SNU ACM

Introduction to Python

Contents of this talk

- Basics
- Data types
- Operators
- Functions
- Control Statements

Basics

The Python Interpreter

```
$ python3

Python 3.5.1+ (default, Mar 30 2016, 22:46:26)
[GCC 5.3.1 20160330] on linux
Type "help", "copyright", "credits" or "license" for
more information.
>>> x = 5
>>> x
5
>>> exit()
```

Running a Python Program

- Type the program and save it in a file (say, `prog1.py`):

```
x = 5
print(x)
```

- Open `Terminal` or `cmd` and go to the directory with the above file, then type

```
python3 prog1.py
```

- You should see the output as `5`.

Enough to Understand the Code

- Assignment uses = and comparison uses ==
- For numbers + - * / % are as expected.
  - Special use of + for string concatenation.
- Logical operators are words (and, or, not) not symbols.

Enough to Understand the Code

- The basic printing command is print.
- The first assignment to a variable creates it.
  - Variable types don't need to be declared.
  - Python figures out the variable types on its own.

Whitespaces

- Whitespace is meaningful in Python: especially indentation and placement of newlines.
- No braces { } to mark blocks of code in Python. Use consistent indentation instead.
  - The first line with *less* indentation is outside of the block.
  - The first line with *more* indentation starts a nested block
- Often a colon appears at the start of a new block. (E.g. for function and class definitions.)

Comments

- Start comments with # and the rest of line will be ignored.

For example,

```
# This is a comment
# and it will be ignored.
print("Hello World!")
```

Assignment

- Variables can be assigned using the = operator.

```
>>> x = 5
>>> x
5
```

- Multiple assignments are also possible.

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
```

Datatypes

Basic Datatypes

1. Integers (default for numbers)

```
z = 5 / 2       # Answer is 2, integer division.
```

2. Floats

```
z = 5.0/2.0   # Answer is 2.5
```

Basic Datatypes

3. Strings
   - Can use “” or ‘’ to specify.

```
s1 = "my string1"
s2 = 'my string2'
```

   - Use triple double-quotes for multi-line strings or strings than contain both ‘ and “ inside of them.

```
s1 = """a'b"c"""
s2 = """this is
a multi-line string"""
```

Tuple

A simple **immutable** ordered sequence of items.

```
t = (1,2,3,'a', 'hello')
print( t )              # (1, 2, 3, 'a', 'hello')
print( t[0] )           # 1
print( t[-1] )          # 'hello'
print( type( t[0] ) )   # <class 'int'>
print( type( t[4] ) )   # <class 'str'>
print( type( t ) )      # <class 'tuple'>
```

```
t1 = (1, 2, 3)
t2 = (4.0, 'hello')
t = t1 + t2
print( t )              # (1, 2, 3, 4.0, 'hello')
```

List

Mutable ordered sequence of items of mixed types

```
lst = [1,2,3,'a', 'hello']
print( lst )                # [1, 2, 3, 'a', 'hello']
print( lst[0] )             # 1
print( lst[-1] )            # 'hello'
print( type( tst ) )        # <class 'list'>
```

Append items to list:

```
>>> lst = [1, 'abc']
>>> lst.append( 'xyz' )
>>> print( lst )
[1, 'abc', 'xyz']
```

Dictionaries

- Dictionaries store a mapping between a set of keys and a set of values.
- Keys can be any immutable type.
- Values can be any type
- A single dictionary can store values of different types
- You can define, modify, view, lookup, and delete the key-value pairs in the dictionary.

Dictionaries

```
>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user']
'bozo'
>>> d['pswd']
1234
```

```
>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user'] = 'clown'
>>> d
{'user':'clown', 'pswd':1234}
>>> d['id'] = 45
>>> d
{'user':'clown', 'id':45, 'pswd':1234}
```

Dictionaries

```
>>> d = {'user':'bozo', 'p':1234, 'i':34}
>>> del d['user']              # Remove one.
>>> d
{'p':1234, 'i':34}
>>> d.clear()                  # Remove all.
>>> d
{}
```

Dictionaries

```
>>> d = {'user':'bozo', 'p':1234, 'i':34}
>>> d.keys()                # List of keys.
['user', 'p', 'i']
>>> d.values()              # List of values.
['bozo', 1234, 34]
>>> d.items()         # List of item tuples.
[('user','bozo'), ('p',1234), ('i',34)]
```

Slicing

Used to obtain a subset of a string, list, or tuple.

```
>>> lst = [1, 2, 3, 4, 5]
>>> len( lst )
5
>>> lst[1:5]  # Elements of lst with index 1 to 4
[2, 3, 4, 5]
>>> lst[1:]   # Elements with index 1 to the last element
[2, 3, 4, 5]
>>> lst[:4]   # Elements from 0th element to the 3rd
[1, 2, 3, 4]
>>> lst[2:3]  # 2nd element to 2nd element
[3]
>>> lst[:]    # All elements
[1, 2, 3, 4, 5]
```

Operators

- in operator
- + operator
- * operator

in operator

Boolean test whether a value is inside a container:

```
>>> t = [1, 2, 4, 5]
>>> 3 in t
False
>>> 4 in t
True
>>> 4 not in t
False
```

For strings, tests for substrings

```
>>> a = 'abcde'
>>> 'c' in a
True
>>> 'cd' in a
True
>>> 'ac' in a
False
```

+ operator

The + operator produces a new tuple, list, or string whose value is the concatenation of its arguments.

```
>>> (1, 2, 3) + (4, 5, 6)
(1, 2, 3, 4, 5, 6)
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
>>> "Hello" + " " + "World"
'Hello World'
```

* operator

The * operator produces a new tuple, list, or string that “repeats” the original content.

```
>>> (1, 2, 3) * 3
(1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> "Hello" * 3
'HelloHelloHello'
```

Functions

Functions

- def creates a function and assigns it a name.
- return sends a result back to the caller.

```
def <name>(arg1, arg2, ..., argN):
    <statements>
    return <value>
```

e.g.

```
def times(x,y):
    return x*y
```

Optional arguments in a function

Can define defaults for arguments that need not be passed.

```
def func(a, b, c=10, d=100):
    print a, b, c, d
>>> func(1,2)
1 2 10 100
>>> func(1,2,3)
1 2 3 100
>>> func(1,2,3,4)
1,2,3,4
```

Functions (cont.)

- All functions in Python have a return value
  - even if no return line inside the code.
- Functions without a return return the special value None.
- Functions can be used as any other data type. They can be:
  - Arguments to function
  - Return values of functions
  - Assigned to variables
  - Parts of tuples, lists, etc

Control Statements

if, elif, else statements

Used to control the flow of the program.

```
if x == 3:
    print "X equals 3."
elif x == 2:
    print "X equals 2."
else:
    print "X equals something else."
print "This is outside the ‘if’."
```

for loop

```
for x in range(10):
    print "Still in the loop."
    if x == 8:
        break
print "Outside of the loop."
```

while loop

```
x = 0
while x < 10:
    print "Still in the loop."
    if x == 8:
        break
    x = x + 1
print "Outside of the loop."
```

User input

Input can be taken from the user using the `input()` function.

```
>>> x = input("Enter a number: ")
Enter a number: 5
>>> print(x)
'5'
>>> print( type(x) )
<class 'str'>
```

Note: You can use the `raw_input()` function instead if you're using Python2.x

Thank You 😀