57NJ3dysqLmUxvr
DZGILl3GO1MLUxRZQP7qhu52R59eIbgMOP/ZO86aB6Eq9SHsZ2dNoMjNMowIsWvQutZs9
W4xPiESOaMjeWFbeV0dzvpktv9nFEbzA/cV89uPU54cCD5eXbfc48x/V7o7vCL6MXe2AbEZv
DMuA86Woye6RbnzjmjOV7fwkdlTjwhdnfDrmcg/TKIn+B5cR6k4VwHawpPkJ+XSnRYsNly/Jf
EyTDqEr+YN+yNbUBsBk/aDEjMhB3Wd2MtyEpmREQwz29zIl7/yHtQX2Y8VVqcVbsqOdfRxV
1z0s2W4v/MuBfqDhktj/0E24DYDB4R40v0xE44sctsNS4RFhzIiukj2VRSQ23jBabUtv8JIuKM
WkcWRlV59pNjTB0ZQ87IGLPl+D+ZzyE0BnY8ZbYSt2EbEBvXyL0dgiNg258uvK+Pc8ccYzL95
R0DhPQ2VBhJlNM/B0Gh3hPnAQqOneZATRN3zRlttpShQUiE0a2wdI3fTKbbBsTGNcKHG5npx
Suhpd5sNS7RM5n+/LbjdPU3mV7wF8OHPfML3hXnAZ7deoyo0CCW5tqT515j1hehq80vohfBNi
A27mD2l41qtLv+ZrYSl/nc3DGU15/j7X19FEfobDdu/IkLjQxjC1Pf3M6Gompump5GRIi1w5At
ReIUIzN9+5OWr2gNtgGxcQdJWZB+uTE30N1lthqXWJCVREpMGH/56MhnN+5dY1QinvUl7w
tzMy9uL6e9q5s77clz7zPnq3C2AvavN1uJy9gGxMY9zP6yUXH04CazlbhEUGAA98xNZ0tZHfu
rzyvzvv1PMGIsjL/WHHFuoqOrm6e3HOWyCXF25rkZ9IxgP/k/s5W4jG1AbNzD5CUQlQrbnjB
bicvcMWs0oUEBPLXl6N9XVhfB8Y8NH3aAtW+bDUVVVJ9t5QuXjTVbytAkIBBmfdkI560uNlu
NS1j7TrDxHQKDYebnoextqD1gthqXGDEshBunpfHargrOtLQbK7f8HkIiLZ95rqo8+eERxsUP45
pJdtl205h2NwSFwzZrj0JsA2LjPmbc9//bu/PwKuqrgePfk5CQQEB20OLAkCAqBQBQJTIIosRFIRS
FZVNpVZARBbRKpW2vi19qxQtVepSfAVUsCiWRbAI1aIoYBCTsENEQFmCQWPQhACBLOf9Yy
40QEK2m8y95Hye5z5Z5G5UJZubOPXcgOXd+v5lzILA6fP6y25GU2y+7R5Cdk8/ChEOQcdi5yuzaXzh
XnfmxpAM/sjUlg/u7RxAQ4N8NsPxajXrOJb3bFvn11YuWQIz3hDVyGitteRuOfed2NOXSpklt
urWsz/z4/eR/Pgs035n89HZ13/DZaFB3NmpqduhmC5jIfckJPhvMXFLIMa7rp/oNJvy81NzgF
E9IsnMOEpewuvOXcR1/fuKpUNHT/DBziMM79zcLt31BY2joHU/2DgLTp9wO5oysQRivC7UQ7

/g4Q5cOpY8dv7sN5tGjGxzgaCcrPI7zbR7XDK7fXP9hMgwn3X+3civKT0eBROpDvFOf2QJRDj
fd0nQXYGbPLvGwsDNJcR8j6f57flo8xwt8Mpl/SsU7z9xUFu7XgFl18W6nY45owW3ZwqvfEvQ
l6O29GUmivnsSJSD3gHiAD2A0NU9cfztrkReL7AojbAMFVdJiJvADcAGZ51v1TVLWWJJScnh5
SUFLKzS9mN7hIVEhJC06ZNCQoqR2nvprHOjYUbXnbuDwn00zLhO5ZS4+QR3g0dyZdr9nJT20
aI+OfE85z135Cdm8e4G/27/Pwlqcej8PZQp7dOx6FuR1Mqbg2ETgE+UtXpIjLF8/MTBTdQ1TV
ADJxNOHuBgnepTVbVxeUNJCUlhVq1ahEREeG3fxy8RVVJT08nJSWFyMhy3iPQ/WF4awhsX
wwxw70TYGXKy4W1z0KjKKKvGcI7y3exYV8617dq4HZkpfbTidPMj9/Pz6Ivp1WjMLfDMedr3
ddpi7D+eYge7Ff3GbkV6W3AmWpi84Di6mLfBaxSVa/PNGVnZ1O/fv0qnzwARIT69et752ys1c
3QOBrW/sU/a/7sWOz0boibwl2xzWlYqzp//2Sf21GVyWuf7ef46Twm9LazD58UEADdH4G0Z
NjzgdvRlIpbCaSxqqYCeL4Wd0fTMODt85Y9LSLbROR5ESmyrraIjBGRRBFJTEtLK2qbUoR+afP
asQgIgBt/6zRe2rbQO/usLHm58Okz0Lg9tPk5IUGBjO4Ryfq9P7D54I/FP9+HZGbn8Ppn39Cv
XWPaNKntdjimKO3vgDotYM00p+Oln6iwBCIiq0VkRyGP20q5n8uBaKBgav4NzpzIdUA9zhv+Kk
hVX1XVWFWNbdiwYRneiSmzq/vDFdfCJ884lWz9xfZ/wtGvIW7K2eGEe7q2oH7NYGZ8uNvl4
Epnfvx+jmXnMrF3a7dDMRcTGOR84DqyDZKXux1NiVVYAlHVm1S1fSGP5cB3nsRwJkEUUjv7r
CHAu6p69hIFVU1VxyngdaBzRb2PyvD000/Trl07OnToQExMDBs3bmT06NHs2rXL7dDKRwR6/
w4yDsLm+W5HUzJ5ufDps9AkGtoMPLs4rHo1JvRuxWd701m3p/AzWV+TcSKH2eu+oXebRrQP
t46DPi96MDRsCx8/7TfDvm4NYb0HnGkofR9wsZQ7nPOGrwokH8GZP/HbimQbNmxgxYoVbNq
0iW3btrF69WqaNWvGnDlzIziIqKcju88ruyj3OZ4toZkHPS7WiKt20h/PgNxP3GSYAF3N2lOeF1
Qnn237vJL6rhlA95+ZO9ZGbn8Hjfq90OxZREQCD0fhLS98DW80fsfZNbV2FNB/4pIqOAg8Bg
ABGJBcaq6mjPzxFAM+DT856/QEQaAgJsAbxSY+KP/9rJrm8zvbGrs6KuqM0fft6uyPWpqak0
aNCA6tWdaZwGDZyrfOLi4pgxWaxsbGEhYUxadIkVqxYQWhoKMuXL6dx48Dh4
8CMDMmTPp3r27V+MvNxHnl2LeQEh8DbqNdzuiop0+4YxBXx4DVw+4YHX1aoH86uareGzRVlb
uSGVgB9/t5Hfo6Ane+Gw/d1zTlKgrbO7Db7T5mWfYd7pTK8vH2ya7cgaiqumq2kdVW3u+HvUs
TzyTPDw/71fVcFXNP+/5vVU12jMkdq+qZlX2e/CWvn37cujQIa666irGjRvHp5+enyvh+PHjdO
3ala1bt9KrVy9mz54NwKRJeJk3j00UdJSEhgyZIljB7to42OIntCyzjniixfLhwX/wJkHoZb/nzB2c
cZt18TztWNa/HXD78iJ893Jzv/+uFuRODxfle5HYopDRHo83un4VTia25HUywriFPAxc4UKkpY
WBhJSUmsW7eONWvMWHToUKZPn37ONsNsHwBwc6IzHd+rUif/85z8ArF69+px5MI4dO
0atWj7YJKjfNHilJ6x5Gn72V7ejuVBGCqyf6dS8anF9/kZsFBgiT+13N6PmJLEw4xIiuvlcWZHtK
Bsu2fMu4uCvtrnN/1DIOIm9wzkKih0DN+m5HVCRLID4gMDCQuLg44Ozt68eeesDwoKO
nt5bWBgILm5zgRbfn4+GzZsIDTUD/5ING7ntIJNmO2UfW8S7XZE51o91am4e/P/Frtpn7aN6
NqyHjM+2M2A9k2oHoH+Y7wwyqyrSVydSrGczYuCvdDseUhQj0fwZmdYePpsKtL7odUZH855bHS
