NUS | Computing

National University of Singapore

IT5001 Software Development Fundamentals

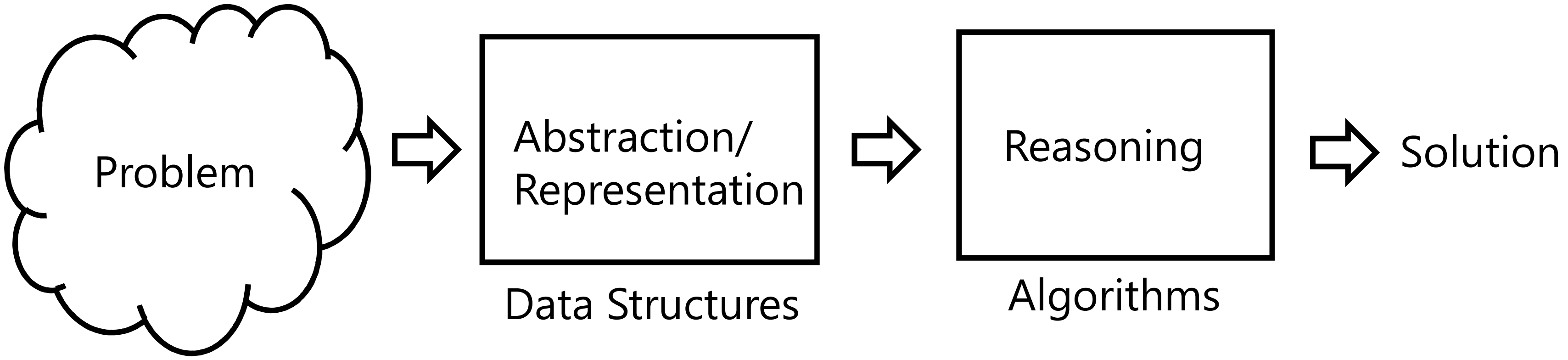
1. Introduction

Rajendra Prasad Sirigina

August-2022

Software Development

Steps involved in problem-solving:

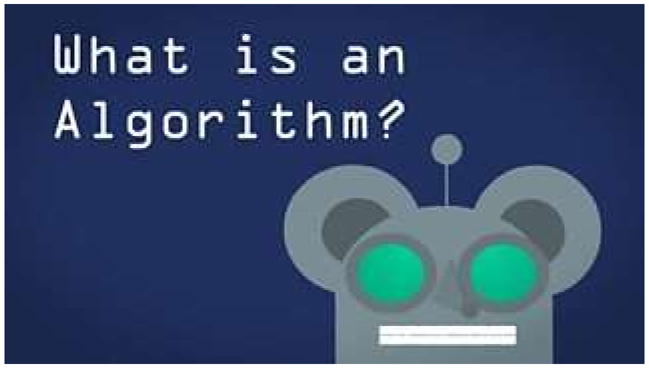


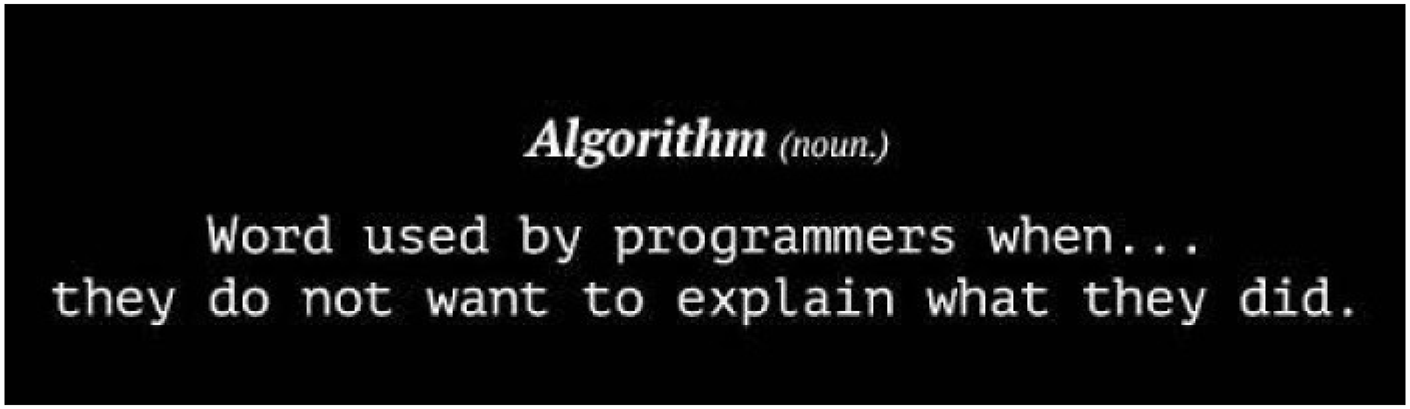
Programming languages provide tools to:

1. build data structures
2. do reasoning

Representation

- Numbers
- Strings
- Arrays
- Multi-dimensional Arrays
- Graphs and Trees




Algorithm *(noun.)*

Word used by programmers when...
they do not want to explain what they did.

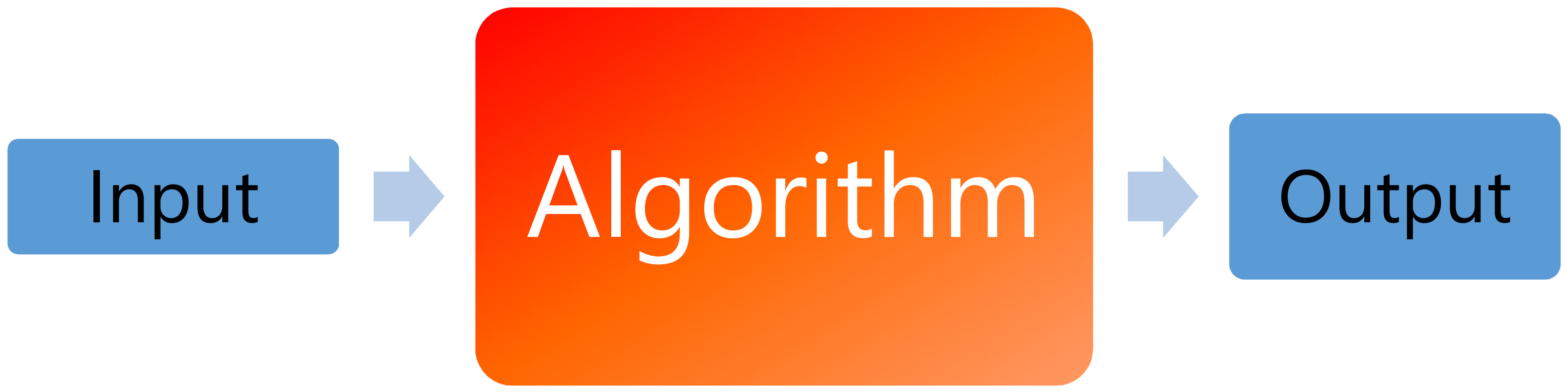
Algorithms

- Named for al-Khwārizmī (780-850)
    - Persian mathematician

- Many ancient algorithms
    - Multiplication: Rhind Papyrus
        - Babylon and Egypt: ~1800BC
    - Euclidean Algorithm: Elements
        - Greece: ~300BC
    - Sieve of Eratosthenes
        - Greece: ~200BC

Algorithm

- An **algorithm** is a well-defined computational procedure consisting of *a set of instructions*, that takes some value or set of values as *input*, and produces some value or set of values as *output*.

