Introduction to Python

Basic Datatypes

Topics

1) Software, Python Scripts
2) repl.it
3) Printing with print()
4) Basic Built-In Number Types
    a) Integers
    b) Floats
    c) Booleans
5) Casting
6) User inputs

Software

A **program** is a collection of program statements that performs a specific task when run by a computer. A program is often referred to as **software**.

A program can be written in many programming languages. We will be using Python in this course.

By convention, Python code is stored in a file called a **script** with a *.py* extension. Scripts are written using an **IDE**(Integrated Development Environment).

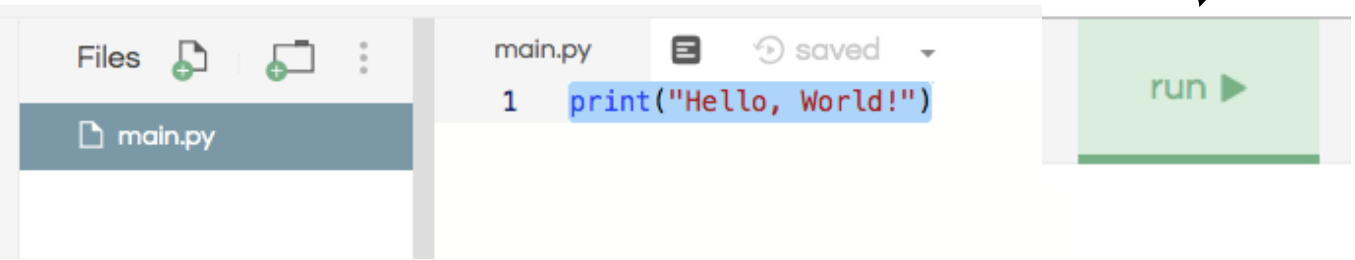
We will initially use an online IDE (repl.it, login with your Google account) to learn the basics of Python. A popular IDE that can be installed locally on your computer is Visual Studio Code.

Python Scripts

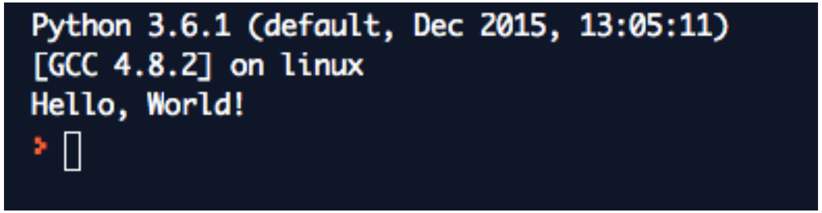
```python
print("Hello, World!")
```

repl.it: Create a new repl and pick Python as the language, name the repl.

Type in the code in the file main.py. Click on run.



```python
print("Hello, World!")
```



```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
Hello, World!
› ▯
```

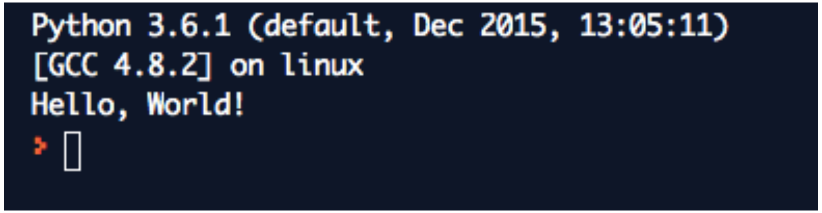
Our First Program

```
print("Hello, World!”)
```

1) The print() function prints messages on the console.

2) Characters enclosed in quotes(single or double) forms a **literal string**.

3) The console output string does not include the quotes.



```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
Hello, World!
› ▯
```

print()

```
print("hello”)
print("Mike”)
print()
print("line1\nline2\nline3”)
```

By default, print() will end each output with a newline character. A **newline character** is a special control character used to indicate the end of a line.

In a string literal, '\n’ denote a newline character.

Output:

```
hello
Mike

line1
line2
line3
```

empty line

print()

```
print("hello”)
print(4)
print(3.14)
print(3 + 4)
```

Note: The console output string does not include the quotes.

Output:

```
hello
4
3.14
7
```

print() will evaluate math expressions before printing.

print()

The print function can accept any number of **positional arguments**, including zero, one, or more arguments.

Arguments are separated by commas. This is useful when you'd want to join a few elements together(e.g. strings, numbers, math expressions, etc…).
print() concatenated all arguments passed to it, and it inserted a single space between them.

```
print(”I have”, 3, ”apples.”)
print(”You have”, 3+2, ”apples.”)
```

Output:

```
I have 3 apples.
You have 5 apples.
```

Dynamic Typing

Python is ***dynamically typed***: variable names can point to objects of any type.

