Python Basics

Sumner Evans

September 15, 2020

Mines ACM

A Small Survey

Welcome everyone! I'd like to get to know everyone a bit more and get a feel for everyone's prior experience with programming and Python.

- What year are you in?
- How many of you have programmed in any language before?
- How many of you have programmed in **Python** before?

A Small Survey

Welcome everyone! I’d like to get to know everyone a bit more and get a feel for everyone’s prior experience with programming and Python.

- What year are you in?
- How many of you have programmed in any language before?
- How many of you have programmed in **Python** before?

A Small Survey

Welcome everyone! I’d like to get to know everyone a bit more and get a feel for everyone’s prior experience with programming and Python.

- What year are you in?
- How many of you have programmed in any language before?
- How many of you have programmed in **Python** before?

Overview

1. What is Python?
2. Programming Basics in Python

What is Python?

A Bit of History

- Python first appeared in early 1991. *This means that Python is older than Java and Ruby.*
- Guido van Rossum (GvR, the creator of Python) designed his language with **emphasis on readability**.
- Python was named after *Monty Python's Flying Circus*.
- The language quickly gained popularity because of its appeal to long-time UNIX/C hackers[1].

A Bit of History

- Python first appeared in early 1991. *This means that Python is older than Java and Ruby.*
- Guido van Rossum (GvR, the creator of Python) designed his language with **emphasis on readability**.
- Python was named after *Monty Python's Flying Circus*.
- The language quickly gained popularity because of its appeal to long-time UNIX/C hackers[1].

A Bit of History

- Python first appeared in early 1991. *This means that Python is older than Java and Ruby.*
- Guido van Rossum (GvR, the creator of Python) designed his language with **emphasis on readability**.
- Python was named after *Monty Python's Flying Circus*.
- The language quickly gained popularity because of its appeal to long-time UNIX/C hackers[1].

A Bit of History

- Python first appeared in early 1991. *This means that Python is older than Java and Ruby.*
- Guido van Rossum (GvR, the creator of Python) designed his language with **emphasis on readability**.
- Python was named after *Monty Python's Flying Circus*.
- The language quickly gained popularity because of its appeal to long-time UNIX/C hackers[1].

Why Learn Python?

- Python is a *general purpose, multi-paradigm* language meaning that it is very flexible and can be used in many different scenarios.
- Some of the main applications of Python in industry are web programming, data science, machine learning, automation scripting.
- Python is an easy language to learn.
- Python runs anywhere, and generally requires little setup compared to other languages.

Why Learn Python?

- Python is a *general purpose, multi-paradigm* language meaning that it is very flexible and can be used in many different scenarios.
- Some of the main applications of Python in industry are web programming, data science, machine learning, automation scripting.
- Python is an easy language to learn.
- Python runs anywhere, and generally requires little setup compared to other languages.

Why Learn Python?

- Python is a *general purpose, multi-paradigm* language meaning that it is very flexible and can be used in many different scenarios.
- Some of the main applications of Python in industry are web programming, data science, machine learning, automation scripting.
- Python is an easy language to learn.
- Python runs anywhere, and generally requires little setup compared to other languages.

Why Learn Python?

- Python is a *general purpose, multi-paradigm* language meaning that it is very flexible and can be used in many different scenarios.
- Some of the main applications of Python in industry are web programming, data science, machine learning, automation scripting.
- Python is an easy language to learn.
- Python runs anywhere, and generally requires little setup compared to other languages.

A Note on Python 2 and Python 3

There are two main versions of Python: Python 2 and Python 3. As of earlier this year, Python 2 is no longer supported, so nobody should use it. Unfortunately, many projects and operating systems have not gotten with the times and are still reliant on Python 2.

Python 3 has many major advantages over Python 2 as it fixes many annoying inconsistencies with the older version.

For the purposes of this presentation, we will be talking *strictly of Python 3*.