9Tu3bvZs2fP2Z2Z+3bNlCixYl+1Tbt29fXnrppXOe69Nu/A2E1oWVVvwb1oUnoQ1/A9kVw/YQSVd
wVEf50W3uOn8rlz6u+rIQAS27FtlQ2fJ3OpD6tqR3ipyVkDDRqC10fgk3z4VCC29EUyRKIy7Ky
srjvvvuIoqiQ4cO7Nq1i6l6l5bouS+88AKJiYl06NCBqKgoXnnllYoNtrxC6zrjuwfwfjYccSt6Nx5OfB
qicgrAn0+FWJn9a6cS3G9GrJ4qQUNn6dXoEBltyPx08z9b2ddGh6Gff64NCaKaW4KVDrclj5mP
P/1AeJ+tInwQoWGxuriYmJ5yxLTk6mbdu2Lkx6mbdu2LkXkmyr0mOTnwewbIet7GPe5+x3+1s+E1X+AO+
dC9F2leurJ03nc/PynhAYF8v7DPQmu5u7nsccXbeXdzYf514qeduXVpWLHElg8EgbMcAqTukRE
klQ19vzldgZiKldAIAycciKdFTYOVkd2P5/ktnUr/NQGh/Z6mfHhocyB9vbcee77OYs/7rCgiw5Nbv
+YHFSSk82KulJY9LSbs7nAn1j/4EPx1yO55oLWAIxlS/8Wuj1a6dkyI6l7sSQlwvLxkL1Wk5CK2
MNsD5tG9O/fRNm/mcPOw5nFP+ECnDydB6/fXc7kQ1q8q8nAfK1lySRGBn/8NNA+WjvG5oSxLI
MYdPR+D8E6w4lHI/LbyX/+zmfDtZueS4rDy1UibNiiaejWDmfj2ZrJOVX4Jiqnv7eTg0RNMGx
RNSFBgpb++qWD1Ip3/pwfjYd1zbkdzDksgxh2B1WDQq07/9O/eqrEMJzjX27e6AdoPKvbu
6NYOZOSyGA+nH+f3yyq2q807CQd5JPMTE3q3odvXv3i9gyqnDUGh/F3zyZ+eqQR9hCcS4p0Er
6PsU7PvY+cWoDBmHYeHdcFm4V29o7NqyPg/3ac3STYdZkpTitf1ezIz17DGfzP8p30aNWAR26y
O84vaSIw8DmoHQ5LRsNJ32grYAnEuCt2JFxzr9N/Y0sFQIWDneKOg5fCDXqeXX3E3u
pktkPZ5ctqPC+4ZknMhh3IJN1K8ZzN+GxRAYYD1tLnkhl8Gdc5wh34X3QI77bbbgtgfiAI0eOM
GzYMK688bbbqioqIYMGAAAX331Van2MWDAAH766acKirACiAAci7iT2JG94L2JsH99xbyOqjNUlroN7p
rr3KjlZYEBwot3X0PDWtW5/40EvvrumNdfA+DE6VzGvJlIasZJXr7nWp+6E95UsOZdYNArc0
AzWPqA65PqlkBcpqoMGjSIuLg49u3bx65du2bRrfffddqfazcuVK6Rx+Z6KsgoMgiFvQr2Wz
ier75O9u//8fFj1a9i5FG6aClf18+7+C2hUK4R/jOpCcGAAI+Zu5NBR73ZhPnk6j1FvJJKw/yh/
HRLDtc3renX/xg9E3wV9n4bk9+DfU1yt6mC1sApaNQWObPfuPpu0h5umF/npNJm5PfuPptEQ//pRa5es2YNQUFBjB37
34r0MTExqCqTJ09m1apqTJ09m1apViAhPPvkkQ4cOJTU1laFDh5KZmUlubi7z5s2jZ8+eREREkJiYSFZWFvAgN0ZuFdPeEkJiYSFZWFv
3796dHjx7Ex8cTHh7O8uuBzxX6w8+fTTH8TCQ7KkZaLaXLZnvYZ1S2nW0iruVlK+ubFBimzBmF1oF7/glz+8Jr
t8CwtyDCC6Xp83Jg2UNQqjJuE6D7D7pPLvsxjN69dg/qjODHllAyPmbmTBA10Jr1P+emXZOXm
Mnp/Axm/SeW5IDLd29N1y8qaCXT8BjqXChpcgoJozlxhQ+Vfg2Rmly3bs2EGnTp0uWL506VK2bJlC1q1bWb16NZMnT+Q2UUNW33nnD
2bNnC1q1bWb16NZMnTyY1NZW33nqLfv36nV0XOXG6pmqD4Nf/PX8B8VmQmvNlmZ/ODuBE6KdjI1HJmgTUx
MppEt6jDA9hWx+D3P1kP7hX/ZLkMCXAV37BZpsyiUiBodtTvf5XK8nVBy6+GHmnZ0cKVJKLJGhOfQa4nO6YSmb1ROqjHBbB+
2NKE7iBe+gi7T3JpOXFvYPZS+7j7+5IOd/qlCaZ1ZOhGEMVDq4L2JsQhxVT4Ffyw0XFWaDd3
Cv6Q+uPPPNknkDDHZz7MdUS/9sT4ncJmS3LGxVXJ/YHKHGi7/w1XjuI0s4vkxBzGl6oT3ydzA
PZkhH0pqQX+pFs7Je+vi8nqm0lUNfl2Q6b3qSMxyN7Yk1ocgG+9LbrbK3eJB69u6BAmYtLEu
Hyhl1WgpdTcIObXuZrnZgVFdUjrJDK2cAjPmdH5m1QX4bL9p7zZ3GzF8EbOq59iWE49gpIqX
3p0GBjP7dDFOVgjSk1Y89pVdS4JxRyvbTZ4fJ/q7tUdHPa82hBa0oZzjTo/A9aLHxKSOiP5Z
IfxOTJVjQl05oCbsBq7bP+YrXv9YDGfHWYCLcYkX/f5HrPumM8iS6FfgqPqDwX8/n9QSG1nA6
oB8ZP8mGIF58FHU3ENo4HPPKbEQTrbvWDRHS+JrM2jKDkb4KuDvzANMDfUd7Hqh3X5kzn7
fVZ8gG+6wq4Ub4gP8BO9YW+EM6Js/UMjyGjEz3FbCiDWOwZvFKZr0Lhbs0/M0eUnY9Iw+7+b
l64Fg2L8r6xUXJVgbCwO1z0k+1Abw0YxE5ZmNAB4PAhoDAW6s8aGI0nFexsqDhsV1r9D0gS9

N0kNq/f35n0rNP8/MX1xO/9oVz7O5KRzS/mfkH8vnRmDO7I7deEeylS47du/hN0fhA+/zu8N
RSyK/8+JDsDKegiZwqVbf369QwfPpzAwEAaN27MDTfcQEJCAtdddx0jR44kJyeH22+/vdAEEh
kZeXZ5p06d2L9/P1lZWcTHxzN48OCz2506darS3k+JnUkiC++GJaOc4axej0NQKT/BH9wI/5o
EaV86cyyx91dIuBfTqUVdlk/ozoNvJnHv3I1M6d+GB3q2PFtZuaQ+2HmEJ5Zs43RuPjOHxnBbj
CUPAwQEwIBnoVEbp6rDnJTh2AJoUHk3k7pyBiIig0Vkp4jke7oQFrXdLSKyW0T2isiUAssjRWSj
iOwRkXdEJLhyIve+du3akZSUdMHyomqU9erVi7Vr1xIeHs6IESOYP//CXuNnuhvCf8u/5+fnU6
dOHbZs2XL2kZzs5bkGb6lRD0YsgDYsg47DYd0Mekkz7FxWsrHe7AxY8St4rR+cOgb3LHYleZZzRsmE
Y747vzi3tmzBt5ZcMfmUD6/akFfnvW9DB9BM8sXgbD76ZRLO6NVgxsYclD3Oh2JHO78vx7+H
vXeH9x+BY6eZQy8qtIawdwB3A2qI2EJFA4GWgPxAFDBeRM03CnwGeV9XWwI/AqIoNt+L07
t2bU6dOne0yCJCQkEDdunV55513yMvLI1y0tjbVr19K5c2cOHDhAo0aNeOCBBxg1ahbSbNm0q0
evUrl2byMhIFi1aBDgJauvWrRXynwiKMS52uS+FRBSGxbxbdB7N7O73Vv93sTIyfkXMSkv/lXB/
/XDtIeh26joPxG6H1Te69B4+w6tV4+e5rmTYomsM/nWTE3C+4Y1Y8ixIPkZyaebazYX6+8l1m
Nh8lf8fINxK4YcYaFm9K4cEbWrLkoetp2TDM5XdifFZkT6c46bW/gKQ34IUYp21C8gqncGkFc
WUIS1WTgeJO5TsDe1X1a8+2C4HbRCQZ6A3c7dluHjAVmFVR8VYkEeHdd9/lkUceXYf06YSEh
BAREcHMmTPJysqiY8eOiAjPPvssSTZo0Yd68efzlL38hKCiIsLCwsQs9AirJgwQIeeughnnnrqKXJych