Unlike Java or C, there is no need to “declare” the variable.

```
x = 1           # x is an integer
x = ‘hello’     # now x is a string
```

Comments are a form of **program documentation** written into the program to be read by people and do not affect how a program runs. Python uses # for comments.

Variables

We can use variables to refer to values that can be used later. You can create a new variable by given it a value.

```
x = 4
print(x)        # 4
```

Variable names can use letters, digits, and the underscore symbol (but they can't start with a digit). It is considered best practice to use meaningful variable names:

```
num_apples = 10
num_oranges = 5
total = num_apples + num_oranges
```

= is not equality

Unlike in math, = is not equality in Python.

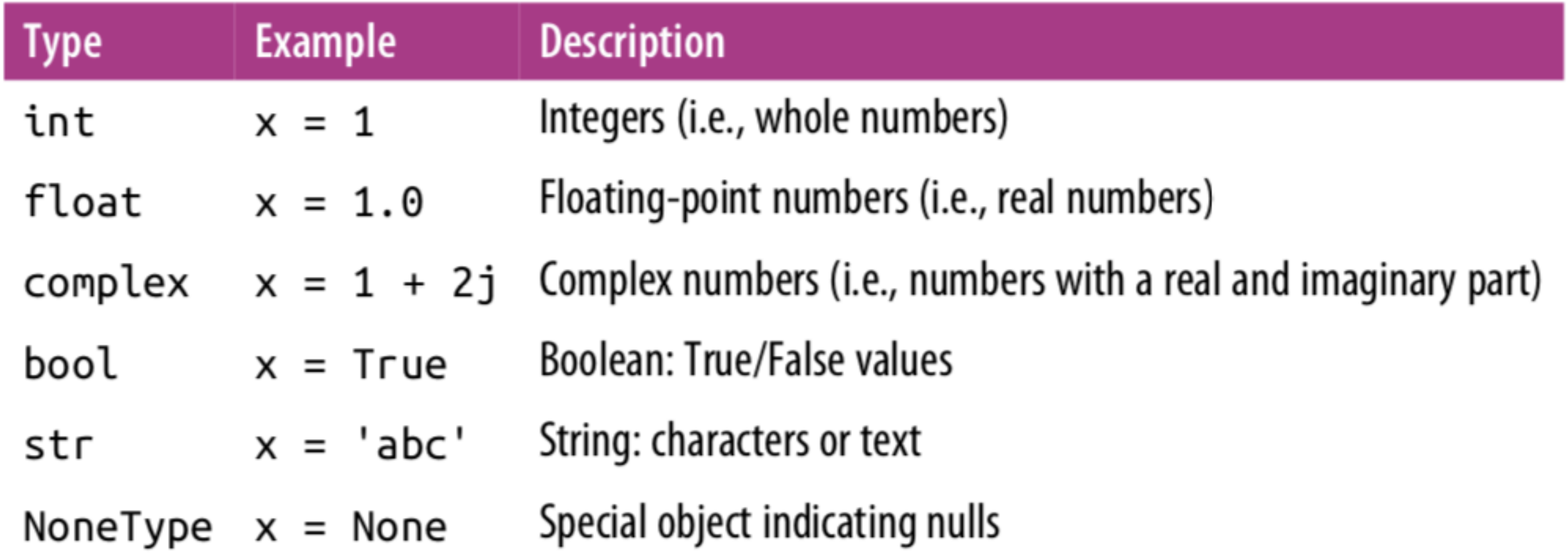
It is an **assignment**: assign the expression on the right side of = to the variable on the left.

```
x = 4
x = x + 1   # in math, this has no solutions!
            # evaluate right side, assigns to left side variable
print(x)    # 5
```

Assignment is not symmetric.

```
x = 4     # correct!
10 = y    # error!
```

Basic Built-In Types



Type	Example	Description
`int`	`x = 1`	Integers (i.e., whole numbers)
`float`	`x = 1.0`	Floating-point numbers (i.e., real numbers)
`complex`	`x = 1 + 2j`	Complex numbers (i.e., numbers with a real and imaginary part)
`bool`	`x = True`	Boolean: True/False values
`str`	`x = 'abc'`	String: characters or text
`NoneType`	`x = None`	Special object indicating nulls

In this lecture, we'll focus on integers, floating-point numbers, strings and boolean values.