A Note on Python 2 and Python 3

There are two main versions of Python: Python 2 and Python 3. As of earlier this year, Python 2 is no longer supported, so nobody should use it. Unfortunately, many projects and operating systems have not gotten with the times and are still reliant on Python 2.

Python 3 has many major advantages over Python 2 as it fixes many annoying inconsistencies with the older version.

For the purposes of this presentation, we will be talking *strictly of Python 3*.

A Note on Python 2 and Python 3

There are two main versions of Python: Python 2 and Python 3. As of earlier this year, Python 2 is no longer supported, so nobody should use it. Unfortunately, many projects and operating systems have not gotten with the times and are still reliant on Python 2.

Python 3 has many major advantages over Python 2 as it fixes many annoying inconsistencies with the older version.

For the purposes of this presentation, we will be talking *strictly of Python 3*.

Programming Basics in Python

Follow Along

You can either install Python on your machine or use an online Python environment such as `https://repl.it/languages/Python3`.

Most of the things we will cover today can be done directly in the REPL (read-evaluate-print-loop) on the right, however you may want to write code in the file on the left and run it.

Storing Data

At its core, programming is about storing and manipulating *data*.

In almost every programming language, there is a concept of a **variable** which *stores* data.

In Python, you can create a variable using the following syntax:

```
name = "Sumner"
age = 22
likes_acm = True
```

If you have ever programmed in a language such as Java, you may notice that there is no special keyword for declaring a variable.

Storing Data

At its core, programming is about storing and manipulating *data*.

In almost every programming language, there is a concept of a **variable** which *stores* data.

In Python, you can create a variable using the following syntax:

```
name = "Sumner"
age = 22
likes_acm = True
```

If you have ever programmed in a language such as Java, you may notice that there is no special keyword for declaring a variable.

Storing Data

At its core, programming is about storing and manipulating *data*.

In almost every programming language, there is a concept of a **variable** which *stores* data.

In Python, you can create a variable using the following syntax:

```python
name = "Sumner"
age = 22
likes_acm = True
```

If you have ever programmed in a language such as Java, you may notice that there is no special keyword for declaring a variable.

Showing the Data

Storing data isn't any good unless you can actually use it for something useful. One of the most basic things we can do with the data stored is print it out to the console.

To print anything in Python, use the `print` function:

```
name = "Sumner"
age = 22
print(name)
print(age)
```

If you want to print multiple things at once, you can separate them with a comma:

```
print(name, age)
```

Showing the Data

Storing data isn't any good unless you can actually use it for something useful. One of the most basic things we can do with the data stored is print it out to the console.

To print anything in Python, use the `print` function:

```python
name = "Sumner"
age = 22
print(name)
print(age)
```

If you want to print multiple things at once, you can separate them with a comma:

```python
print(name, age)
```

Showing the Data

Storing data isn't any good unless you can actually use it for something useful. One of the most basic things we can do with the data stored is print it out to the console.

To print anything in Python, use the `print` function:

```python
name = "Sumner"
age = 22
print(name)
print(age)
```

If you want to print multiple things at once, you can separate them with a comma:

```python
print(name, age)
```

Getting Data From the User

Often, we want to get some input from the user and store it in a variable. To do this in Python, we use the `input` function.

```
name = input()
print("Hello", name)
```

We can also optionally include prompt text:

```
age = int(input("How old are you? "))
print("In one year, you will be", age + 1, "years old")
```

What do you think the difference between `input()` and `int(input())` is?

Getting Data From the User

Often, we want to get some input from the user and store it in a variable. To do this in Python, we use the `input` function.

```
name = input()
print("Hello", name)
```

We can also optionally include prompt text:

```
age = int(input("How old are you? "))
print("In one year, you will be", age + 1, "years old")
```

What do you think the difference between `input()` and `int(input())` is?