g2bBgdO03aswHfnBZE94cG1sGme84vx8Z+cR3AYSADknoI8z1BcaD1oPwhiR8EVFw7tuUlEuLtL
c+7sFM7ipBT+vmYfkxdvAyC4WgANw6rz/bFscvKcM5MGYdWZeGMGMrhndpbl0FTcnUagIDn3cuF
lkzzfkg9cX/OevqRsLwt71++bqr5dxF5BPgcVVNLGTdXcAtZ3qki8gIoAtOsvhcVVt5ljcDVqlq+y
JeYwwwBqB58+adDhw4cM56K+d+IZ8+Jlnfw741cDjJSSDVgqFaKDTvChE9nEuC/UBevvLND1n
s/DaTXd9mknbsFE0uC+HyOqE0r1eDbi3ru14e3vi5nGxI3QqHPnfKn9w+yzmbL4OiyrlX2BmIiKw
GmhSy6nequrwkuyhkmV5keaFU9VXgVXD6gZTgdY0vC2sEHYc6Dz8WGCC0alSLVo1q2byGqRh
BIc6Nh827VNhLVFgCUdXyDj6nAM0K/NwU+Bb4AagjItVUNbfAcmOMMZXIl8+RE4DWniuugo
FhwHvqjLmtAc60j7sPKMkZTZGqUlfG4tixMMaUlFuX8Q4SkRSgG/C+iHzgWX6FiKwE8JxdTAA
+AJKBf6rqTs8ungB+JSJ7gfrA3LLGEhISQnp6uv3hxkeenphYKpz8/TsDe1X1a8+2C4HbRCQZ6A3c7dluHjAVmFVR8VYkEeHdd9/lkUceYfr06YSEh
JCd7X5lS18QEhJC06ZNCQryj8loY0zfq/RJdH8RFBREZGSk22EYU4z7fGx8eU5EGOMMMT7MEogxx
pgysQRijDGmTKrUJLqIUAIpAEHit2wcA1w7kGpyuwYY2DGo6u8PQSSAmIyC0isltE9IFqfl/LjfpPC
C7VjcICJ1RGSxiHzp+f/Qze2YKpuIPOr5PdghIm+LSEjAnThIm+LSJUuXW0JpBgiEgi8DHDhIhLlLblSVKh
d4TFXbAl2B8VXs/Rc0Cae1QFX1N+DfqtoG6EgVOoA+Xf2NcCSD8AAwQWtv0X4kOfuaEKT6iaqqhmsD+p6nqB+QcryqCntTrA4fv/jOyfknBqQYX7mBYxwfjdGgYW6ri8mvqYQbPRqgqAK7OGvnRaNqGpBIRWFFNIlUJBlvPHcTYNaW+3AxPLmG/zQNIxXrdy2MMG0cU2hnmf2HSmzmdbC/3Fo
AdFoQ6a1Wrkx8ptnBfspsfuPoPwWqBwzBNRlVPnWwrtJR4LSPHZB5u57sxkGWNWwHVqu0cn3wdsmfQ8efOqPNSw+vSkYz4nvdHB3AQ
Haq6l6ZXNpSoW4Ogp+dMjWJC+p3ndQKBj0Skg6pu86zLVNXOIvILLnKrBA3F6dzyvqutFpDlOJYW2pT8SxhTN
EogxRftfnFpo2TiNhArzsIgM8nzfDGgNpKvqHBEZDIwFYgp5XgLwmqdQ5TJV3VLINgUN8ZzhV
AMux2ludiaBvF3g6/Oe728CopySTQDUFpFaxbyGMaViQ1jGFK0eEAbUAi5oXSoicTh/qLupakec
2kghnnU1cCo349nHOVR1LU6DpsPAm56zh0KJSCTwONBHVTsA758XjxbyfYAnrhjPI9zTEMwY
r7EEYkzRXgX+B1gAPFPFI+suAH1X1hIi0wWn5n8e8Yznuf9Hph9/hNFpAVOj5HZONWOL1YavTZ
O/40MEWmM0165oKEFvm7wfP8hMKHA6xV2FmRMudgQljGJF8JwR5qW555/7quaJyJ0icr+q/K985QBZQ5BmIqm4Vkc3ATpwAqw+K
7N/AWBHZBuwGPvc89wbgOqC7quaJyJ0icr+qvl7guXE4k985QBZQ5BmIqm4Vkc3ATpwquJ+d
t0l1EdmI84FwuGfZw8DLntiqAWtxhtOM8RqrxmuMMaZMbAjLGGNMmVgCMcYYUyaWQIwxxpSJJRBjjDFlYgnEGGNMmVgCMcYYUyaWQIwx
xpSJJRBjjDFlYgnEGGNMmVgCMcYYUyaWQIwxxpTJ/wOtI45gaidN7gAAAABJRU5ErkJggg==
\n",
       "text/plain": [
        "<Figure size 432x288 with 1 Axes>"
       ]
      },
      "metadata": {

"needs_background": "light"
},
"output_type": "display_data"
}
],
"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'])"
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"id": "R5IeAY03L9ja"
},
"source": [
"### Subplots "
]
},
{
"cell_type": "markdown",
"metadata": {
"colab_type": "text",
"id": "CfUzwJg0L9ja"
},
"source": [
"You can plot different things in the same figure using the subplot function. Here is an example:"
]
},
{
"cell_type": "code",
"execution_count": 90,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/",
"height": 281
},
"colab_type": "code",
"id": "dM23yGH9L9ja",
"outputId": "14dfa5ea-f453-4da5-a2ee-fea0de8f72d9"
},
"outputs": [
{
"data": {
"image/png":
"iVBORw0KGgoAAAANSUhEUgAAAXIAAAEICAYAAAABCnX+uAAAAABHNCSVQICAgIfAhkiAAAA
AlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9u

My4xLjMsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy+AADFEAAAgAElEQVR4nO3dd1zVZf/H8dfF
YW8FFAUBBwqKynJmNmxoWpppjjQtuy2btve4W3e71OwuV5qZZZplZVlqZlZYLBBRFBcGBqIA
IyF7X7w/w/llpOQ58z/g8Hw8fDznAOe9zhLfX9zrf6/oqrTVCCCGsl4PRAYQQQlwYKXIhhLBy
UuRCCGHlpMiFEMLKSZELIYSVkyIXQggrJ0Uu7JZS6ial1PdG5xDiQik5j1zYOqVUP+A1oAtQC6
QBU7XWWwwwNJoSZOBodQIjGpJTyBlYCU4ClgDNwMVBpZC4hzEmmVoSt6wigtV6sta7VWpd
rrb/XWm9TSk1USq0/+YVKKa2UkMpla6UOq6UmqmUUqd8/lalVFrD51YrpUKNeEJC/JkUubB
1e4BapdQCpdQgpVSzf/j6IaAIPoDtwI3A1gFJqGPAEMBwIANYBixzs2cjmEJpMiFzWso34la62Ag
CmgNvHOGLz9/yyt/LAM+Gv4c05RSho34la62Ag
CmgNvHOGLz9yyt/LAM+Gv4cC05RShUqpQqAAUEBQI8UW4qxJkQu7orXeBcynvtDPxUHgdq2
17yl/3LTWv5s9pBDnSIpc2DSlVIRS6kGlVHDDx22AGcXKHMDXoMudWLmGBkGOD6Tpg1YbaA
UuTC1p0AegGblFKl1Bd4KvDgudyJ1no58CqwRClV3HAfg8yUcMKpy
