# Programming in Python 3

# Programming transforms your computer from a home appliance to a power tool

Al Sweigart, The “invent with Python” Blog

# Programming

- Write programs that solve a problem using a set of computational steps
- Write programs/scripts that automate a repetitive task
- Write “glue code” that coordinates several other programs to achieve a task

# Algorithms

- Algorithms are the computational steps that the computer needs to follow to perform a function
- Computer programs implement algorithms

# Algorithm: find max of 4 numbers a, b, c, d

Let x represent the largest integer

At the end of algorithm, x must be the largest of the 4

1. Assume a is the largest. i.e assign x value of a
2. If b greater than x, assign x value of b
3. If c greater than x, assign x value of c
4. If d greater than x, assign x value of d
5. Output value of x

Computer implementation

Requires 5 variables to store a,b,c,d and x.

A comparison operation

An assignment operation

Python

- Python is a high-level, general purpose programming language
- Elegant, readable syntax that is easy to learn
- A large collection of standard modules. Batteries included philosophy
- Allows a programmer to create programs that perform complex functions in less time and less lines of code than other languages

# Python Versions

- The Python language has been steadily evolving with the addition of new language features
- The 2.x series gained wide acceptance and has a large base of existing scripts
- 2008: Python 3.0 was released in parallel with 2.6

# Python 3

- Python 3 aimed to clean up flaws in the Python language
  - But this required changes that would make it “backward incompatible”
  - Existing 2.x code will not run in Python 3 without modification
- Python 2.7 is the last release of the 2.x interpreter support will end in 2020
- Python 3 represents the future of the language
  - But you will most likely encounter existing code written in 2.x
  - We will learn Python 3 in this course

# Python Versions

- Most Linux distributions have both Python 2.7 and 3.x installed side by side

```
$ python --version
Python 2.7.12

$ python3 --version
Python 3.5.2
```

# Interactive Interpreter

- Python provides a shell interface to the interpreter
- Much like the Bash shell
- Each line entered is executed by the interpreter
- Running the Python interpreter without any arguments starts an interactive session

```
$ python3
Python 3.5.2 (default, Jul  5 2016, 12:43:10)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- >>> prompt indicates that the interpreter is ready to accept Python statements

Python

# Interactive Interpreter

- The interpreter prints output after each statement is executed
- Suitable for doing quick tests and trying out Python
- Our first line of code

```
>>> print('Hello, World')
Hello, World
>>>
```

Python

Getting Help

- Documentation can be accessed with the `help()` built-in function

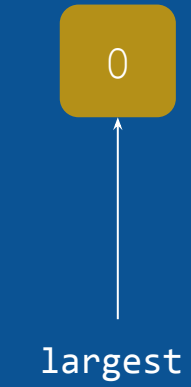
```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
     print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

     Prints the values to a stream, or to sys.stdout by default.
     Optional keyword arguments:
     file:  a file-like object (stream); defaults to the current sys.stdout.
     sep:   string inserted between values, default a space.
     end:   string appended after the last value, default a newline.
     flush: whether to forcibly flush the stream.
```

# Variables

- Variables in Python are created with the an assignment statement
- A variable in Python is simply an identifier

```
>>> largest = 0
>>>
```

- In this case the name `largest` is a reference to an object in memory holding a `0` integer value
- The name `largest` is said to be bound to the integer object `0`



Variables

- Reassigning a variable to another value causes it to be bound to a new object

```
>>> largest = 0
>>> largest = 'aeiou'
>>>
```

- `largest` is now bound to a string object

## Variables

- Python is a dynamically typed language
- The `type()` built-in function can tell us what is the type referenced by a variable

```
>>> largest = 0
>>> type(largest)
<class 'int'>
>>> largest = 'aeiou'
>>> type(largest)
<class 'str'>
>>>
```

Variables

- Even a reassignment to a new value of the *same* type creates a new object

```
>>> largest = 0
>>> largest = -12
>>> type(largest)
<class 'int'>
>>>
```

- Now the name `largest` is a reference to an object in memory holding a `-12` integer value

Expressions

Literals

- Whatever appears on the right side of an assignment is an expression
- i.e., expression evaluates to an object with a value. And the assignment operation binds a variable name to that object
- One of the simplest expressions are *literals*
- In the previous examples `0` is an *integer literal* and `'aeiou'` is a *string literal*

- Integer literals in Python can additionally be input in Hexadecimal, Octal and Binary notation
- Hexadecimal is a numeral system with 16 symbols
  Numerals are `0 1 2 3 4 5 6 7 8 9 A B C D E F`
  Prefixed with `0x`
- Octal has 8 symbols
  Prefixed with `0o`
- Binary has 2 symbols: `0` and `1`
  Prefixed with `0b`
- `print()` function accepts variables as arguments
  And multiple arguments separated by commas

