Python Language Fundamentals

olsen software

Contents

1. Defining and using modules
2. Defining and using packages
3. Basic data types

Demo folder: **02-PythonLang**

1. Defining and Using Modules

- The Python standard library
- Understanding modules
- More about modules
- Listing the names in a module

The Python Standard Library

- Python defines an extensive and powerful standard library
  - Comprises a large number of modules

- Built-in modules are implemented in C
  - Provide access to low-level system functionality
  - E.g. file I/O

- Other modules are implemented in Python
  - See the `Lib` folder in the Python installation folder

- For full info, see:
  - https://docs.python.org/3.8/library/

Understanding Modules

- You can create your own Python modules
  - Here's a simple module, which just defines some variables

```
morning = "Good morning"
afternoon = "Good afternoon"
evening = "Good evening"
```

greetings.py

- To use a module elsewhere, use the `import` keyword
  - Several ways to do this:

```
import greetings
print(greetings.morning)
```

```
from greetings import morning, afternoon
print(morning + " " + afternoon)
```

```
from greetings import *
print(morning + " " + afternoon + " " + evening)
```

More About Modules

- You can access the name of a module
  - Use the __name__ property

```
import greetings

print("Name of current module is %s" % __name__)
print("Name of greetings module is %s" % greetings.__name__)
```

usegreetings.py

- Python only imports a given module once
  - Regardless of how many times you try to import it

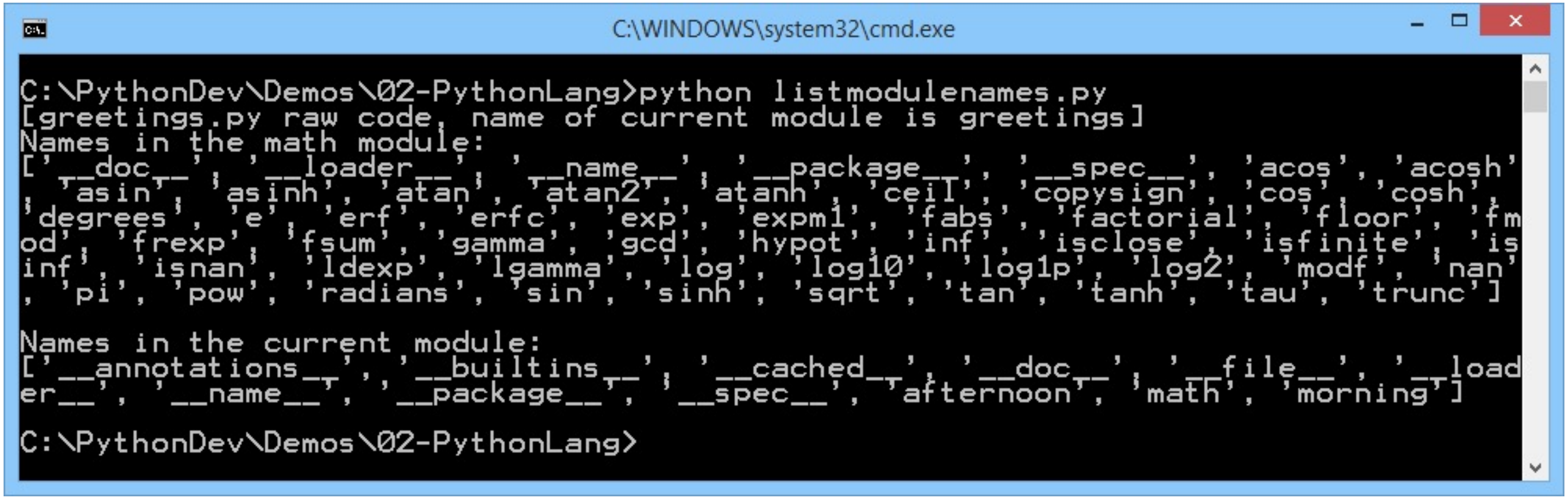
- Python searches the following locations for a module
  - The directory containing the input script (or the current directory)
  - The directory specified by PYTHONPATH
  - The installation-dependent default