Getting Data From the User

Often, we want to get some input from the user and store it in a variable. To do this in Python, we use the `input` function.

```
name = input()
print("Hello", name)
```

We can also optionally include prompt text:

```
age = int(input("How old are you? "))
print("In one year, you will be", age + 1, "years old")
```

What do you think the difference between `input()` and `int(input())` is?

Getting Data From the User

Often, we want to get some input from the user and store it in a variable. To do this in Python, we use the `input` function.

```
name = input()
print("Hello", name)
```

We can also optionally include prompt text:

```
age = int(input("How old are you? "))
print("In one year, you will be", age + 1, "years old")
```

What do you think the difference between `input()` and `int(input())` is?

What Sorts of Data Can We Store?

There are many different *types* of data that we can store in Python. Here are the most basic data types (primitives):

- `bool` — either `True` or `False`
- `int` — an integer
- `float` — a real number[2]
- `string` — a sequence of characters[3]

[2]Not all real numbers can be represented as a floating point number, but that's not normally important.
[3]Note that unlike other languages, there is no `char` datatype. Chars are just one-character strings.

Manipulating Data: Assigning a New Value to a Variable

Having variables to store is nice, but a lot of times we want to modify the value stored in the variable!

To do that, we use a very similar syntax to defining variables:

```python
name = "Sumner"  # declares a variable and gives it an
                 # initial value
print(name)

name = "Jack"    # assigns the value "Jack" to the "name"
                 # variable
print(name)
```

Manipulating Data: Basic Operations

Similar to how you can perform operations on variables in algebra, you can perform operations on variables. Here are some basic operations on primitive data types in Python:

- `+`, `-`, `*`, `/`, `//`: add, subtract, multiply, divide, integer division.
- `**`: exponentiate (3^8 would be written `3**8`).
- `<`, `>`, `==`: tests if two numbers are less than, greater than, equal to each other, respectively.

Try it Yourself!

Try creating a few different variables of various types and performing operations on them.

A few examples to get you started:

```
pi = 3.14159265
r = 10
print("diameter =", pi * (r ** 2))

email = input("Enter your email: ")
email_again = input("Verify your email: ")
print(email == email_again)
```

Making Decisions: Selection Using `if`

It’s all well and good that we can compare numbers and get a boolean value, but we need to make *decisions* with that information. That’s where **`if` statements** come in.

The syntax for `if` statements in Python is:

```
if condition1:
    # do whatever is in this indented section if condition1
    # is True
elif condition2:
    # do whatever is in this indented section if condition2
    # is True
else:
    # do whatever is in this indented section otherwise
```

You can have arbitrary many `elif` blocks (including no elif blocks). It is also not necessary to have an `else` block.

Making Decisions: Selection Using `if`

It's all well and good that we can compare numbers and get a boolean value, but we need to make *decisions* with that information. That's where **`if` statements** come in.

The syntax for `if` statements in Python is:

```python
if condition1:
    # do whatever is in this indented section if condition1
    # is True
elif condition2:
    # do whatever is in this indented section if condition2
    # is True
else:
    # do whatever is in this indented section otherwise
```

You can have arbitrary many `elif` blocks (including no elif blocks). It is also not necessary to have an `else` block.

Making Decisions: Selection Using `if`: Example

Here is a simple example of an `if` statement at work:

```python
likes_python = input("Do you like Python? (y/n)")
if likes_python == "y":
    print("You like Python!")
elif likes_python == "n":
    print("You don't like Python :(")
else:
    print("I don't know if you like Python...")
```

Making Decisions: Selection Using `if`: Your Turn!

Now it’s your turn! Try to write some Python which does the following:

1. Creates a variable with a secret number of your choice.
2. Asks the user to guess a number.
3. Tells the user if their guess is above, below, or equal to the secret number.

Extra Credit: Look up the documentation for the `random.randint` function and see if you can make the secret number random.

ohea

* Variables * Data types * Basic operations on simple data types * If statements * For loops * Functions * Parameters * Union data types