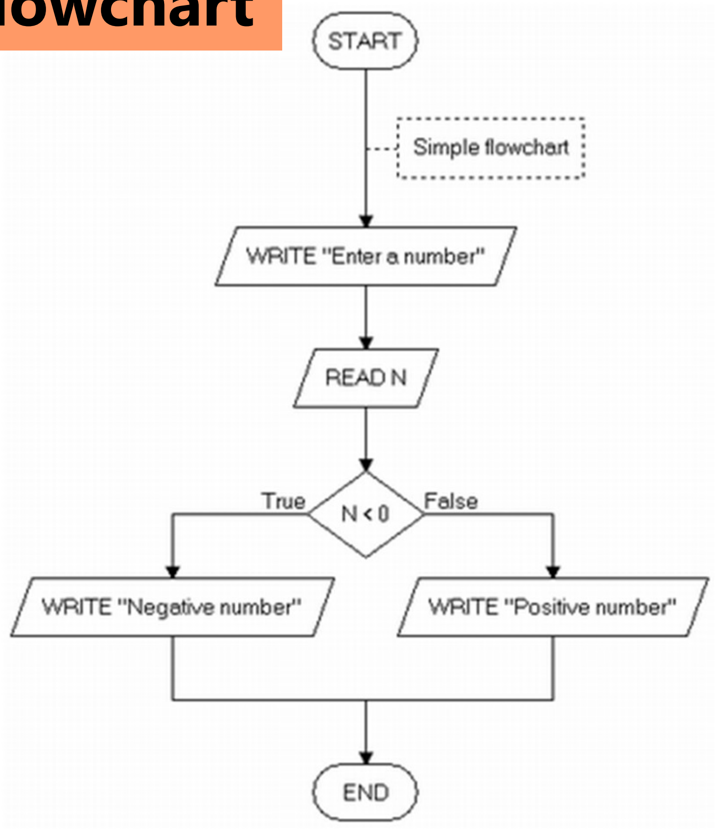


'Algorithm' stems from 'Algoritmi', the Latin form of al-Khwārizmī, a Persian mathematician, astronomer and geographer.
Source: http://en.wikipedia.org/wiki/Algorithm

Algorithm

- Ways of representing an algorithm:

Flowchart



https://en.wikipedia.org/wiki/Flowchart

Pseudocode

```
get a number

read the number and store it in N

if N is less than zero
        print negative number
else
        print positive number
end If
```

Algorithm Vs Program

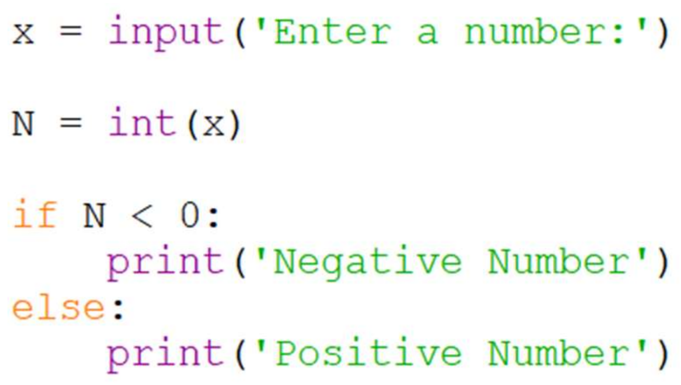
Algorithm

- Ideas

```
get a number

read the number and store it in N

if N is less than zero
        print positive number
else
        print negative number
end If
```

Program

- The final code on a machine



```python
x = input('Enter a number:')

N = int(x)

if N < 0:
    print('Negative Number')
else:
    print('Positive Number')
```

Writing a Program

- Requires
  - ➢ Understanding of language issues
    - o Syntax and Semantics
  - ➢ Data Structures
    - o Representation of the problem
  - ➢ Reasoning ability
    - o Algorithms

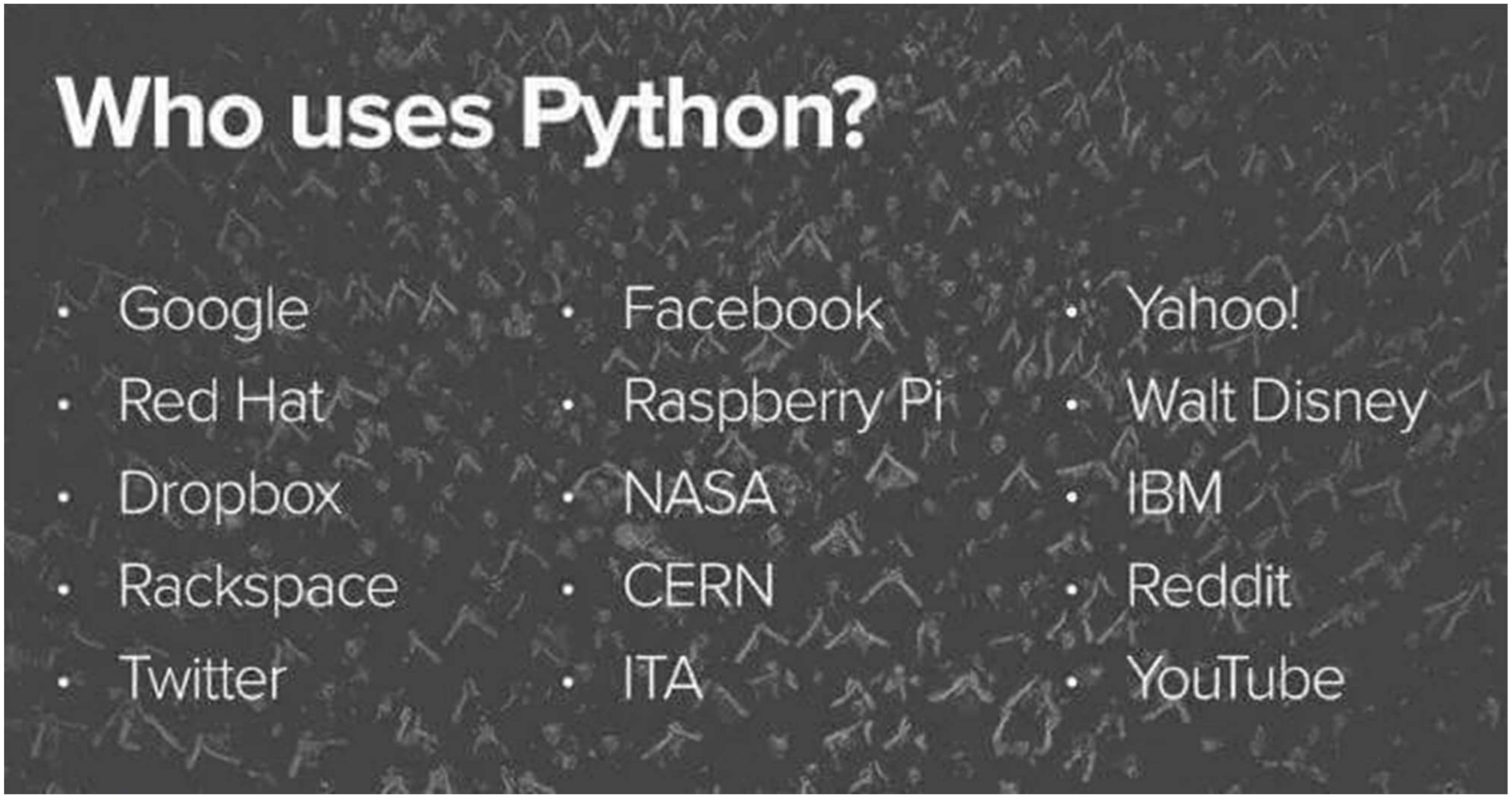
An overview of

python™

Why are we learning Python?