Listing the Names in a Module

- You can list all the names defined in a module
  - Use the `dir()` built-in function

```
import math
from greetings import morning, afternoon

print("Names in the math module:")
print(dir(math))

print("\nNames in the current module:")
print(dir())
```

listmodulenames.py



```
C:\PythonDev\Demos\02-PythonLang>python listmodulenames.py
[greetings.py raw code, name of current module is greetings]
Names in the math module:
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh'
, 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fm
od', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'is
inf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan'
, 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']

Names in the current module:
['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__load
er__', '__name__', '__package__', '__spec__', 'afternoon', 'math', 'morning']

C:\PythonDev\Demos\02-PythonLang>
```

2. Defining and Using Packages

- Overview of packages
- Example modules
- Importing specific modules
- Aliasing imported modules
- Importing all modules

Overview of Packages

- Python allows you to organise related modules into packages and sub-packages
  - A package is a folder that contains a file named `__init__.py`

- Example

```
utils/                    Top-level package, named utils.
    __init__.py           Initialize the utils package.

    constants/            Sub-package for constants.
        __init__.py       Initialize the constants package.
        metric.py
        physics.py
              ...

    messages/             Sub-package for messages.
        __init__.py       Initialize the messages package.
        french.py
        norwegian.py
              ...
```

Example Modules

- Here are the modules we've defined in the `utils` package
  - Modules in the `utils.constants` sub-package:

```
INCH_TO_CM = 2.54
MILE_TO_KM = 1.61
```

`metric.py`

```
ELECTRONIC_CHARGE = 1.602e-19
PLANCKS_CONSTANT = 6.626e-34
```

`physics.py`

  - Modules in the `utils.messages` sub-package:

```
HELLO = "Bonjour"
GOODBYE = "Au revoir"
```

`french.py`

```
HELLO = "Hei"
GOODBYE = "Ha det bra"
```

`norwegian.py`

Importing Specific Modules

- To import specific module(s) from a package:

```
import utils.constants.metric

print("Inch to centimetre: %.4f" % utils.constants.metric.INCH_TO_CM)
print("Mile to kilometre:  %.4f" % utils.constants.metric.MILE_TO_KM)
```

useutils.py

- To import specific module(s) from a package, into the current symbol namespace:

```
from utils.constants import metric

print("Inch to centimetre: %.4f" % metric.INCH_TO_CM)
print("Mile to kilometre:  %.4f" % metric.MILE_TO_KM)
```

useutils.py

- To import specific name(s) from a module from a package, into the current symbol namespace:

```
from utils.constants.metric import INCH_TO_CM, MILE_TO_KM

print("Inch to centimetre: %.4f" % INCH_TO_CM)
print("Mile to kilometre:  %.4f" % MILE_TO_KM)
```

useutils.py

Aliasing Imported Modules

- You can specify a local alias for a module
  - Use `import … as`

```
# import a module and give it an alias.
import utils.constants.metric as metric

print("Alias example")
print("Inch to centimetre: %.4f" % metric.INCH_TO_CM)
print("Mile to kilometre:  %.4f" % metric.MILE_TO_KM)
```

useutilsAliased.py

Importing All Modules

- You can use * to indicate you want to import all modules from a package

```
from utils.messages import *

print("Hello in French:   %s"    % utils.messages.french.HELLO)
print("Goodbye in French: %s"    % utils.messages.french.GOODBYE)
print("Hello in Norwegian:   %s" % utils.messages.norwegian.HELLO)
print("Goodbye in Norwegian: %s" % utils.messages.norwegian.GOODBYE)       useutils.py
```

- You must tell Python which modules to actually import from that package
  - In the package's `__init__.py` file …
  - Define a global variable named `__all__` and set it to a list of all the modules to be imported

```
__all__ = ["french", "norwegian"]                         utils/messages/__init__.py
```

3. Built-in Types

- Numbers
- Numeric operators
- Bitwise operators
- Using the `math` module
- Booleans
- Relational operators
- Boolean logic operators
- Operator precedence
- Strings
- Other built-in types