BCWDkpciGEsHJS5EIIYeUM2TTL399fh4WFGfHQQghhtRITE/O11gh8LCyMhIcGIhx
ZCCKullNp/utvNMrWilJqSqWa4/6EEKRBsBKX9XfavNrWu3paD//u6hUFHGBAvY8HAlAtHVoyfOdbz3fU0ZDPOz3Xm1FIlcM5oanxTzeQ36HrFVjGsV0nvxwC2vGUygi5fu4Ae+s6VKMH6DAk7FhUUtAGz8ebfpHTG7THpLK93wZRI1BXrgYGpb0bQuKKgRR39Oj9ehpd3HYWtUe7ns83Y/Fsg9sMsVFtUOQSNbqNLIBedn3cR8KjPpbzpTnyAgwPd5hYRxvmHkd4zCxm2Pnf+gdgn3rP4gN1gsTVZXdmd9SMjRvQw7MWqHoQQMoyFMvcG71ZmAbW+mQ4W7jHLyMgUbNOHpAtOG2KTOaUmqGJUoq9Y7uB0qlGE8QUW2nfyY4P87XcNJCZ4VpN6vbL7b+Yqo54ddBskHZHybWOm+t3F6vM2JWQQ12tLyhW8oFQ05D1Ru0HoR+yoUR7Qr3PZcrnkx6RdlxN4ByU5vRjn+4Fd7AgNmSvIKNn6o91kDlHa8ZFfoa6dQjzJx6kB/hBmj+PJB3hL3Y51yQmk8Zn/0pSUnmKiUuLDPphywEQ
rJOFhyP+QKQvzbwNeyqaJJZf+VoD0P1nrHpUlYqqwh6JhAv+4b8r+YmBvLmFYV7rF+0oaNvn0UwkeBuGbT0x4oLxOdufwVH1PRU/kjKT4S0uDvP9ztIyXK8bZ3rl7DDkWsNgDADpQ42qRTzNBIiA0DhEuwO+5O/fl8qCRHnp1Ac3IHXH2bjiBKYOhvPyEaPgrRkKNKcAHNHMrI7MWZ2rjkhf/dDpRAFFjMGnVMpOL5kpG7WBi9a2iU5QFLK8h3qiyPi3Fh5k4qIFEzUrBVAzOZo+Jeb1vUMnzdbF8GVWJ6eLrpBuZ0xAVqw1J0R6xbOnqWtcGvIzf6yH9SbAqUcbHLsmtIPY9yH/fLxH/NklfR1WRwpV6qoPsSF5ZN0PrUq3ngaN1bMwx9Oz85e8Jk6cAv5X/ygD3fa2uIFU4Z0TQTCWwDyQjpdvKO/w90yJjfhIrlqIiW4WJo5EFnNkNkBNpmBsRjaXjRYMa/WBBYlIHTyWCPgTdtLh6tzEx3rhdjvj5QVSuAavNuQ3EpXtXsYW7xFmQw4+KbVT7mOqfPjMG+fXGm3Zeg1IWaOHgjjgfNPW8OEVc2aA4RBtGsD4qtzZfWILwrXqIOVBbQ3eNPfCGRkQp6xQq0GGrmURi48O3Y9r9Bou1+YELJbcBUgZnUyD2E7BHjGC+Qw9bd4fH8knfyoGt1mPA5ssXhvcg93ngL/XtO7WzgJ6zf+0jHeYeCeWtr1AjPwxTyLjtqBANCHEJQ7f/oCMaGkd6xItBHkL+/bqr2SDBF89nEXsb5wIGhIuGd8xVp/kFQF89qF8PgSBIjH3xnaAzz+bmdfE+VD3R3gAkyPbJ1uRvQ3CsiH4W3e2ivdZ9LfjRMANLU7I8hkP/EH2iAcxuhTdHuzPc8nmVL9dD/Hrn8Sr4TXZc5w45yxqiOI8OE1dZWbO8ob5bmEUcQ57LpvK1UodqUKmHO+bA3wW5m7Hx69f65C7yaV/s3RrkbpWZbL6Q9GVdGy8I4zV7Ll9+lYcHajE7g6dfzJ3lVKGyU2kIB0DbdoZpB8r8lxA==

VdOA4Ty1P5aIOfjw+KMLoOBZDKcWLw6KIbuPLA0tT2HWk2OhI/2MXRb7g930s25rNfQPCu
bKz9V6vr7FcHB7Ak4M788POo0z/yfIOG0XTyS2u4I6PE2np48K7Y2LlAiV/4upk4oPxcXi6OPK
vjxI4XlpldCTADop8U+YxXvgmjSsiW3LfgHCj41isWy8K4/qYIKatSefn3blGxxEGqK6t485FWyk
ur2HW+Hi5oPcZtPR25f3xcRwtqmSqhRzF2nSR5xZXcNcnSYT6ufPWqO442OhKNHNQSvHy9V
3p1NKL+5Yky9a3duiV73aRsP84r47oRmQrb6PjWLTYkGY8c21nftmTZxFHsTZb5CdPnSqtrOH
9cXF428kihgvh5ly/NLlOa6YsSqSi2jLPmRXm9822w8xtuKThdd1tf5WzOdzUK4ThsZZxFGuzRf
7697vZvK+A/wzvKm9unoMwfw/evjGa1EPFPPfVDqPjiCawN6+ERz5PISbElyeuiTQ6jtVQSvHS
sPqj2KmfJpN93LijWJss8u93HOGDXzIZ1zuEYTFBRsexOld0bsmdl7ZnyZaDLE/KNjqOaETlVbV
M+TgRFycT790Ui7OjTVZCozl5FFtbq7nrkySqauoMyWFz/2oHC8p48LMUugX78PSQzkbHsVoP
XNmRnm2b8+TyVDJyS4yOIxrJs1+lkp5bwjujomnl42Z0HKsU5u/BayO6kXKwkFdX7TIkg00Ve
VVNHXcvTgJg5thYXBztd+XmhXI0OTBjTAxuTibuWrTVYveYEOfvi63ZLE3I5u7LOtC/o/Ve1tA
SDOraiol9w5i7PsuQPcxtqshfW7WLlIOFvD6iG22auxsdx+q19Hbl7VHR7Mk9wbNfpRodR5hRR
u4JnlyeSs+2zeW0XDN5/JoIugb58NBnKU1+1pfNFPmPO48yZ30WE/qEMjCqldFxbEb/jgHcd
WkHliZksyL5kNFxhBlUVNdy16Ik3JxNTB8dI4t+zMTF0cTMsbFoDfcsTqK6tunmy23iXzCnsJyH
Pk+hS2tvHpd33c1u6hXhxIc248nlqezLLzU6jrhAz6/cye6jJ3jrxu4E+rgaHcemhPi588oN3Ug+
WMib3zfdfv9WX+S1dZqpS5Kprqnj3bGxdr2jYWNxNDkwbUwMJgfFPYuNe2deXLjvth/mk00H
uL1/Oy7t1MLoODZpcLdWjOkZwvu/7OXXPXlN8phWX+Qzfkpn874CXhgWRVt/D6Pj2KwgXz
deG9GN7YeKDHtnXlyY7ONlPLpsG92DfXjwqk5Gx7FpzwzpTMeWnjywNKVJrixk1UW+OauA6
WvSGR4TxPDYYKPj2LyruwQyoU8oc9dnsVb2Y7EqNbV1TF2STJ2GGWPkfPHG5uZsYsaYWE5
UVPPA0sbfj8Vq/zULy6qYuiSJkObuPD8syug4duPxayKJCPTioaUp5J6oMDqOOEvT16STsP84L
10fRYifnNHVFDoFevHMtZ1Zl57PnPWZjfpYVlnkWmseW7advJJKZoyJxdPF0ehIdsPVycSMMT
GUVtXw4NIUi9j5Tfy9jZnHeHdtBiPjghkaLSudm9LYniEM7BLI66t3sz27qNEexyqLfPHmg6zacY
RHro6ga7CP0XHsTnhLL54e0jQjDXFhCsuquP/TZEL9PHjuui5Gx7E7SileuaEr/p4u3LukfhO/xm
CWIldKDVRK7VZKZSilHjPHfZ5J+tETPL9yBxeH+zOpX9vGfCjxN06ONF5btZtt2YVGxxGnobX
m0WXbyC+pZProGDzkyNUQvu7OvD0qmn3HSnm2kTaiu+AiV0qZgJnAIKAzMEYp1SibnFRU13L
P4iQ8nB1580bZX9xIJ0caAV4u3Ls4Sa73aYE+2XyA1TuO8vDVneTI1WC92/lx92Ud+Dwxm9
WNsITfHCPynkCG1jpTa10FLAGGmuF+/+K1VbvZdeQEb4zsTgsvWchgtJMjjf0FZbLlrYVJP3qC
F1bu5OJwf27r187oOAK4b0A4jw6M4OJwf7PftzmKPAg4eMrH2Q23/YFSarJSKkEplZCXd34n
yV/TNZCHr+7EZRGykMFSnDrS+Colx+g4gv8/cnV3duTNkXLkaikcTQ5MubQ97s7mn+IyR5Gf7
