CSC 223 - Advanced Scientific Programming

Python Built-In Types

Simple Values

Type	Example	Description
`int`	`x = 1`	integers
`float`	`x = 1.0`	floating point numbers
`complex`	`x = 1 + 2j`	complex numbers
`bool`	`x = True`	boolean: True/False values
`str`	`x = 'abc'`	string: characters or text
`NoneType`	`x = None`	null value

Integers

- Integer values are numbers without decimal points.

```
>>> x = 1
>>> type(x)
int
```

- Python integers are variable precision; computations do not overflow

Floating-Point Numbers

- Floating-point values can store fractional numbers
- Floating-point values can be defined in standard or exponential notation

```
x = 0.000005
y = 5e-6
```

- An integer can be converted to a float with the `float` constructor

```
float(1)
```

Complex Numbers

- Complex numbers have real and imaginary parts (both floating point values).
- Complex numbers can be created with the `complex` constuctor:

```
>>> complex(1, 2)
(1+2j)
```

- Or alternatively with the "j" suffix

```
>>> 1 + 2j
(1+2j)
```

String Type

- Strings in Python can be created with single or double quotes

```
message = "what do you like?"
response = 'spam'
```

- Python strings have useful functions and methods
- Examples:

```
>>> len(response)
4
>>> response.upper()
'SPAM'
>>> message[0] # zero-based indexing
'w'
```

Boolean Type

- The Boolean type has two possible values: `True` and `False`.
- Values of any other type can be converted into boolean values with the `bool` constructor.
- Examples:

```
>>> bool(123)
True
>>> bool(0)
False
>>> bool('')
False
```

None Type

- The `NoneType` has only a single possible value: `None`

```
>>> type(None)
NoneType
```

- A Python function that does not return a value returns `None`

Built-In Data Structures

Type	Example	Description
`list`	`[1, 2, 3]`	ordered collection
`tuple`	`(1, 2, 3)`	immutable ordered collection
`dict`	`{'a': 1, 'b': 2}`	unordered (key,value) mapping
`set`	`{1, 2, 3}`	unordered collection

Lists

- Lists are the basic ordered and mutable data collection
- Lists can be defined comma-separated values between square brackets

```
>>> L = [2, 3, 5, 7]
```

- Lists have many useful methods
- Examples:

```
>>> len(L)
4
>>> L.append(11)
[2, 3, 5, 7, 11]
```

List Indexing

- Elements of a list can be indexed for single values.
- Lists use zero based indexing

```
>>> L = [2, 3, 5, 7, 11]
>>> L[0]
2
```

- Lists can be indexed from the end with negative integers

```
>>> L[-1]
11
>>> L[-2]
7
```

List Slicing

- Elements of a list can be sliced for multiple values.
- List slicing syntax uses a colon to indicate the (inclusive) start point and the (exclusive) end point.

```
>>> L = [2, 3, 5, 7, 11]
>>> L[0:3]
[2, 3, 5]
```

- An optional third integer can be used to represent a step size

```
>>> L[::2]
[2, 5, 11]
```

Tuples

- Tuples are an immutable, ordered collection
- Immutable means that once a tuple is created it cannot be changed
- Tuple are defined with parentheses or using commas

```
>>> t1 = (1, 2, 3)
>>> t2 = 1, 2, 3
```

- Tuples can be indexed and sliced like lists

Dictionaries

- Dictionaries map keys to values
- Dictionaries are created by a comma separated list of `key:value` pairs between curly braces

```
>>> numbers = {'one': 1, 'two': 2}
```

- Items are accessed using the key

```
>>> numbers['two']
2
```

Sets

- Sets are unordered collections of unique items
- Sets are defined by a comma separated list of values between curly braces

```
>>> primes = {2, 3, 5, 7}
>>> odds = {1, 3, 5, 7, 9}
```

- Sets support mathematical set operations
- Example

```
>>> primes | odds
{1, 2, 3, 5, 7, 9}
>>> primes.union(odds)
{1, 2, 3, 5, 7, 9}
```