Numbers

- Python has three numeric types
  - Integers
  - Floating point numbers
  - Complex numbers

```
i1 = 12345
i2 = 1234567890123456789
i3 = int("123", 8)
print("%d %d %d" % (i1, i2, i3))

f1 = 1.23
f2 = 4.56e-34
f3 = 7.89e+34
f4 = float("123.45")
print("%g %g %g %g" % (f1, f2, f3, f4))

c1 = 1 + 2j
c2 = 3 - 4j
c3 = 5j
c4 = complex("6+7j")
print("%g + %gi" % (c1.real, c1.imag))
print("%g + %gi" % (c2.real, c2.imag))
print("%g + %gi" % (c3.real, c3.imag))
print("%g + %gi" % (c4.real, c4.imag))
```

numbers.py

Numeric Operators

- Python supports the following operators on numbers
  - `x ** y`
  - `pow(x, y)`
  - `divmod(x, y)`
  - `c.conjugate()`
  - `complex(re, im)`
  - `float(x)`
  - `int(x)`
  - `abs(x)`
  - `+x`
  - `-x`
  - `x % y`
  - `x // y`
  - `x / y`
  - `x * y`
  - `x - y`
  - `x + y`

Bitwise Operators

- Python supports the following bitwise operators on integers
  - ~x
  - x >> n
  - x << n
  - x & y
  - x ^ y
  - x | y

Using the `math` Module

- The `math` module defines several useful mathematical constants and functions
    - For details, see https://docs.python.org/3.8/library/math.html

- Example

```
import math

print(dir(math))

print("pi is %f" % math.pi)
print("360 degrees in radians is %g" % math.radians(360))
print("2 * pi radians in degrees is %g" % math.degrees(2 * math.pi))

print("sin(90 degrees) is %.4f" % math.sin(math.pi / 2))
print("cos(90 degrees) is %.4f" % math.cos(math.pi / 2))
print("acos(0) is %g degrees" % math.degrees(math.acos(0)))

print("hypoteneuse of right-angled triangle (sides 3, 4) is %g" % math.hypot(3, 4))
print("5 factorial is %g" % math.factorial(5))
```

`usemath.py`

Booleans

- Boolean is a built-in type
  - Represents truth or falsehood
- The following values are considered false:
  - `None`
  - `False`
  - Zero of any numeric type, e.g. `0`, `0.0`, `0j`
  - Any empty sequence, e.g. `''`, `()`, `[]`
  - Any empty mapping, e.g. `{}`
- All other values are considered true
  - Including the `True` keyword ☺

Relational Operators

- Python supports the following relational operators
  - `<`
  - `<=`
  - `>`
  - `>=`
  - `==`
  - `!=`
  - `is`
  - `is not`