- Clear and readable syntax
- Intuitive
- Natural expression
- Powerful
- Popular & Relevant

- Example: Paypal
  - ➢ ASF XML Serialization
    - o C++
      - 1580 lines
    - o Python
      - 130 lines



Who uses Python?

- Google
- Red Hat
- Dropbox
- Rackspace
- Twitter
- Facebook
- Raspberry Pi
- NASA
- CERN
- ITA
- Yahoo!
- Walt Disney
- IBM
- Reddit
- YouTube

Python Program without Learning

```
a = 1
b = 2
c = a + b
if c < 0:
      print('Yes')
else:
      print('No')
```

Intuitive!

Pseudo Code to Program

Algorithm

```
get a number

read the number and store it in N

if N is less than zero
        print positive number
else
        print negative number
end If
```

Program

```
x = input('Enter a number:')

N = int(x)

if N < 0:
    print('Negative Number')
else:
    print('Positive Number')
```

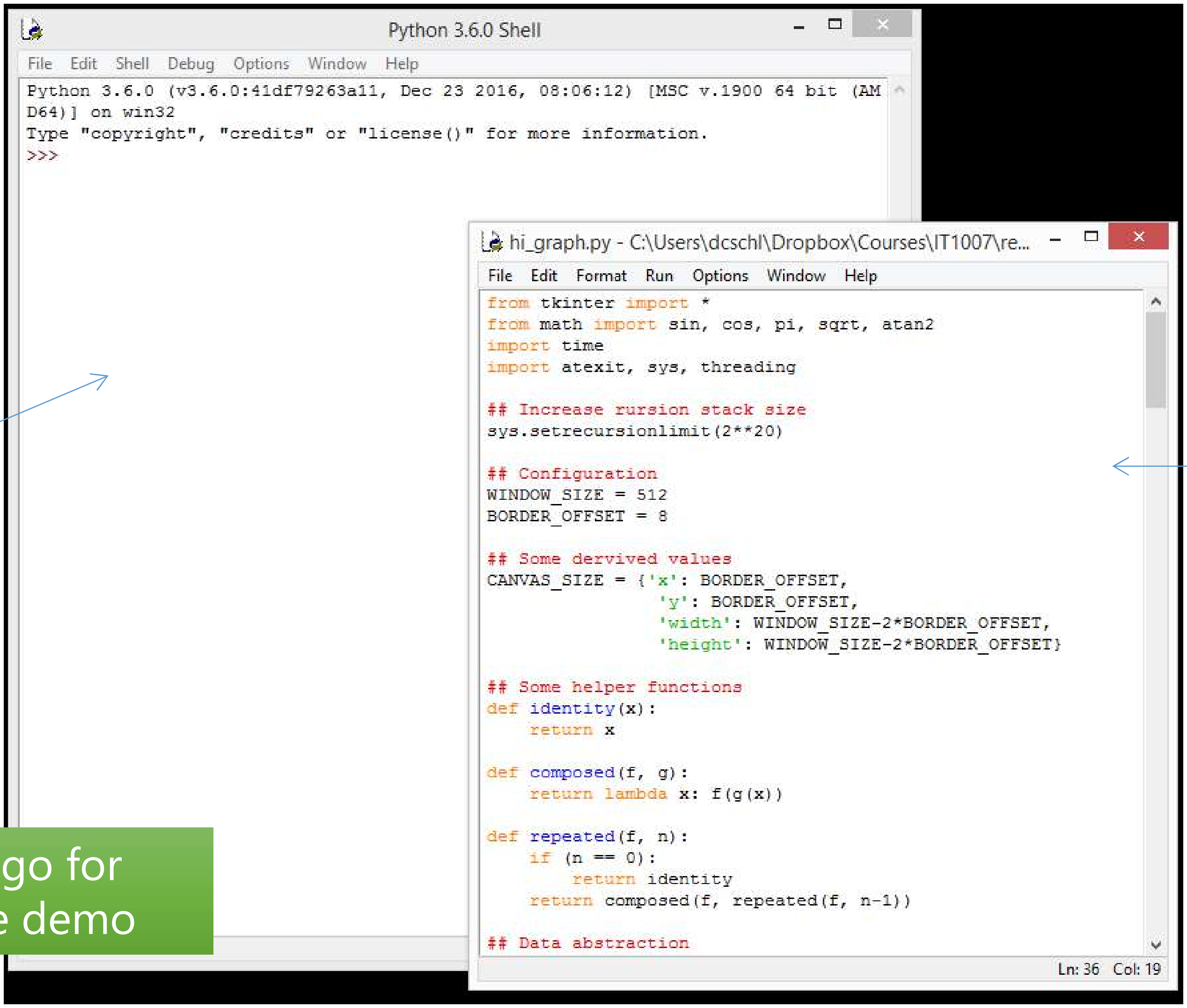
Automatic Vs. Manual Transmission:
Which is the best choice for you?

The Environment: IDLE

- **IDLE** as an IDE
  - ➢ IDLE:
    - o Integrated development and learning environment
  - ➢ IDE:
    - o Integrated development environment
      - Edit, run and debug
- Other tools
  - ➢ Jupyter notebook
  - ➢ PyCharm
  - ➢ Spyder
  - ➢ Visual Studio Code, etc.

A Screenshot of IDLE

Console
- Input
- output


Python 3.6.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

hi_graph.py - C:\Users\dcschl\Dropbox\Courses\IT1007\re...
File Edit Format Run Options Window Help
Ln: 36 Col: 19