qfkL6cyaK1naa3jtdbxAQHnt9NafFhz7rqsw3l9r2g89w4IJybElye/2C6XiLMAr67a1XDk2o0W
3nLkag/MUeTZQJtTPg4GZGhmR5xMDkwfHQPAfUuSqGnCzYLEH63dlcuHv+1jYt8wLo9oaXQc
0UTMUeRbgHClVFullDMwGvjKDPcrrEib5u68eH0UWw8UMm2N8RejtUe5xRU89FkKEYFePDYo
wug4ogldcJFrrWuAu4HVQBqwVGst73zZoaHRQYyIC+bdtRls2HvM6Dh2pa5O88DSFEqrapgxJ
kY2j7MzZjmPXGv9rda6o9a6vdb6JXPcp7BO/76uC2F+Htz/aTLHS6uMjmM3Pvg1k/UZ+Tx7b
RfC5WLjdscqV3YKy+Xh4siMMTEcK63kkWXb0FqW8De2pAPHefP73Qzu2orRPdr88zcImyNFL
swuKsiHRwdG8MPOo3y0Yb/RcWxacUU19y5JoqW3Ky8P74pScqqhPZIiF41iUr+2XB7Rgpe+SS
P1UONtFmTPtNY8/sV2cgormD4mGh83J6MjCYNIkYtGoZTijZHdaebhxD2yhL9RfLL5AN9sO8y
DV3UkLrS50XGEgaTIRaNp7uHMtNEx7D9WytNfpsp8uRmlHS7m+a/rl+Df0b+90XGEwaTIRaP
q3c6PeweEszzpEJ8lZBsdxyaUVdVw9ydb8XZz4u1R0bIEX0iRi8Z3z+XhXNTBj6dXpJJ2uNjoO
FZNa82Ty1PJzC9l2qho/D1djI4kLIAUuWh0JgfFO6Ni8HFz4q5FW2W+/AIs3nyQ5UmHmDqgI
307mH8XPWGdpMhFkwjwcmH6mBj2HSvl8S+2y3z5eUg9VMRzX+2gf8cA7rlcNo8T/0+KXDSZ
3u38ePCqTnydksOC3/cZHceqFJVXM2VRIn6ezrwj8+LiT6TIRZOackl7rohswYvfpLFlX4HRcaxC
XZ3mwaXJHC6s4N2xsTT3cDY6krAwUuSiSTk4KN68MZrgZm7cuWgrucUVRkeyeDN+yuDHtFy
eGhxJXGgzo+MICyRFLpqcj5sT74+Po6Sihrs+2Uq17F9+RmvSjvL2j3sYHhvEhL5hRscRFkqKXBgi
ItCbV27oypZ9x3lh5U6j41ikrPxSpn6aTFSQNy9fL/uoiDMz/8XjhDhLQ6ODSD1UxOx1WXQK9
OKmXqFGR7IYxRXVTP4oAUcHxfvj4mR/cfG3ZEQuDPXYoEgu7RTAsyt2yMUoGtTU1nHPJ0lk5
Zfy3k1xBDdzNzqSsHBS5MJQJgfF9DExhPq5c+eiRA4ck4s3v/RtGr/syePFYVH0ae9ndBxhBaTI
heG8XZ2YM6EHdRpuXbCForJqoyMZZtGm/Xz42z4m9WvL6J4hRscRVkKKXFiEtv4evD8ujgPH
ypi8MIHKmlqjIzW5tbtzeWbFDi7rFMAT10QaHUdYESlyYTH6tPfj9ZHd2JRVwEOfbaOuzn6W
8accLOTOj7cSEejF9DExmGTlpjgHctaKsChDo4PIKazg1VW7aO3jyuN2MDLdl1/KrfO34O/lzIe
39MDLVa70I86NFLmwOHdc0o6cwnI++DUTX3dnplxquxdOyD1Rwc3zNqOBBbf0pIWXq9GRhB
WSIhcWRynFc9d1obiimldX7cLDxcTNfcKMjmV2x0oquWn2JvJLKll0Wy/aBXgaHUlYKSlyYZF
MDvXX/CyrquWZFTtwd3ZkRFyw0bHMprCsinFzN3OgoIz5t/QkJkT2UBHnT97sFBbLyeTAjDEx
9OvgzyOfp7A8yTYuFVdcUc3N8zazN7eE2TfHy7ni4oJJkQuL5upkYtbNcfRu58cDS1P4ZNMBoy
NdkGMllYydvZG0w8X8d1ws/TsGGB1J2AApcmHx3J0dmTexB5d2DOCJ5duZuz7L6Ejn5XBROT
d+sIH0oyV8MD6OAZEtjY4kbIQUubAKrk4mPhgfz6CoQF5YuZM3Vu+2qvPMs/JLGfHfDeQWV
7JwUi8uj5ASF+YjRS6shrNj/Zz5qPg2vLs2g3uXJFFRbfkrQDfsPcbw936jvLqWxZN707Ntc6Mj

CRsjZ60Iq+JocuCVG7rSNsCDV77bxaHCcmbfHI+/p4vR0U5r0ab9PLtiB6F+7syd0IMwfw+jIwk
bJCNyYXWUUtxxSXv+e1MsaYeLGTJ9PZsyLWsL3IrqWp7+MpUnl6fSL9yf5XddJCUuGo0Uub
Bag7q24vM7+uLmbGLM7I3MWJNOrQXMm+8+coJhM39j4cb9TO7fjrkTeuAty+5FI5IiF1YtKsi
Hr+/px7XdW/PmD3u4ac5GMvNKDMlSV6eZ/1sW1767nvySSj6c2IMnromUDbBEo1NaN/0IJj
4+XickJDT54wrbpbXms8RsXli5k8rqOqZc2p4pl7ZvskukJR8s5NmvdpBysJDLOgXw2ojuBHhZ5
ry9sF5KqUStdfyfb5c3O4VNUEpxY3wbLu0UwIsr05i2Jp0VyYeYekVHhnRrhaOpcQ4+DxeV89b
3e/gsMZsALxfeHNmd4bFBcqFk0aRkRC5s0rr0PF5cmcbuoydo6+/BnZe257ro1rg4mmeEvutIMb
N+zeSr5ByUglsvass9A8LxdJGxkWg8ZxqRX1CRK6VGAs8BkUBPrfVZtbMUuWgKdXWa73ceZ
fqadHYeLsbb1ZHB3VozPDaIuJBmOJzj3PXR4gpWpR7h2+2H2ZRVgJuTiVE92jCpX1vaNJcLJI
vG11hFHgnUAR8AD0mRC0uktWZdej7Lkw6xKvUI5dW1+Lg5ERPiS1xIMyJbeePv5YKfhzPerk6
UVddQWllDYVk1u46cYOfhYlIPFbEtuwiA8BaeDI1uzU29Qmnm4WzwsxP2pFHmyLXWaQ13fiF
3I0SjUkrRv2MA/TsG8OKwGn5MO8qGvcdI3H+cn3fn/eP3e7s60rm1Nw9c2ZFBUYGEt/RqgtR
CnL0mm9BTSk0GJgOEhMjVwYUxPFwcGRodxNDoIACKyqrJOlZKQWkl+SVVnKiowd3ZhIeLI16
ujnQI8CS4mZsMVoRF+8ciV0r9CASe5lNPaq1XnO0Daa1nAbOgfmrlrBMK0Yh83J2Idvc1OoYQ
F+Qfi1xrfUVTBBFCCHF+ZGWnEEJYuQs9a+V6YAYQABQCyVrrq8/i+/KA/ef5sP5A/nl+r62Q1
0BeA3t//mCfr0Go1vovl5UyZEHQhVBKJZzu9Bt7Iq+BvAb2/vxBXoNTydSKEEJYOSlyIYSwctZ
Y5LOMDmAB5DWQ18Denz/Ia/A/VjdHLkRTUUrtAO7SWv9sdBYh/o41jsiFOC2l1FilVIJSqkQp
dVgp9Z1Sqt/53p/WuouUuLAGUuTCJiilHgDeAV4GWgIhwHvAUCNzCdEUrKrIlVIDlVK7lVIZSq
nHjM7TlJRSbZRSa5VSaUqpHUqp+4zOZBSllEkplaSUWtnwsQ/wPPXTIF9orUu11tVa66+11g8
rpVyUUu8opXIa/ryjlHJp+F5/pdRKpVShUqpAKbVOKeXQ8Ll9SqkrGv7+nFJqqVLqI6XUiYZ/g/
hTMrVWSi1TSuUppbKUUvc24vP3VUp9rpTa1fDz0KexHstSKaXub/g3SFVKLVZKuRqdyKgFZ
RKKRMwExgEdAbGKKU6G5uqSdUAD2qtI4HewF129vxPdR+QdsrHfQBXYPkZvv5J6l+zaKA70B
N4quFzDwLZ1C9qawk8AZzpjaPrgCWAL/AV8C5AQ/F/DaQAQcAAYKpS6h8Xx52nacAqrXUE
