Argonne NATIONAL LABORATORY

Learning Python

Getting results for beamlines and scientific programming

1. Basic Python: variables and common data types

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

U.S. DEPARTMENT OF ENERGY | Office of Science

Outline of topics to be covered

1. Variables
2. Seeing the contents/type
3. Primitive data types: string, Boolean, int, float, complex
   - String methods
4. None type
5. Type conversion

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

Variables in Python

- A variable is simply a name that one creates to store something
- Variable names in Python follow the expected rules:
  - Upper and lower case letters and numbers are allowed. No symbols, except underscore (_)
  - Variable names must start with a letter
  - Variable names are case sensitive (MyVar and myvar are different)
  - The length of variable names is limited only by sanity
  - You cannot use special Python keywords (such as if or else) as variable names (see Downey, section 2.3 for list)
- A value is associated with a variable using the "=" sign
  - A = "this is a string"
  - A = 1.1
  - A = 1
- No declaration is needed for what type of value is stored
  - The contents of a variable has a type, but there is no type associated with the variable name

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

Seeing the contents of a variable

- In an interactive shell, simply typing a variable name will cause the value to be shown
- The print command will also show the contents of a variable
- Note: in Python 3.0, the print command changes from
  - print arg

  to
  - print(arg)

```
toby$ python
>>> a = 1.1
>>> a
1.1000000000000001
>>> print a
1.1
>>>
```

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

Seeing the type for contents of a variable

- Variables themselves do not have a type (unlike C and Fortran) but the value associated with them does. Use type( ) to see the type for a value:

```
>>> A = "test"
>>> B = 1
>>> C = 1.1
>>> type(A)
<type 'str'>
>>> type(B)
<type 'int'>
>>> type(C)
<type 'float'>
```

- Note the difference between a float and a string (quotes):

```
>>> D = '1.00'
>>> type(D)
<type 'str'>
```


Primitive data types in Python

Python has many data types, some simple and some sophisticated

- Common simple data types include:
- Boolean: (no quotes; note 1st letter only must be upper case)

```
A = True
A = False
```

- Integer:

```
A = 1
```

- Float:

```
A = 1.0
```

- String:

```
A = "this is a string"
```

- Complex:

```
A = complex(1,2)
```

Note that Python has many other less commonly used primitive data types, including decimal (fixed precision), fractions (rational numbers), long (infinite length integer)

Note that several delimiters can be used to define strings

- Single quotes ('), double quotes (" – shift from ' not two ' ' chars)

```
>>> "a string"
'a string'
>>> 'another string'
'another string'
>>> 'quote with " a quote'
'quote with " a quote'
```

- Quotes tripled (" " ") or (' ' ') indicate a string that can span lines

```
>>> """
... multi
... line
... string
... """
'\nmulti\nline\nstring\n'
>>>
```

  - Note that \n here represents a new line (carriage return)

Long character strings

- Character strings get appended if no operator is supplied

```
>>> a = 'a ' 'string' " in four" ' parts'
>>> a
'a string in four parts'
```

- Note that commas make this into a tuple not a long string:

```
>>> a = 'a ', 'string', " in four", ' parts'
>>> a
('a ', 'string', ' in four', ' parts')
```

- This is helpful to make long strings that do not contain new lines. Note that parenthesis here tell Python that the line did not end.

```
>>> a = ('a ' 'string' " in four"
...       " parts"
... )
>>> a
'a string in four parts'
```

Strings come with handy routines

- The following functions are "attached to" strings:

  capitalize, center, count, decode, encode, endswith, expandtabs, find, format, index, isalnum, isalpha, isdigit, islower, isspace, istitle, isupper, join, ljust, lower, lstrip, partition, replace, rfind, rindex, rjust, rpartition, rsplit, rstrip, split, splitlines, startswith, strip, swapcase, title, translate, upper, zfill

- Note how the routines associated with strings are used:

```
>>> s = "this is a string"
>>> s.upper()
'THIS IS A STRING'
>>> s.capitalize()
'This is a string'
>>> s.startswith('this')
True
>>> s.startswith('is')
False
>>> s.find('is')
2
```

- One can even use the plus (+) operator on strings:

```
>>> s.capitalize() + ". " + "And another one."
'This is a string. And another one.'
```


One unusual data type: None

- Python defines a data value of None (note no quotes!)
  - A = None

  This is used frequently where a null value makes sense

Conversion between data types

- One can force Python to use a particular data type by calling the type as a function:

```
>>> float(3)
3.0
>>> int(3.1)
3

>>> int(complex(3,2))
TypeError: can't convert complex to int

>>> complex(3,2).real
3.0
>>> int(complex(3,2).real)
3
```


Homework

- What is the difference between these three statements?

```
A = 1
A = 1.0
A = "1"
```

- Is there a difference between any of these:

```
B = 'a string'
B = "a string"
B = """a string"""
```

- Where can you find more information on the routines associated with strings listed on slide #9?