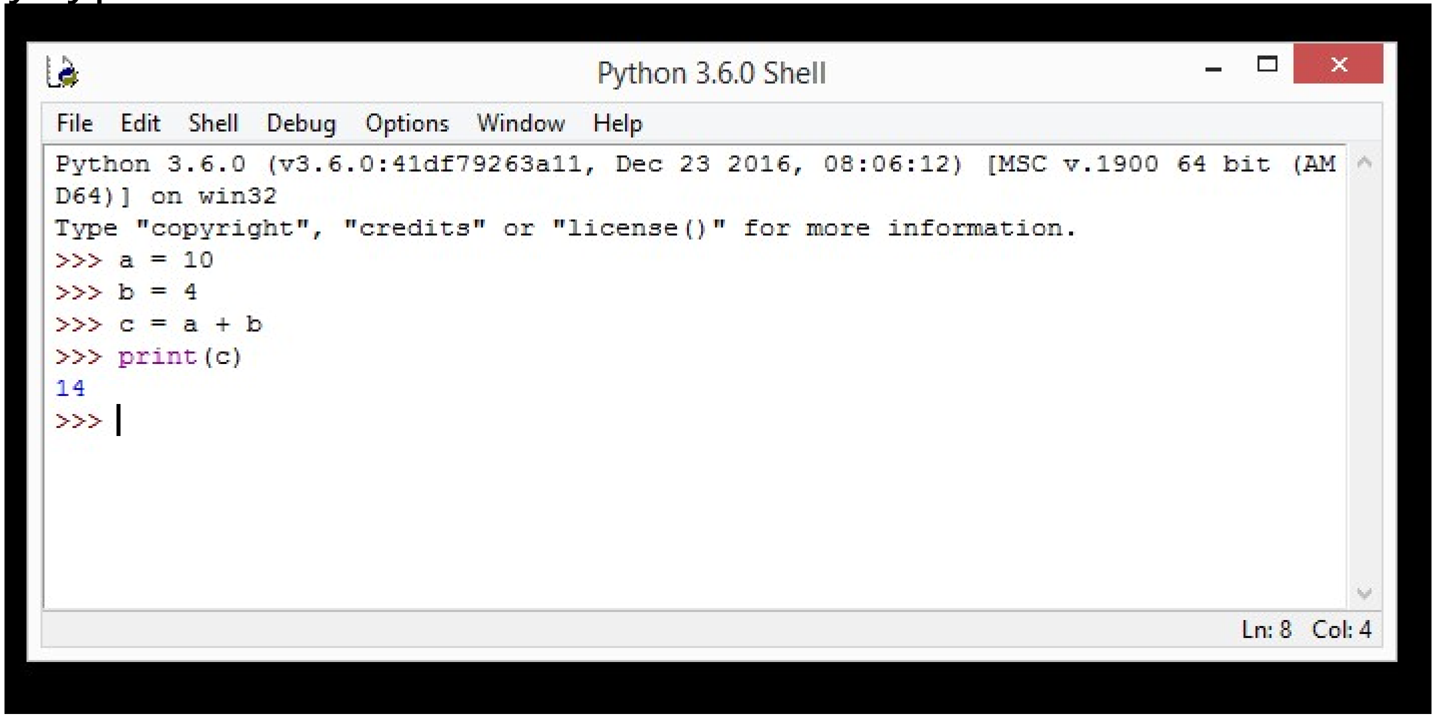
```
from tkinter import *
from math import sin, cos, pi, sqrt, atan2
import time
import atexit, sys, threading

## Increase rursion stack size
sys.setrecursionlimit(2**20)

## Configuration
WINDOW_SIZE = 512
BORDER_OFFSET = 8

## Some derived values
CANVAS_SIZE = {'x': BORDER_OFFSET,
                'y': BORDER_OFFSET,
                'width': WINDOW_SIZE-2*BORDER_OFFSET,
                'height': WINDOW_SIZE-2*BORDER_OFFSET}

## Some helper functions
def identity(x):
    return x

def composed(f, g):
    return lambda x: f(g(x))

def repeated(f, n):
    if (n == 0):
        return identity
    return composed(f, repeated(f, n-1))

## Data abstraction
```

Editor and Your program

Let's go for some demo

You can

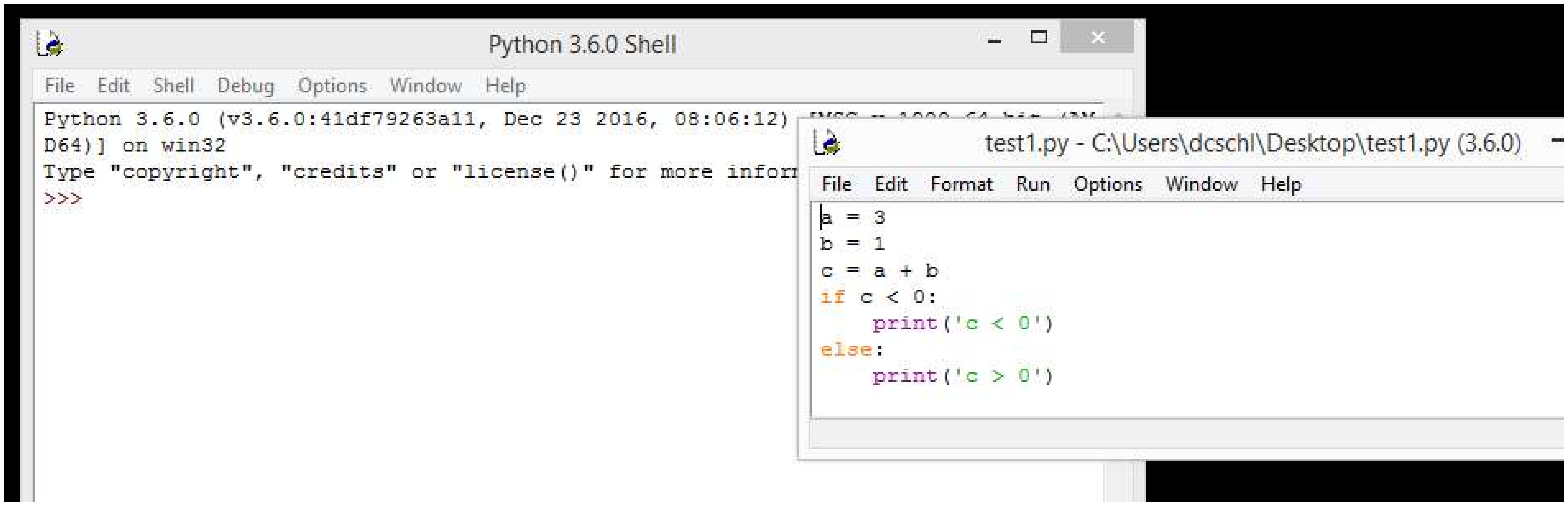
- Directly type into the console



```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a = 10
>>> b = 4
>>> c = a + b
>>> print(c)
14
>>> 
```

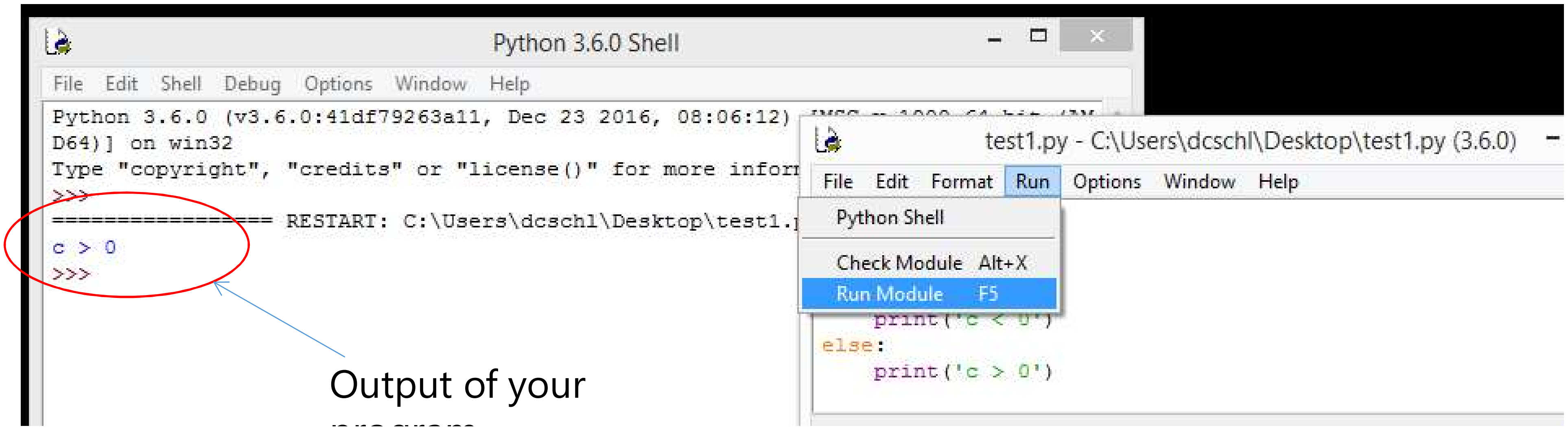
- In which, we seldom do this

Or Run a file


Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
test1.py - C:\Users\dcschl\Desktop\test1.py (3.6.0)
File Edit Format Run Options Window Help