9c8n7R++3qYopYKAe4F4rXUUYAJGG5vKWFZT5NT/8mVorTO11lXU/0LZzWGz1vqw1nprw99
PUP/LG2RsqqanlAoGBgNzTrnZD8jXWtec4dtuAp7XWXudbSifsVctdZ6nT7zG
QDrtdbfaq1rgYXUlyhADyBAa/281rpKa50JzKYRykUp5Q30B+YCNDxeobkfxwo4Am5KKKUfAHc
gxOI+hrKnIg4CDp3ycjR0WGYBSKgyIATYZm8QQ7wCPUH9Bk5OOAf4Nv9Sn05o/bgmxv+E2g
NeBDOB7pVTmP0zZHTnl72WAa8NjhgKtG6ZnCpVShdSP7Fue7ZM6B+2APODDhumlOUopj0Z
4HIultT4EvAEcAA4DRVrr741NZSxrKvLTbQhtd+dOKqU8gWXAVK11sdF5mpJSagiQQq7VO/NO
nNgAVwLAzfGsO9WV7UkjDbWitT2itH9RatwOuBR5QSg04xSytte8pf7y01tec4/2cDUcg
Fviv1joGKAXs7f2iZtQfjbel/j9kD6XUOGNXzXUOGNTGcuaijwbaHPKx8HY2eGUUsqJ+hJfpLX+wug8BrgI
uE4ptY/6qbXLlVIfa62LgGeAmUqpYUopd6WUk1JqkKvKLQP5p5d/wtR9D/X8OSqk
Oqv7KEcVAbcOfc7EZKFZKPaqUcmt4MzZKKdXDLM/6j7KBbK31yaOxz6kvdntyBfX/ceZprauB
L4C+BmcylDUV+RYgXCnVVinlTP3841cZ2oyDUUzF0jTWr9ldB4jaK0f11oHa63DqP/3/0lrPa7
hc28BD1D/JmYe9aPku4EvgReBBGAbsB3Y2nAbQDjwI1BC/cj+vXM9d7xhzvxa6t9MzaJ+R745
gM95PtW/e6wjwEGlVKeGmwYAO839OBbuANC74T9sRf1rYFdv+P6ZVa3sVEpdQ/0cqmQ/0cqQmYp7
V+yeBITaZhYcs66ovo5PzwE1rrb41LZXMA9c8AABw3SURBVByl1KXUX/B7iNFZmppSKpr6/yi
cgUzgFq31cWNTS2l1L+BUdSfzZUE3Ka1rjQ2lXGsqsiFEEL8lTVNrQghhDgNKXIhhLByUuRCCGHl
GHlzrSAolH5+/vrsLAwIx5aCCGsVmJiYv7prtpliXFEEL8lTVNrQghhDgNKXIhhLByUuRCC
Xai9aba2plPjDQTPclhBDiHJhlRK61/rVh/49GlXa4mLwTlfi6O+Hj5kRzD2e8XJ0a+2GFsBrHS6
vIK6mktLKG8qr6Bar+Xi4EeLrg4+aEg8PpdroQ1q7J5siVUpOByQAhISHndR8fb9zPR0k0H/nBbS
HN3ugX7EN3Gl8siWtA+wPOCswphDbTWbD9UxI87j7LtUBFph4s5WnzmNTEujg50D/YlLqwZP
cKa0be9P65OpiZMLBqL2RYENYzIV57NHHl8fLw+nznynMJycgrLKSyrpqi8miPFFWzPLmJbdiE5
RRUAdGntzbXdW3N9TBAtve16r3lhozJyT7Bo0wVIUnka28iWzlRSsfNzxdH
HF3NlGnIb+kkvySSg4UlLF1/3F25BRTU6fxcnHkmq6tuD42iJ5hzWW0bgWUUola6/i/3G5NRf5
3cgrL+S71CF+n5JB8sBBnkwM3xAVxe//2hPnb1S6fwgZprdmUVCsDsXzNZsysXZ0cH+ocHMDA
qkAERLWjm4XzW91VeVcuWfQWsSM7hu9TDlFXV0rGlJ/cOCOeaqFZS6BbM5ov8VPvyS5mz
PpOlCdnU1NYxLDqIxwZF0EJG6MIKpR0u5vmvd7Ih8xjXPNZy5uU8o43uH4ufpcsH3XVZVw3fbj
/DfX/aSkVtCeAtPHriyIwOjAqnfj0pYkkYtcqXUYuBSwB84CjyrtZ57up+Pj4/XCRs/qBCjyrtZ57pq9v7CI/Kbe4gjnrs5j/+z
6cTQ7cf2VHJvQJxdEk66CE5TteWsVbP+xh0ab9eLs5MXviad9N
wSLukYwIvDomjT3N3sjyXOX6OPyM9FUxX5SfvyS3n2qx38siePiEAv3hkdTUSgd5M9vhDnau2u
XB7+PIXjZdWM6xXC/Vd2xNf97KdPzldtneajDft4/ft0Dft4Y/VuarXmvgEdmdy/HSaZbrEIdl3kDUD/HuH
rHUZ5ekUpReTVPDY5kfO9QOXwUFqW8qpaXv01j4cb9RAR68faoaCJbNf2g43BROc+u2MH3
O4/Su11zpo2OkZMHLIDdF/lJ+SWVPPRZCj/vzKJa8MbJbk4x0hPgnmXklTF6YSEZuCbf1
a8vDAzvh4mjc6YFaaz5PzOaZFTtwdzbx9qho+nf8y+pw0YTOVOR2N1ns7+nCvAk9ePbjIZ37Zk
8v17/1OZl6J0bGEnftlTx5DZ/5GQWkVH0/qxVNDOhta4gBKKUbGt+Gruy/Cz9OZCR9uZuba
DOQaBpbH7oocwMFBMalfWz75V2+Kyqu5/r3f+T0j3+hYwg5prZmzLpNbPtxMkK8bK+66iH7h

/kbH+oPwll6suKsf13Zrzeurd/Posm1U19b98zeKJmOXRX5Sj7DmrLjrIlp6u3DzvM0s3XLQ6Ej
CjtTVaf799U5e/CaNqzoHsmxKX4s9S8TN2cS00dHce3kHliZkM/HDzRSVVxsdSzSw6yIHaNPcn
WVT+tKnvR+PLNvG3PVZRkcSdqC6to4HP0th/u/7mNSvLe/dFIuHiyG7Sp81pRQPXNWJN0Z2
Z3NWAaNnbeRYid1eJtOi2H2RA3i5OjFnQjyDogJ5YeVOpq9Jl3lA0WgqqmuZ8nEiy5MO8dBVH
XlqcKRVraYcERfMnAk9yMwrYdSsjeQWVxgdye5JkTdwcTQxY0wMN8QG89YPe3jlu11S5sLsK
qpr+ddHCazZlcsLw6K4+/JwqzwF9pKOAcy/pSc5heXc+MEGDhWWGx3JrkmRn8LR5MDrI7oxv
ncoH/yayVs/7DE6krAhVTV13LloK+vS83n1hvqfM2vWp70fCyf14lhJFaNnbeBIkYzMjSJF/icO
Dop/X9eFUfFtmPFTBjPXZhgdSdiA6to67v5kKz/tyuXl67tyY3wboyOZRVxoMxbe1ouCkirGzd0
kc+YGkSI/DQcHxcvDuzI0uv50q3nyBqi4AHV1mgeXpvD9zqP8+7oujO11fvvxW6roNr7MndiDg
wVl3DxvM8UVcjZLU5MiPwOTg+LNkd0Z2CWQ51fuZEXyIaMjCSukteaFb3byVUoOjw6MYELf
MKMjNYre7fx4f3wce46eYNL8LVRU1xodya5Ikf8NR5MD08ZE07tdcx76LIXfZNGQOEezfs3k
w9/2cetFbbnjknZGx2lUl3VqwTujYkjYf5ypS5KprZOTBZqKFPk/cHE08cH4eNr5e3LHwkTSDhc
bHUlYieVJ2fznu10M6daKpwZHWuXZKedqcLdWPD24M6t2HOGlb9KMjmM3pMjPgo+bEx/e0g
MPF0cmfriZHDnVSvyDjZnHePizbfRp58ebN3a3qvPEL9St/dpy60VtmfdbliywayJS5Gepta8b82
/tQWllLbctSKCsqsboSMJC7csv5Y6PEwn1c+f98XGGb35lhCcHRzKwSyAvfrOT73ccMTqOzZMi
PwcRgd5MHxNN2pFiHlyaQp3MAYo/KSqvZtKCLQDMndADHzcngxMZw+SgeGd0NN2CfZn6aTK
7jsiUZGOSIj9Hl0e05IlBkXyXeoR3fpQFQ+L/1TScK36goIz3x8XZ/UW/XZ1MzBofh6eLI7ctS