```
>>> a = -63
>>> b = -0x3F
>>> c = -0o77
>>> d = -0b00111111
>>> print(a,b,c,d)
-63 -63 -63 -63
```

- Floating point numbers are the computer approximation of real numbers

```
>>> f = 3.14159
>>> type(f)
<class 'float'>
>>> print(f)
3.14159
```

- Scientific notation also supported
- Note: # character begins a comment
  Everything from the # to the end of the line ignored by interpreter

```
>>> f = 1.3e-9 # same as 1.3x10^-9
>>> type(f)
<class 'float'>
>>> print(f)
1.3e-09
```

Expressions

Complex Literals

- Complex numbers consist of a real part and an imaginary part
- Each of which is a `float` type
- Imaginary unit uses symbol `j` (rather than i)

```
>>> cnum = 2+3j
>>> type(cnum)
<class 'complex'>
>>>
```

Expressions

Attributes

- The complex type has two *attributes*
  `real`
  `imag`
- Attributes are accessed using the . (dot) operator

```
>>> print(cnum.real)
2.0
>>> print(cnum.imag)
3.0
>>> type(cnum.real)
<class 'float'>
>>>
```

Expressions

Bool Type

- Boolean type takes either of two values `True` or `False`

```
>>> mybool = True
>>> type(mybool)
<class 'bool'>

>>> mybool = False
>>> type(mybool)
<class 'bool'>
```

Expressions

String Literals

- An `str` type represents a sequence of characters
- Characters of a string can be enclosed in either single or double quotes
- A single quote can be present unescaped in a double quoted string and vice versa

```
>>> mystr = 'a string'
>>> type(mystr)
<class 'str'>

>>> mystr = "it's a word"
>>> print(mystr)
it's a word

>>> mystr = 'won't work with single quotes'
  File "<stdin>", line 1
    mystr = 'won't work with single quotes'
                  ^
SyntaxError: invalid syntax
```

# String Literals

- String literals can be implicitly concatenated
- Allows us to split long strings across lines in scripts

```
>>> mystr = 'supercalifrag' 'ilisticexpialidocious'
>>> print(mystr)
supercalifragilisticexpialidocious
```

Expressions

String Literals

- Triple quotes preserve newlines
- In the interactive interpreter, … is the secondary prompt
  - Interpreter expects continuation lines

```
>>> mypara = """This is a multi-line
... string. Each line in the
... paragraph is separated by a
... newline character"""

>>> print(mypara)
This is a multi-line
string. Each line in the
paragraph is separated by a
newline character
```

Expressions

String Literals

- Python 3 has full native support for Unicode
- Unicode defines a standard for how text of most of the world's writing systems is represented and stored on computers

```
>>> myname = 'ഇർഷാദ് ബഷീർ'
>>> print(myname)
ഇർഷാദ് ബഷീർ
>>>
```

- Like `str`, `bytes` is a sequence type
- A `bytes` type represents a sequence of 8-bit bytes
- Therefore each element represents an integer between 0 and 255 inclusive
- Byte literal syntax like strings but prefixed with a ‘b’
- Elements can be represented by
  - ASCII characters for values 0-127 and/or
  - hexadecimal digits: \x00-\xFF for value 0-255

```
>>> bvar = b'abc'
>>> type(bvar)
<class 'bytes'>
>>> print(bvar)
b'abc'
>>>
```

# Types Seen So Far

- Numeric Types
  - `int`
  - `float`
  - `complex`
  - `bool`
- Sequence Types
  - `str`
  - `bytes`

# Operators

## Operations on Numeric Types

- All familiar operations available for numeric types
- Numeric literals along with operators are expressions: They return an object that can be bound to a name

```
>>> a = 2 + 1 * 5
>>> print(a)
7
>>>
```

## Operators

### Arithmetic

- Operators follow precedence rules
- Use of parenthesis can make evaluation order explicit

```
>>> a = (2 + 1) * 5
>>> print(a)
15
>>>
```

- Other operators include
  - Raise to the power: `**`
  - Modulo: `%`