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12)
D64)] on win32
Type "copyright", "credits" or "license()" for more infor
>>>
```

```
a = 3
b = 1
c = a + b
if c < 0:
    print('c < 0')
else:
    print('c > 0')
```

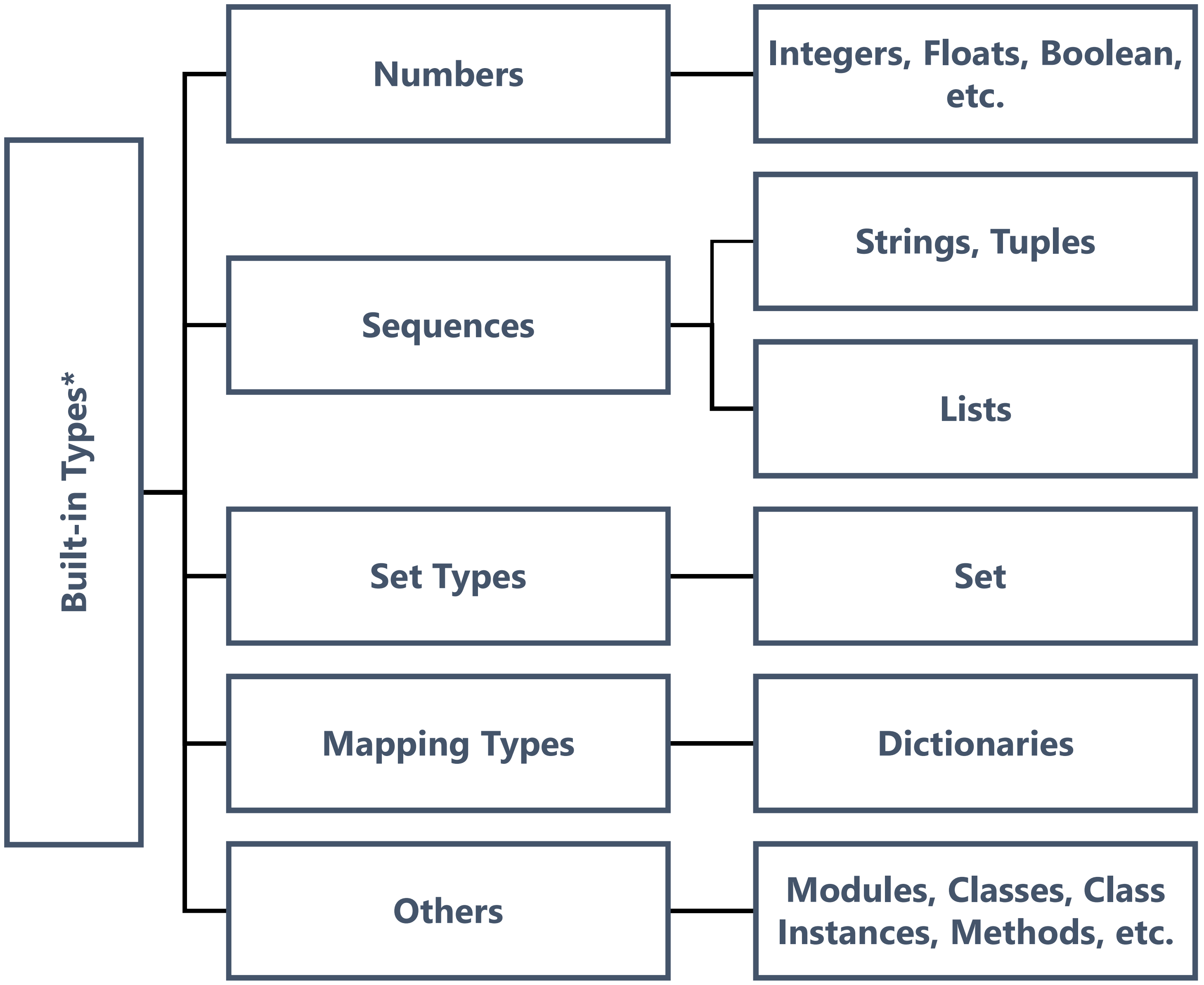

Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
test1.py - C:\Users\dcschl\Desktop\test1.py (3.6.0)
File Edit Format Run Options Window Help
Python Shell
Check Module Alt+X
Run Module F5

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12)
D64)] on win32
Type "copyright", "credits" or "license()" for more infor
>>>
================= RESTART: C:\Users\dcschl\Desktop\test1.
c > 0
>>>
```

```
    print('c < 0')
else:
    print('c > 0')
```

Output of your

Representation



*List is incomplete

Built-in Types

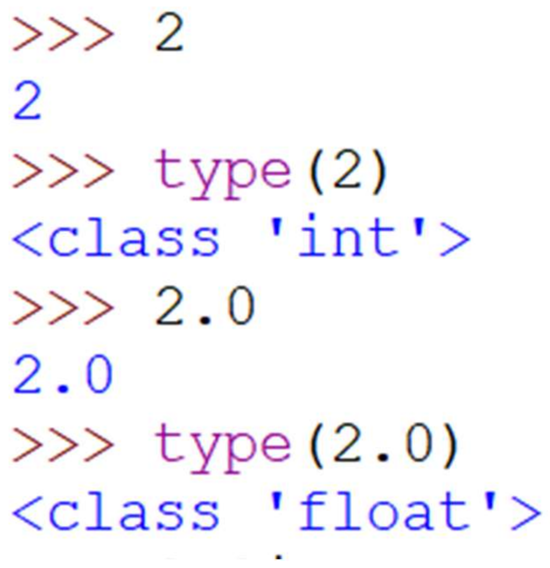
Type	Description	Immutable?
int	Integer	Yes
float	Floating-point number	Yes
bool	Boolean value	Yes
string	Character String	Yes
list	Sequence of objects	No
tuple	Sequence of objects	Yes
set	Unordered set of distinct objects	No
dict	Associative Mapping (dictionary)	No

Primitive types

Immutable: Cannot modify

Numbers: Numeric Types

- Integers: *int*
- Floats: *float*
  - ➢ Stores real numbers as binary fractions
  - ➢ 64-bit double precision*



```
>>> 2
2
>>> type(2)
<class 'int'>
>>> 2.0
2.0
>>> type(2.0)
<class 'float'>
```

- Self Exercise:
  - ➢ Convert the decimal numbers 0.375 and 0.1 to binary. What do you learn from the conversion?