JBzzBuRFPl5uO3itoyMC2b6Txl8nZJjdBxhIV75bhfr0vN5cVgUvdv5GR3HIrT0dmXWzfHknqhk
yqKtVNXI9reNQYr8PCileOn6rsSHNuORz7fJYaNgRfIh5qzPYkKfUEb1sK0FPxcquo0vr93Qjc1
ZBbywcqfRcWySFPl5cnZ04L2bYvFydeT2hYmyN7Md25lTzKPLttEzrDlPDelsdByLNCwmiMn92
7Fw436WJWYbHcfmSJFfgBbervx3XCw5heXc/2myvPlphwrLqrj94wR83ZyZeVMsTib5lTqTR6
7uRJ92fjyxfDs7coqMjmNT5KfuAsWFNueZIZ35aVcu09akGx1HNKG6Os3UT5M5WlTJf8fFEu
DlYnQki+ZocmDG2BiauTtzx8eJFJZVGR3JZkiRm8G43qEMjw1i+k/p/Lonz+g4oom893MGP+/O
4+lrOxMT0szoOFbB39OF98bFcqSoQo5izUiK3AyUUrw0rCsdW3gx9dNkDhfJyk9b93tGPm/9sI
eh0a0ZZ2O7GTa22JBmPHNtF9buzuP9X/caHccmSJGbiZuziZk3xVJZXcs9nyTJVcZt2NHiCu5
dkkS7AE9evr6rXeyhYm7jeoUwpFsr3li9m02Zx4yOY/WkyM2oQwtPXh7elYT9x3l99W6j44hG
UFNbxz2LkyitrOW/VnDBZEullOI/w7sS6ufBPYuTyJfFQhdEitzMhkYHMa53CLN+zeSnXUeNji
PMbPqadDZnFfDS9VGEt/QyOo5V83J1YubYWIrKq2W+/AJJkTeCpwZ3pnMrbx5cmiLz5Tbkt
4x8ZqzNYGRcMMNjg42OYxM6t/bm39d1YV16Pv/9RebLz5cUeSNwdTLx7tgYKmvquG9xMjUy
X2718k5Uct+SZNoHePLvoV2MjmNTRvVow7XdW/PWD3tI2FdgdByrJEXeSNoFePLisCg27ytgu
pxfbtXq6jQPLE3mREU1746Nwd1Z5sXNSSnFy9dHEeTrxr2Lk+T88vMgRd6IhscGMyIumBlrM/
h9r1wmzlrNWpfJuvR8nr22CxGB3kbHsUlerk68OzaGvJJKHvl8G1rLfPm5kCJvZM8P7UJbfw/u
/zSZglIZaVibpAPHeWP1bgZ3bcWYnm2MjmPTugX78ujACL7feZSFG/cbHceqSJE3MndnR6aP
juF4aTWPfJ4iIw0rUlxRzb1Lkmjp7crLw+V88aZw60VtubRTAC9+kya7ip4DKfImEBXkw2ODIv
gxLZePNshIwxporXlqeSo5hRVMHxNtt1f6aWoODoo3RnbH29WJexcnUV5Va3QkqyBF3kRuuSi
MyyNa8NK3aaQdlpGGpVu29RBfpeRw34Bw4kKbGx3Hrvh7uvDWjd3Zc7SEF7+R/cvPhlmKXCk
1UCm1WymVoZR6zBz3aWuUUrw+ohs+bk7cIyMNi5aVX8ozK1Lp1bY5d13Wweg4dql/xwBu79
+ORZsOsCpVLt78Ty64yJVSJmAmMAjoDIxRSsnu+qfh1zDSyMiVkYalqqqp474lSTiZHHh7VDQ
mB5kXN8qDV3WiW7APj32xTRbW/QNzjMh7Ahla60ytdRWwBBhqhvu1SReHy0jDkr35w262Z
Rfx6g3daO3rZnQcu+bs6MC00TFU1dQxdUkytbKE/4zMUeRBwMFTPs5uuO0PlFKTlVIJSqmEv
Dz73rP7was60TVIRhqWZn16Ph/8ksnYXiEMjAo0Oo4A2vp78O/rurApq4D3ZQn/GZmjyE937
PmX/zq11rO01vFa6/iAgAAzPKz1cnZ0YPqY+pHG/Z/KSMMSHCup5P6lyXRo4cnTg2Vm0JKMi
Av+3xL+rQeOGx3HIpmjyLOBU1dKBAM5Zrhfm3ZypLExU0YaRtNa8/Dn2ygqr2bGmBjcnE1GR
xKnUErx4rAoAr1duW9JEsUVcqHzPzNHkW8BwpVSbZVSzsBo4Csz3K/Nk5GGZVjw+z5+2pXL
44MiiGwlS/AtkY+bE9PHRJNTWMFTy1NlYd2fXHCRa61rgLuB1UAasFRrveNC79ceKKV46fooW
vm4cu9iGWkYIe1wMS9/t4vLI1owsW+Y0XHE34gLbc7UAeF8lZLDsq2HjI5jUcxyHrnW+lutdUe
tdXut9UvmuE974e3qxLTRMRwukpFGUyurquGexUn4uDnx+ohusgTfCtx5WQd6tW3OMytSyc
wrMTqOxZCVnRYgLrTZ/0YanyVmGx3Hbjz/9U725pXw9o3R+Hm6GB1HnAWTg+LtUdE4mRy4
d0kSVTWy1z9IkVuMOy/rQO92zXl2xQ4ycmWk0di+TslhyZaD3HFJe/qF+xsdR5yD1r5uvDaiG
6mHinlt1S6j41gEKXILYXJQTBtdf8bE3Z9spaJalvA3loMFZTzxxXZiQnx54MqORscR5+HqLoHc
3CeUOeuz5Nq4SJFblJberrwxshu7jpzgP9+mGR3HJlXX1nHP4iRQMH10DE4m+RWwVk9cE0lk
w7VxjxRVGB3HUPJTbGEuj2jJpH5tWbBhP6tSDxsdx+a8vno3yQcLeWV4N9o0dzc6jrgAf7g27
pIku15YJ0VugR4dGEH3YB8e/nwbBwvKjI5jM37adZRZv2YyrncIg7u1MjqOMIP2AZ48PzSKTV
kFTLPja+NKkVsgZ0cH3h0bC8Ddn2yVd+bNIKewnAeWptC5lTdPyRJ8mzIiLpgbYoOZ8VM669L
tcx8nKXIL1aa5O6+P6E5KdhH/+U7myy9EdW0d9y5Oorqmjpk3xeLqJEvwbc0Lw7rQIcCTqUuS
OVpsf/PlUuQWbGBUILdcFMaHv+3ju+0yX36+Xlu1i4T9x3l5eFfa+nsYHUc0AndnR967KZayqlr
uWZxETa19HcVKkVu4xwdF0r2NLw9/vk1Wsp2HVamHmb0ui/G9Qxka/ZfdlYUNCW/pxUvXR

7E5q4A3vt9jdJwmJUVu4ZwdHXjvplicTIopH2+lrKrG6EhWIzOvhIc+20b3Nr48NSTS6DiiCQyP
DWZMzxDe/2WvXV24RYrcCgT5ujF9TAx7ck/w+BfbZT+Ws1BeVcudi7biZFK8d1MsLo4yL24v
nruuM92DfXjosxS7OYqVIrcSF4cH8OCVHVmRnMOC3/cZHceiaa15dNk2dh89wTujYwiSS7bZF
RdHE++Ni8PJpLjj40RKK23/KFaK3IrceWkHrohswYvfpLFh7zGj41isWb9m8lVKDg9d1YlLOtr31
ajsVZCvGzPGxJKRW8Ijy7bZ/FGsFLkVcXBQvDUqmlA/d+76ZKssFjqNX/bk8eqqXVzTNZA7L2
1vdBxhoH7h/jwyMIJvth1m5toMo+M0KilyK+Pt6sTsm+Oprq1j8sJEefPzFPvyS7nnk610bOnF6y
O6y/7igtv7t2NYdGve+H4P3++w3Tc/pcitULsAT2aMiWH3kWIe/mwbdXa8x8RJReXVTFqwBQ
cHxazx8Xi4OBodSVgApRSv3NCNbsE+3P9pMruPnDA6UqOQIrdSl3ZqwWODIvhm+2He/GG30
XEMVV1bx52LEjlQUMb74+II8ZPNsMT/c3UyMWt8PO4ujkxasIW8E5VGRzI7KXIr9q+L2zGm
Zwgz1+5l6ZaDRscxhNaap79M5beMY/xneDd6t/MzOpKwQIE+rsy5OZ78kkpu+yiB8irb2u9fit
yKKaV4fmgXLg7354nl21mfnm90pCb3wa+ZLNlykLsv68CIuGCj4wgL1r2NL9NHx7Atu5B7bWz
bWylyK+dkql/52aGFJ1M+TmRnTrHRkZrMF1uzeeW7XQzu1kqu9CPOylVdAnl2SGd+2HmUF1b