Boolean Logic Operators

- Python has thee boolean logic operators:
  - not
  - and
  - or

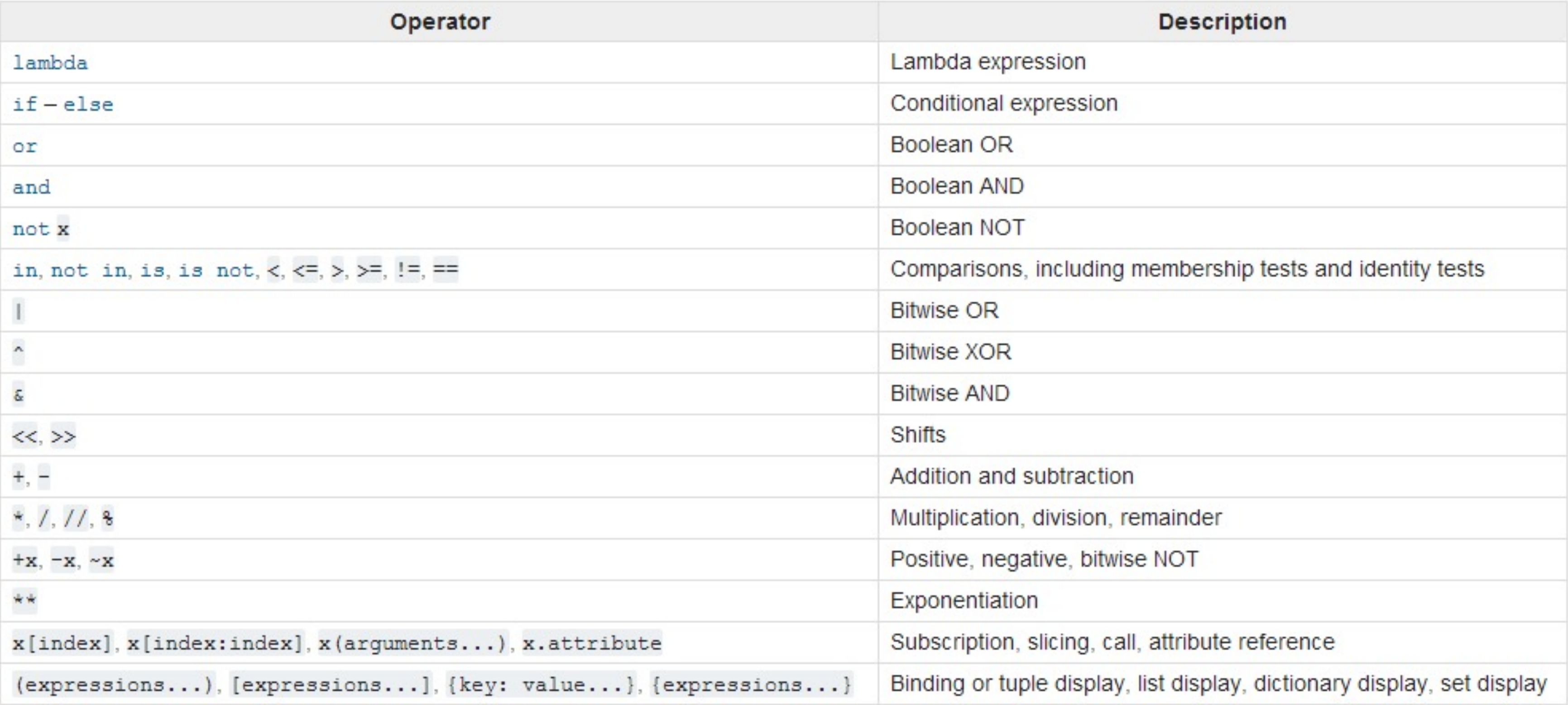
- Example

```
month = int(input("Enter a month number [1-12]: "))

is_summer = month >=6 and month <= 8
is_winter = month == 12 or month == 1 or month == 2
is_transition_season = not(is_winter or is_summer)

print("%s %s %s" % (is_summer, is_winter, is_transition_season))
```

booleans.py

Operator Precedence

- This table shows the precedence of all the operators in Python, from low precedence to high precedence



Operator	Description
lambda	Lambda expression
if – else	Conditional expression
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
|	Bitwise OR
^	Bitwise XOR
&	Bitwise AND
<<, >>	Shifts
+, -	Addition and subtraction
*, /, //, %	Multiplication, division, remainder
+x, -x, ~x	Positive, negative, bitwise NOT
**	Exponentiation
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
(expressions...), [expressions...], {key: value...}, {expressions...}	Binding or tuple display, list display, dictionary display, set display

Strings

- A string is an immutable sequence of Unicode characters
  - Can enclose in single quotes, double quotes, or triple quotes

```
str1 = "The computer says 'No' I'm afraid."
str2 = '<a href="www.bbc.co.uk">Click here for the BBC</a>'

str3 = """Birthday present ideas:
 - Bugatti Chiron
 - 4xHD OLED 64-inch TV
 - Socks"""

print("%s\n%s\n%s" % (str1, str2, str3))
```

strings.py

- The `String` class defines many methods
  - For details, see https://docs.python.org/3.8/library/string.html

- There's also excellent support for regular expressions
  - For details, see https://docs.python.org/3.8/library/re.html

Other Built-In Types

- Text sequence types
  - String - see previous slide
- Basic sequence types
  - List, tuple, and range
- Binary sequence types
  - `bytes`, `bytesarray`, and `memoryview`
- Set types
  - `set`, `frozenset`
- Mapping type
  - `dict`

Any Questions?