```
>>> print(2**4)
16
>>>

>>> print(13%10)
3
>>>
```

# Numeric Type Conversion

- In expressions where arithmetic operators are applied between two numeric values that are not of the same type
  - The less general type is converted to the more general one
  - Generality increases in the order

    `bool -> int -> float -> complex`

```
>>> type(2 + 3)
<class 'int'>
>>> type(2 + 3.5)
<class 'float'>
>>> type(3.5 + (2+1j))
<class 'complex'>
>>>
```

# Division Operators

- Python 3 has two division operators: `/` and `//`
- `//`
  The floor division operator

```
>>> a = 8 // 3
>>> print(a)
2

>>> a = -3.4 // 2
>>> print(a)
-2.0
```

- `//` rounds towards negative infinity

# Division Operators

- /
  Performs true division

```
>>> a = 8 / 3
>>> print(a)
2.6666666666666665
>>> a = -3.4 / 2
>>> print(a)
-1.7
>>>
```

# Comparison Operators

## Numeric types

- Return a value of type `bool`
- Operators are `<`, `>`, `<=`, `>=`, `!=`, `==`

```
>>> print(1 < 2)
True

>>> a = 123
>>> print(a == 12.2)
False
>>>
```

# Comparison Operators

## Numeric types

- Be wary of comparing floating point values directly

```
>>> a = 0.1 + 0.1 + 0.1
>>> print(a == 0.3)
False
>>>
```

Operators

Comparison

- Comparison operators can be chained

```
>>> a = 1
>>> b = 2
>>> c = 3
>>> print(a < b < c)
True
>>> print(a < b  and b < c)  #Functionally identical
True
```

Operators

Logical

# Boolean Operators

## Numeric types

- Operators: `and, or, not`
- `and`

```
>>> print(True and True)
True
>>> print(True and False)
False
>>> print(False and True)
False
>>> print(False and False)
False
```

Operators

Logical

# Boolean Operators

## Numeric types

- `or`

```
>>> print(True or True)
True
>>> print(True or False)
True
>>> print(False or True)
True
>>> print(False or False)
False
```

Operators

Logical

# Boolean Operators

## Numeric types

- `not`

```
>>> print(not True)
False
>>> print(not False)
True
```

# `if` statement

- Execute one or more statements based on the boolean value of an expression

```
if <expression>:
    statement-1
    statement-2
    …
```

- `statement1, statement2, …` are executed if boolean conversion of `expression` has value `True`

- Statement block begins with a clause that ends in colon `:`
- Suite of statements to be executed are at the same indentation level: indentation mandatory in Python

```
if <expression>:
    statement-1
    statement-2
    statement-3
statement-4
```

- Statements 1-3 are conditionally executed
- `statement4` is executed unconditionally

  `statement4` is outside the `if` block

- Nested statement block adds a level of indentation

```
if <expr-1>:
    statement-1
    if <expr-2>:
        statement-2
    statement-3
statement-4
```

- `statement-1`, `statement-3` executed if `expr-1` is `True`
- `statement-2` executed if `expr-1` and `expr-2` are `True`
- `statement-4` executed regardless of `expr-1` or `expr-2`

# Algorithm: find maximum of 4 number a, b, c, d

Let x represent the largest integer

At the end of algorithm, x must be the largest of the 4

1. Assume a is the largest. i.e assign x value of a
2. If b greater than x, assign x value of b
3. If c greater than x, assign x value of c
4. If d greater than x, assign x value of d
5. Output value of x

Computer implementation

Requires 5 variables to store a,b,c,d and x.

A comparison operation

An assignment operation

# Implementation

- Program written into a file

```
a = -12
b = 22
c = 2.222
d = -22.00001

x = a
if b > x:
    x = b
if c > x:
    x = c
if d > x:
    x = d
print(x)
```

```
$ python3 largest.py
22
```

Operators

# `else` and `elif` clauses

```
>>> a = 23
>>> if a % 2 == 0:
...     print("a is even")
... else:
...     print("a is odd")
...
a is odd
```

Operators

# `else` and `elif` clauses

## Nested `if` and `else`

```
if char == 'a':
    print("char is 'a'")
else:
    if char == 'b':
        print("char is 'b'")
    else:
        if char == 'c':
            print("char is 'c'")
        else:
            print("none of the above")
```

Operators

# `else` and `elif` clauses

## Using `elif`

```
if char == 'a':
    print("char is 'a'")
elif char == 'b':
    print("char is 'b'")
elif char == 'c':
    print("char is 'c'")
else:
    print("none of the above")
```