utJnTEqXIbYCXqxPzJvbA09WRm+dtZl9+qdGRGt1Pu47y8Ofb6Nvej7du7I6Dg5yhIs7OxIvacl
u/tsz/fR/v/bzX6DhmIUVuI1r7urFwUk9q6+oYN3eTTV/6KnF/AXcu2kpkKy8+GB8ny+/FOXvi
mkiujwni9dW7bWKltBS5DenQwosFt/bkeGkVN8/bxPHSKqMjmV3qoSJunZ9AKx835t/SEy9X
J6MjCSvk4KB4fUQ3ruzckme/2sGyxGyjI10QKXIb0y3Yl9kT4tl3rIyb5myiwIbKPPVQETfN2Y
SniyMf3doTf08XoyMJK+ZocmDGmBgu6uDHI8u28c02693zX4rcBvVt78/sm+PZm1fC2NkbOV
Zi/efN7sgpYtzc+hJfMrm3XDhZmMXJc8xjQ3y5Z/FWlidZ58hcitxGXdIxgLkTepCVX8qY2Rut
ehHE9uz6kbi7k4nF/5ISF+bl4eLI/Ft60qutHw8sTbHKNRlS5DasX7g/H07swYGCMm78YAP7j1n
f2Sy/7slj1KwNeDg7smRyH1m1KRqFh4sj8yb2oF8Hfx5Zto35v2UZHemcSJHbuL4d/Fl0Wy+Ol
1Ux/L3fSTlYaHSks/Zl0iFunb+FUD8Plt/ZV0pcNCo3ZxOzb47nys4tee7rnby4cqfV7GMkRW4H
4kKbs2xKX9ycTYyetZE1aUeNjvS3tNbMXJvB1E+TiQ9rxqe396aFt6vRsYQdcHUy8f64OCb2D
WPO+izuXLTVKpbzS5HbifYBnnxxZ186tPDkto8SmPZjukWONkoqa5jy8VZeX72b67q3Zv4tPf
GWUwxFEzI5KJ67rgtPD+nM6p1HGDVrg8Xv/S9FbkdaeLny6e29GRYdxNs/7uHWBVsoLLOc0x
P35pUwbOZv/JB2lKcGRzJtdDSuTrLYRxhjUr+2fDAujqy8UgZPX2fR+5lLkdsZd2dH3rqxOy8Oi
+L3jGMMnr6e3zKM3Wyrrk6z4Pd9DJm+noLSKhZO6sltF7eTC0MIw13VJZCV9/YjxM+dyQsT
eWHlTiqqLW+qRRmxaUx8fLxOSEho8scVf5R8sJD7P02uP0WxZxsevyayyacxDhaU8eiybfy+9x
j9Owbw6g1daeUjF0sWlqWyppaXvknjow37aevvwcvXd6VP+6bfMlkplai1jv/L7VLk9q2iupa3f9
jD7HWZtPBy5dFBnbiuexCmRt6EqrSyhlm/ZjJ7XSYKeGpIZ0b3aCOjcGHR1qfn88Ty7fWn9MY
H8/DVEQR4Nd0KYyly8bdSDhbyxPLt7MgpJiLQi4eu6sSAyBZmL9bKmlqWJR7i7R/3kHeiksFd
W/HYoAhZ5COsRnlVLdPWpDN7XSZOJsWEPmFM7t8OvybYMqJRilwpNRJ4DogEemqtz6qdpc
gtU12d5pvth3nrhz1k5ZcSEejF2F4hDI0OwsftwqZcDheV88mmAyzefID8kiriQ5vxxOBIYkOam
Sm9EE0rK7+UGWvS+TL5EK5OJkbEBTMyrg1RQd6NdmTZWEUeCdQBHwAPSZHbhuraOpZvPc
SCDfvYkVOMq5MDV3YO5OIO/vTt4Edws38ePWutycgt4efdeazdncumrALqtGZARAtu7hPGxe
H+Mo0ibEJGbgkz12bwzfbDVNXUERHoxZBurejVzo9uwT5m3Wa5UadWlFI/I0Vuk7ZnF/HJ5g
P8sPMo+Q2bbwX5uhHq506bZu608nXFpBR1Gmrr6jhUWEFWfgmZ+aUUllUD0LGlJ1dEtmRMz
xCZQhE2q6ismq+35fBZYvb/VlA7OzrQpbU3rX3dCPR2paW3C4OiWp3374HhRa6UmgxMBggJ
CYnbv3//BT+uaDpaa/YcLeG3jHySDhZysKCM7OPl/yv3kwK9XWnr70HbAA+6tPbm0k4tCPKV
s1CEfSkorSJhXwGbswpIzSniaHElR4oqKK+u5eNJvegX7n9e93veRa6U+hEIPM2nntRar2j4mp+
REbldqq6tA8CkFEoh0yVCnIHWmpLKGPwdHc57uuVMRe54Fg9+xXk9orALTiZZUybE2VBKNdoV
reS3UAghrNwFFblS6nqlVDbQB/hGKbXaPLGEEEKcLUMWBCml8oDzfbfTHzB2cxDjyWsgr4G9
P3+wz9cgVGsd8OcbDSnyC6GUSjjdZL89kddAXgN7f/4gr8GpZI5cCCGsnBS5EEJYOWss8llGB7
AA8hrIa2Dvzx/kNfgfq5sjF0II8UfWOCIXQghxCilyIYSwclZV5EqpgUqp3UqpDKXUY0bnaUpKq
TZKqbVKqTSl1A6l1H1GZzKKUsqklEpSSq00OosRlFK+SqnPlVK7Gn4e4e+hidqakppe5v+D1IVUotV
kq5Gp3JSFZT5EopEzATGAR0BsYopTobm6pJ1QAPaq0jgd7AXXb2/E91H5BmdAgDTQNWaa0j
gO7Y2WuhlAoC7gXitdZRGAkYbWwqY1lNkQM9gQytdabWugpYAgw1OFOT0Vof1lpvbfj7Cep/
eYOMTdX0lFLBwGBgjtFZjKCU8gb6A3MBtNZVWutCY1MZwhFwU0o5Au5AjsF5DGVNRR4EHD
zl42zssMgAlFJhQAywydgkhngHeIT6K1PZo3ZAHvBhw/TSHKWUh9GhmpLW+hDwBnAAOAwU
aa2/NzaVsaypyE+30bXdnTuplPIElgFTtdbFRudpSkqpIUCu1jrR6CwGcRigf9qrWOAUsDe3i9q
Rv3ReFugNeChlBpnbCpjWVORZwNtvTk4GDs7nFJKOVF4ou01l8YnccAFwHXKaX2+1drlS6
mNjIzW5bCBba33yaOxz6ovdnlwBZGmt87TW1cAXQF+DMxnKmop8zCxCulGqrlHKHm/s2NrwzO
1GRU/aV35gJpWuu3jM5jBK3141rrYK11GPX//r/j9VEp1arhpALDTwEhGOAD0
Vkq5N/xeDMDO3vD9s3+8QpCl0FrXKKXuBlZT/y71PK31DoNjNaWLgPHAdqVUcsNrT2itvzUwk2h7bvzUwk
zDGPcCihgFNJnCLwXmalNZ6k1Lqc2Ar9W9dWzJWHny/Vlib4QQlg5a5paEUIIcRpS5EIIYeWkyI
UQwspJkQshhJWTIhdCCCsnRS6EEFZOilwIazc/wHu8fP7vucweAAAAABJRU5ErkJggg==\n",
"text/plain": [
"<Figure size 432x288 with 2 Axes>"

```
      ]
     },
     "metadata": {
      "needs_background": "light"
     },
     "output_type": "display_data"
    }
   ],
   "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()"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "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)."
   ]
  }
 ],
 "metadata": {
  "colab": {
   "collapsed_sections": [],
   "name": "colab-tutorial.ipynb",
   "provenance": []
  },
  "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"
   }
  },
  "nbformat": 4,
  "nbformat_minor": 1
 }
```