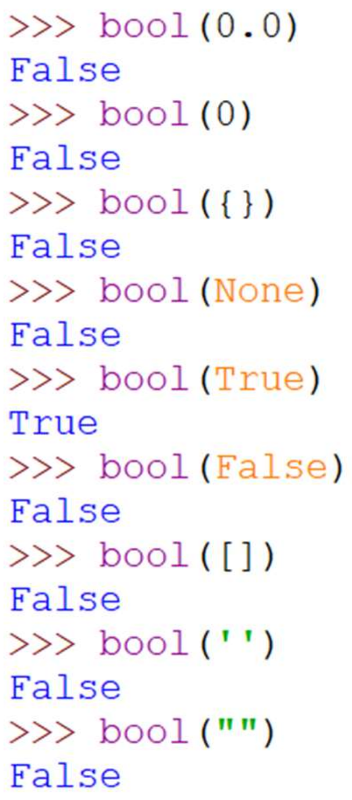
https://en.wikipedia.org/wiki/Double-precision_floating-point_format
https://docs.python.org/3/tutorial/floatingpoint.html#tut-fp-issues

Boolean Type

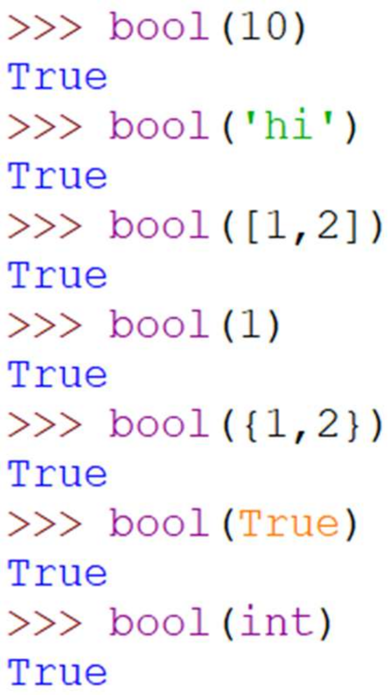
- Following are evaluated to False
  - ➢ False : Keyword
  - ➢ None : Keyword
  - ➢ 0, 0.0, 0j : Value Zero (*int, float, complex*)
  - ➢ " " : Empty String
  - ➢ [ ] : Empty List
  - ➢ { } : Empty Dictionary
  - ➢ range(0) : Iterator
  - ➢ set() : Empty set

Will learn them in subsequent weeks

- Rest are evaluated to True



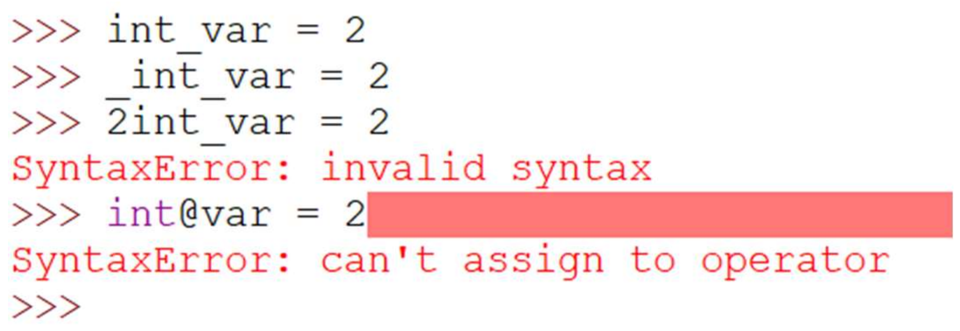
```
>>> bool(0.0)
False
>>> bool(0)
False
>>> bool({})
False
>>> bool(None)
False
>>> bool(True)
True
>>> bool(False)
False
>>> bool([])
False
>>> bool('')
False
>>> bool("")
False
```



```
>>> bool(10)
True
>>> bool('hi')
True
>>> bool([1,2])
True
>>> bool(1)
True
>>> bool({1,2})
True
>>> bool(True)
True
>>> bool(int)
True
```

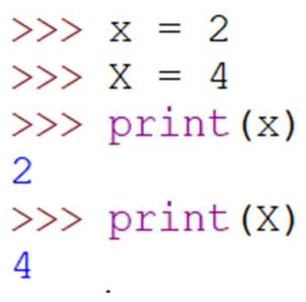
Identifiers

- User-defined names for objects
  - Can enhance readability
- Rules
  - First character should be an alphabet or underscore (_)
  - Other characters can be numbers and underscore
  - Special characters not allowed
  - Names are case sensitive



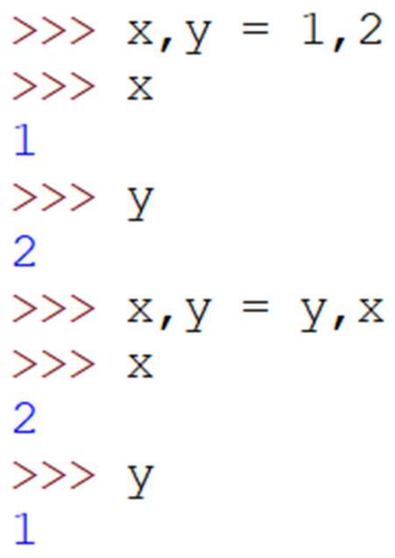
```
>>> int_var = 2
>>> _int_var = 2
>>> 2int_var = 2
SyntaxError: invalid syntax
>>> int@var = 2
SyntaxError: can't assign to operator
>>>
```

assignment operation

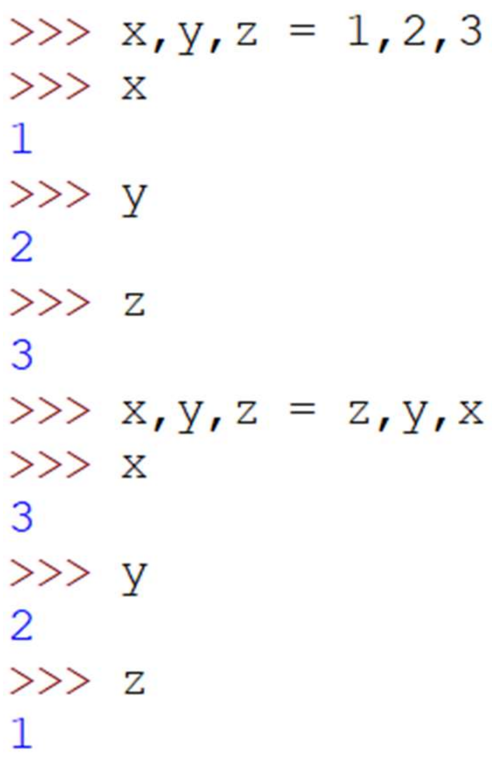


```
>>> x = 2
>>> X = 4
>>> print(x)
2
>>> print(X)
4
```

Multiple Assignments



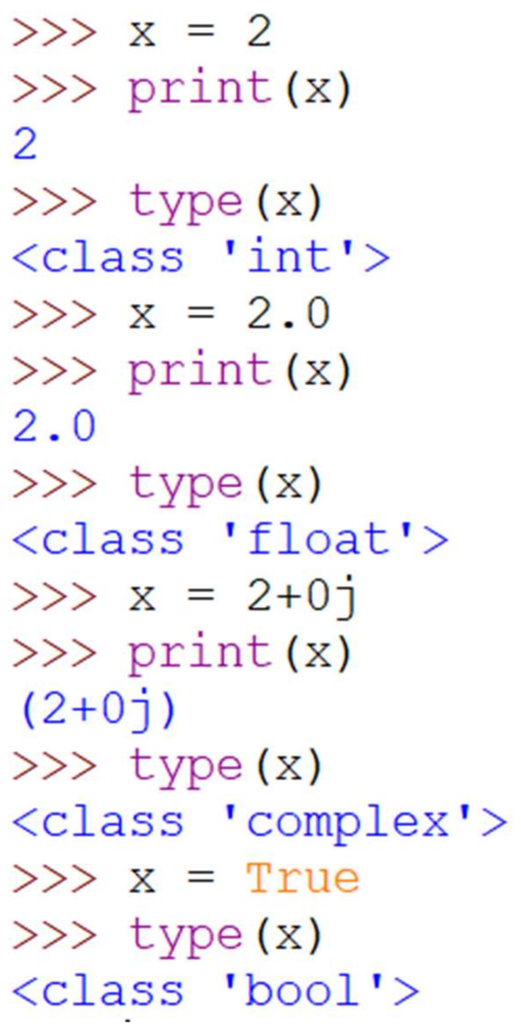
```
>>> x,y = 1,2
>>> x
1
>>> y
2
>>> x,y = y,x
>>> x
2
>>> y
1
```



```
>>> x,y,z = 1,2,3
>>> x
1
>>> y
2
>>> z
3
>>> x,y,z = z,y,x
>>> x
3
>>> y
2
>>> z
1
```

