Data Processing and Analysis in Python
Lecture 2
Data Types and Expressions

University of Maryland

Robert H. Smith School of Business

Dr. Adam Lee

Data Types

- A **data type** consists of
  - a set of values, and
  - a set of operations that can be performed on those values
- A **literal** is the way a value of a data type looks to a programmer
- int and float are numeric data types
  - They represent numbers

Data Types

- Integers

```
>>> type(1)
<class 'int'>
>>> type(0b1011)
<class 'int'>
>>> type(0xabc)
<class 'int'>
```

- Floating-point numbers

```
>>> type(1.2)
<class 'float'>
```

- Complex numbers

```
>>> type(1 + 2j)
<class 'complex'>
```

- Strings

```
>>> type('1')
<class 'str'>
>>> type("1")
<class 'str'>
```

- Structures

```
>>> type((1, 2))
<class 'tuple'>
>>> type([1, 2])
<class 'list'>
>>> type({1, 2})
<class 'set'>
>>> type({1:2})
<class 'dict'>
```

String Literals

- A sequence of characters enclosed in 'char' or "char"
  - '' and "" represent the empty string
- Double-quoted strings are handy for composing strings that contain single quotation marks or apostrophes
- Use ''' and """ for multi-line paragraphs

```
>>>  print("""This very long sentence extends
all the way to the next line. """)
This very long sentence extends all the way to the
next line
>>> """This very long sentence extends
all the way to the next line. """
'This very long sentence extends\nall the way to
the next line.'
```

Escape Sequences

Escape Sequence	Meaning
\\	The \ character
\'	Single quotation mark
\"	Double quotation mark
\b	Backspace
\n	Newline
\t	Horizontal tab
\v	Vertical tab
\ooo	Character with octal value *ooo*
\x*hh*	Character with hex value *hh*

String Concatenation

- You can join two or more strings to form a new string using the concatenation operator +

```
>>> "Hi " + "there, " + "Ken!"
'Hi there, Ken!'
```

- The * operator allows you to build a string by repeating another string a given number of times

```
>>> "." * 10 + "Python"
'..........Python'
```

Variables

- Reserved words cannot be used as variable names
  - Examples: **if**, **def** and **import**
- Naming rules:
  - Name must begin with a letter or _
  - Name can contain any number of letters, digits, or _
  - Names are case sensitive
    - Example: **Weight** is different from **weight**
  - Camel or Hungarian casing
    - Example: **interestRate** or **fltInterestRate**
- All uppercase letters for symbolic constants
  - Examples: **TAX_RATE** and **STANDARD_DEDUCTION**

Assignment Statement

- Variables receive initial values and can be reset to new values with an assignment statement
  <variable name> = <expression>
- Subsequent uses of the variable name in expressions are known as variable references

```
>>> firstName = "Ken"
>>> lastName = "Lambert"
>>> fullName = firstName + " " + lastName
>>> fullName
'Ken Lambert'
>>> fullName = lastName + ", " + firstName
>>> fullName
'Lambert, Ken'
```

Program Comments

- A piece of program text that the computer ignores but that provides useful documentation to programmers

  # This is a comment on the code block as follows

- End-of-line comment might explain the purpose of a variable or the strategy used by a piece of code

  TAX_RATE = 0.06 # General sales tax rate in Maryland

Floating-Point Numbers

Decimal Notation	Scientific Notation	Meaning
3780.0	3.78e3	3.78×10^{3}
37.8	3.78e1	3.78×10^{1}
3.78	3.78e0	3.78×10^{0}
0.378	3.78e−1	3.78×10^{-1}
0.00378	3.78e−3	3.78×10^{-3}

- Real numbers have infinite precision, i.e. digits in the fractional part can continue forever
- Typical range: -10^{308} to 10^{308}
- Typical precision: 16 digits

Arithmetic Expressions

Operator	Meaning	Syntax
**	Exponentiation	a ** b
-	Negation	-a
*	Multiplication	a * b
/	Division	a / b
//	Quotient	a // b
%	Remainder or modulus	a % b
+	Addition	a + b
–	Subtraction	a – b

- Can use () to change the order of evaluation

Arithmetic Expressions

- When both operands of an expression are of the same numeric type, the resulting value is also of that type
- When each operand is of a different type, the resulting value is of the more general type

```
>>> 3 // 4
0
>>> 3 / 4.0
.75
```

- For multi-line expressions, use a \

```
>>> 3 + 4 * \
2 ** 5
131
```

Type Conversions

Conversion Function	Example	Value Returned

int(<a number or a string>)	int(3.77)
	int("33")
		3
		33

hex(<an integer>)	hex(10)	0xa

float(<a number or a string>)	float(22)
	float("3.14")
		22.0
		3.14

str(<any value>)	str(99)	'99'

Built-In Math Function	Example	Value Returned

round(<a floating number>)	round(3.4)
	round(3.5)
		3
		4

Character Sets

- In Python, character literals look just like string literals and are of the **string** type
  - They belong to several different character sets, among them the ASCII set and the Unicode set
- ASCII character set maps to a set of integers
- **ord()** and **chr()** functions convert characters to and from ASCII

```
>>> ord('A')
65
>>> hex(ord('A'))
0x41
>>> chr(65)
'A'
```

Modules and Functions

- Python includes many useful functions, which are organized in libraries of code called **module**s
- A **function** is chunk of code that can be called by name to perform a task
- Functions often take **arguments/parameters**, which may be optional or required
- When function completes its task, it may **return a value** back to the part of the program that called it
- To learn how to use a function, use the **help()** function:

```
>>> help(round)
Help on built-in function round in module __builtin__
...
```

The Math Module

- Functions like **abs()** and **round()** from the **__builtin__** module are always available to use
- Code in one module gains access to the code in another module by the process of **import**ing it.
- To use a resource from a module, write the module name as a qualifier, followed by **.** and the name of the resource
  - **math** module includes functions on basic mathematical operations

```
>>> import math
>>> math.pi
3.141592653589793l
>>> math.sqrt(2)
1.4142135623730951
```

The Math Module

- You can avoid the use of the qualifier with each reference by importing the individual resources

```
>>> from math import pi, sqrt
>>> print(pi, aqrt(2))
3.1415926535897931 1.4142135623730951
```

- You may import all of a module's resources to use without the qualifier

```
>>> from math import *
>>> print(pi, aqrt(2))
3.1415926535897931 1.4142135623730951
```

Program Format and Structure

- (optional) Start with comment with author's name, purpose of program, and other relevant information
  - i.e. **doc-string**
- Then, include statements that:
  - Import any modules needed by program
  - Initialize important variables, suitably commented
  - Prompt the user for input data and save the input data in variables
  - Process the inputs to produce the results
  - Display the results