Integers

The most basic numerical type is the integer. Any number without a decimal point is an **integer.**

Python integers are variable-precision, so you can do computations that would overflow in other languages.

```
x = 1
y = 2 ** 200
print(”x:”, x)
print(”y:”, y)
```

```
Output:
x: 1
y: 1606938044258990275541962092341162602522202993782792835301376
```

Floating Point

The **floating-point type** can store fractional numbers(i.e. real numbers).

The built-in `type()` function identifies the type of the variable.

```
x = 0.5
print(x)            # 0.5
print(type(x))      # <class ’float’>
y = ”3”
print(type(y))      # <class ’str’>
z = 3
print(type(z))      # <class ’int’>
```

Boolean Type

The **Boolean type** is a simple type with two possible values: True and False. Boolean values are case-sensitive: unlike some other languages, True and False must be capitalized!

Comparison operators return True or False values.

```
result = (4 < 5)
print(result)     # True
print(3 >= 5)     # False
print(3 != 5)     # True
```

Strings

In Python, text is represented as a **string**, which is a sequence of *characters* (letters, digits, and symbols). We indicate that a value is a string by putting either single or double quotes around it.

```
p1 = "Aristotle”
p2 = 'Isaac Newton’
```

Whenever you create a string by surrounding text with quotation marks, the string is called a **string literal**. The name indicates that the string is literally written out in your code

Casting

The `int()`, `float()` and `str()` functions can be called to **cast** a value to an integer or float, respectively.

```
x = 1.8
y = int(”3”)   # String is casted to an integer.
z = float(”3”) # String is casted to a float.
w = int(x)     # float is casted to an integer(truncates)
v = str(x)     # float is casted to string ”1.8”

print(y, type(y))   # 3 <class ’int’>
print(z, type(z))   # 3.0 <class ’float’>
print(w, type(w))   # 1 <class ’int’>, truncates decimal point
print(v, type(v))   # 1.8 <class ’str’>, no ”” when printing
```

Program Inputs

Program input is data sent to a computer for processing by a program. Input can come in a variety of forms such as:

- tactile(swipes from a tablet)
- audio(input can be an audio/voice to be processed by a program)
- visual(an image to be filtered by a program)
- text(user input from keyboard or can be a text file input)

Program outputs are any data sent from a program to a device. Program output can come in a variety of forms, such as tactile, audio, visual, or text.

Program Inputs

A program is useful if it takes some input from the user, process it and outputs something meaningful.

We will start with a simple program that accepts user inputs from the keyboard and outputs some result by printing it on the console.

The input function `input()` can be used to accept inputs from the user.

Input

Programs may use the input function `input()` to obtain information from the user. The program waits for the user to enter some input. The inputted value can be stored in a variable once the user presses Enter.

```python
print('Please enter some text:')
x = input()
print('You entered:', x)
print('Type:', type(x))
```

Note that user input is always a string.

```
Please enter some text:
123
You entered: 123
Type:  <class 'str'>
```

The variable x stores the string literal "123". x is not the integer 123!

Input

Since user input almost always requires a message to the user about the expected input, the input function optionally accepts a string that it prints just before the program stops to wait for the user to respond.

```python
x = input('Please enter an integer value: ')
y = input('Please enter another integer value: ')
x = int(x)      # casts to an integer
y = int(y)      # casts to an integer
print(x, '+', y, '=', x+y)
```

```
Please enter an integer value: 4
Please enter another integer value: 5
4 + 5 = 9
```

Input

Or even more succinctly.

```
x = int(input(‘Please enter an integer value: ’))
y = int(input('Please enter another integer value: ‘))
print(x, '+', y, '=', x+y)
```

```
Please enter an integer value: 4
Please enter another integer value: 5
4 + 5 = 9
```

Functions

Throughout this lecture, we were introduced to many functions: print(), int(),float(), str(), type() and input(). These functions are no different than functions you have seen in your math class. Understanding this will help you call functions correctly with the right syntax.

If you have a function in math $f(x) = x^2$. Then the value of $f(3)$ is 9.

Similarly, in Python, for the int() function, the value of int(4.5) is 4.

The value of the variable x below has the value of 3.0:

```
x = float("3")
```

If $g(x) = 3x$ then the function composition $f(g(2))$ has the value 36.

Similarly, the value of the function composition `int(float("3.2")` is 3.

Another example of function composition we saw earlier:

```
x = int(input('Please enter an integer value: '))
```

Lab 1

Create a new repl on repl.it. Write code to match the following console output:

```
Enter your name: Mike
Hello Mike

Enter an integer: 10
Your number 10 doubled is 20.
The next number after 10 is 11.
```

References

1) Vanderplas, Jake, A Whirlwind Tour of Python, O’reilly Media.

This book is completely free and can be downloaded online at O’reilly’s site.