Python is Dynamically Typed

- No need to declare object type
- Interpreter automatically recognizes the type

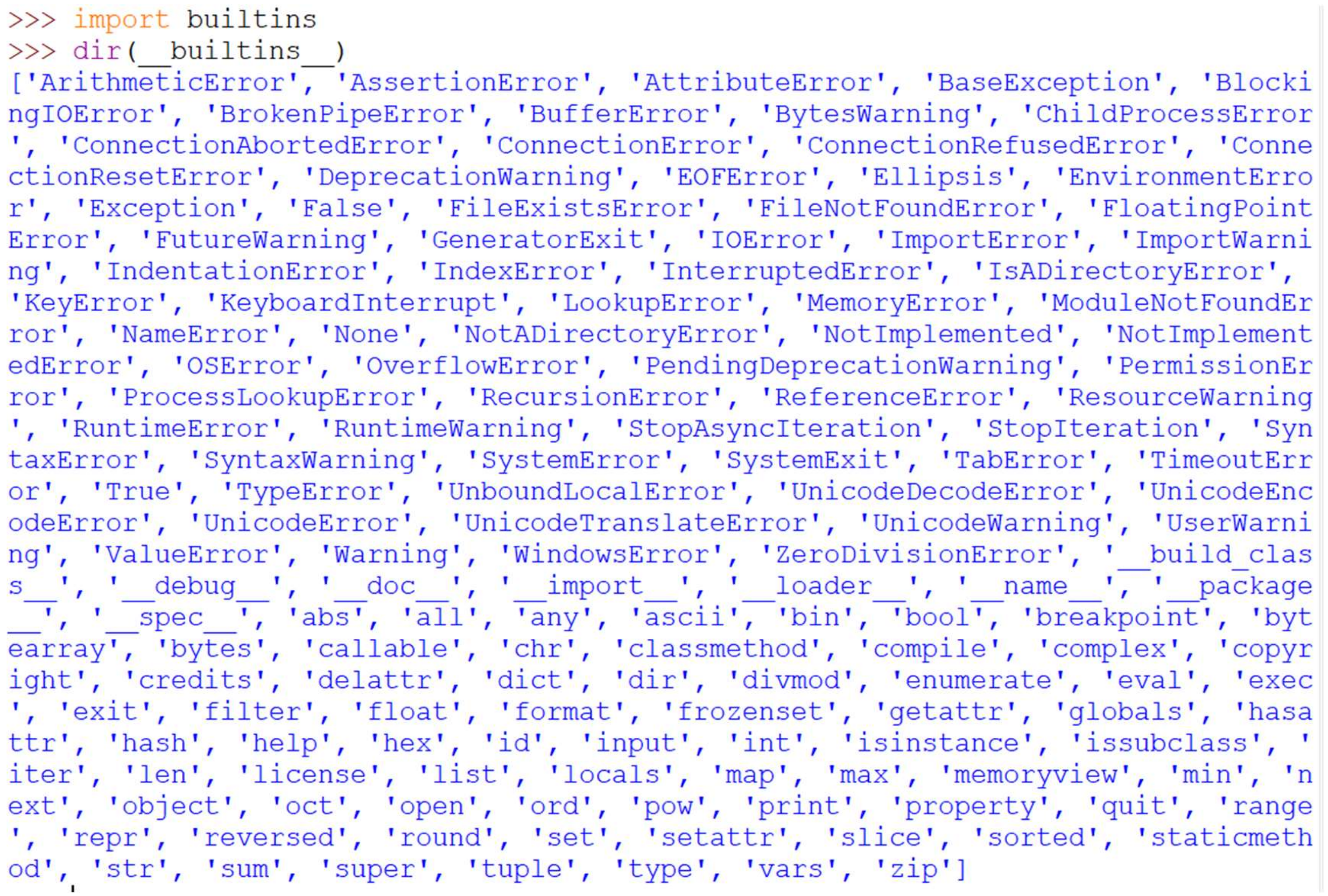


```
>>> x = 2
>>> print(x)
2
>>> type(x)
<class 'int'>
>>> x = 2.0
>>> print(x)
2.0
>>> type(x)
<class 'float'>
>>> x = 2+0j
>>> print(x)
(2+0j)
>>> type(x)
<class 'complex'>
>>> x = True
>>> type(x)
<class 'bool'>
```

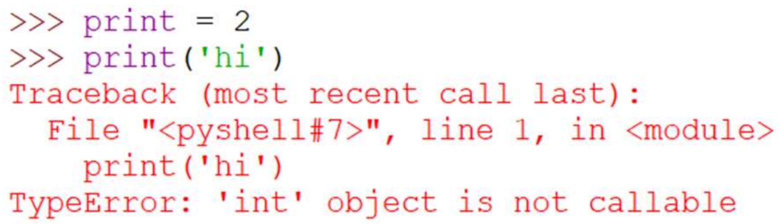
- Keywords cannot be used as identifiers

- Builtins can be used as variables
  - but don’t do it

builtins



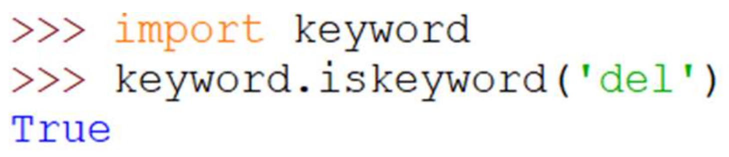
```
>>> import builtins
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'Blocki
ngIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError
', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'Conne
ctionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentErro
r', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPoint
Error', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarni
ng', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError',
'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundEr
ror', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplement
edError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionEr
ror', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning
', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'Syn
taxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutErr
or', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEnc
odeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarni
ng', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__build_clas
s__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package
__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'byt
earray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyr
ight', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec
', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasa
ttr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', '
iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'n
ext', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range
', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmeth
od', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```



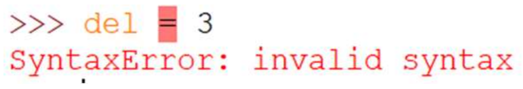
```
>>> print = 2
>>> print('hi')
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print('hi')
TypeError: 'int' object is not callable
```

Keyword Types

Type	Example
Value Keywords	*True, False, None*
Operator Keywords	*and, or, not, in, is*
Control Flow Keywords	*if, else, elif*
Iteration Keywords	*for, while, break, continue, else*
Structure Keywords	*def, class, with, as, pass, lambda*
Returning Keywords	*return, yield*
Import Keywords	*import, from, as*
Exception-handling Keywords	*try, except, raise, finally, else, assert*
Asynchronous Programming Keywords	*async, await*
Variable Handling Keywords	*del, global, nonlocal*



```
>>> import keyword
>>> keyword.iskeyword('del')
True
```



```
>>> del = 3
SyntaxError: invalid syntax
```

Operators

- Arithmetic Operators
- Logical Operators
- Equality Operators
- Comparison Operators

Arithmetic Operators

Operation	Result
`x + y`	sum of *x* and *y*
`x - y`	difference of *x* and *y*
`x * y`	product of *x* and *y*
`x / y`	quotient of *x* and *y*
`x // y`	floored quotient of *x* and *y*
`x % y`	remainder of `x / y`
`-x`	*x* negated
`+x`	*x* unchanged
`x ** y`	*x* to the power *y*

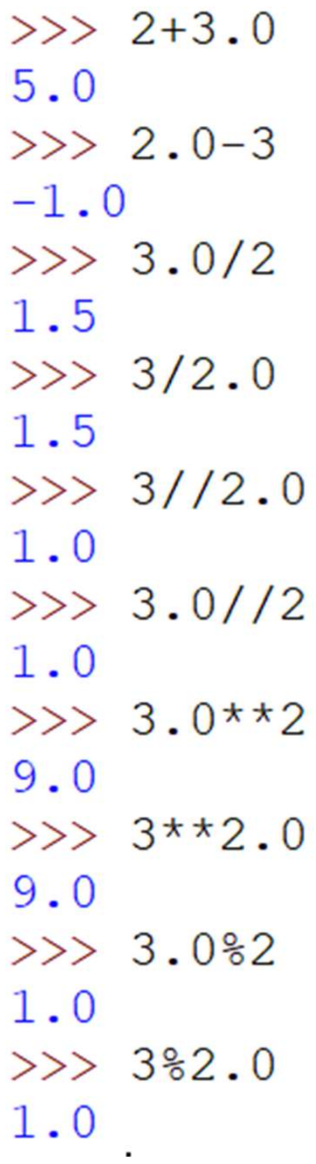
```
>>> 2+3
5
>>> 2.0+3.0
5.0
>>> 2-3
-1
>>> 2*3
6
>>> 2.0*3.0
6.0
>>> 2/3
0.6666666666666666
>>> 3/2
1.5
>>> 3//2
1
>>> 3%2
1
>>> 3**2
9
>>> -2
-2
```

Mixed mode arithmetic

- If operands are of different types?

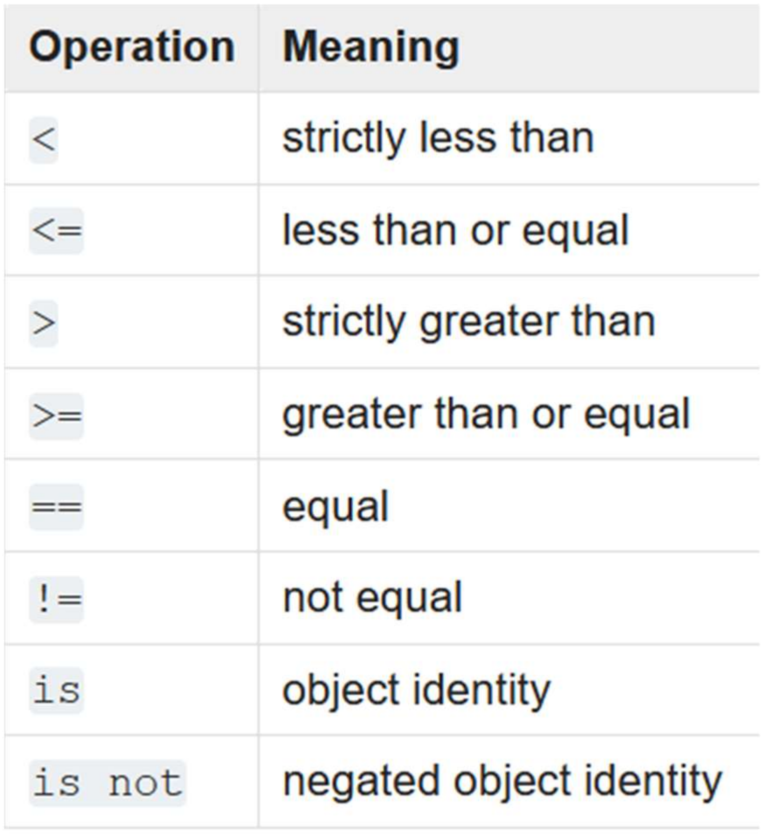
- Narrower (less general) and Wider (more general) Types
  - ➢ Float is wider (more general) than integer
    - o All integers are floats but not vice-versa

- Narrower type is promoted to wider type
  - ➢ Integer is promoted to float

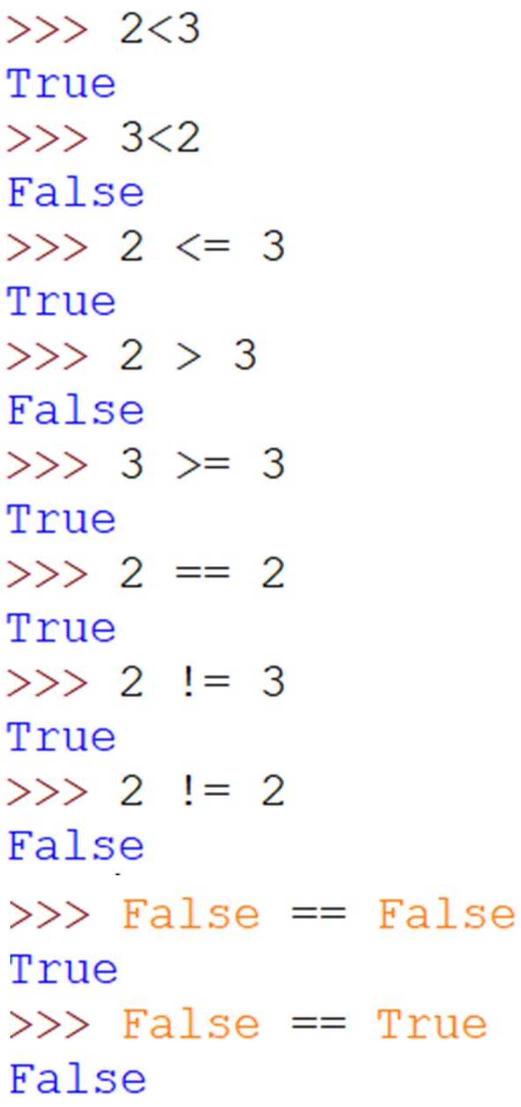


```
>>> 2+3.0
5.0
>>> 2.0-3
-1.0
>>> 3.0/2
1.5
>>> 3/2.0
1.5
>>> 3//2.0
1.0
>>> 3.0//2
1.0
>>> 3.0**2
9.0
>>> 3**2.0
9.0
>>> 3.0%2
1.0
>>> 3%2.0
1.0
```

Comparison Operators



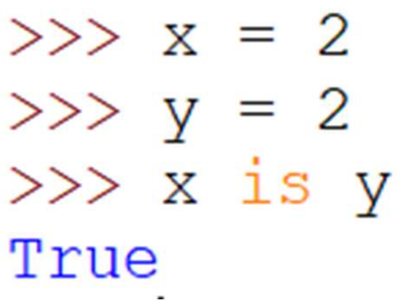
Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity



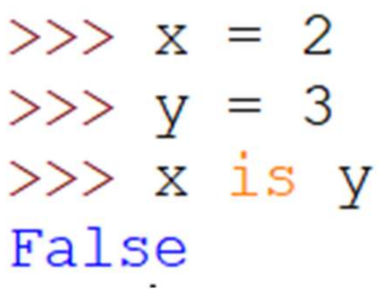
```
>>> 2<3
True
>>> 3<2
False
>>> 2 <= 3
True
>>> 2 > 3
False
>>> 3 >= 3
True
>>> 2 == 2
True
>>> 2 != 3
True
>>> 2 != 2
False
>>> False == False
True
>>> False == True
False
```

What is the difference between == and *is*?

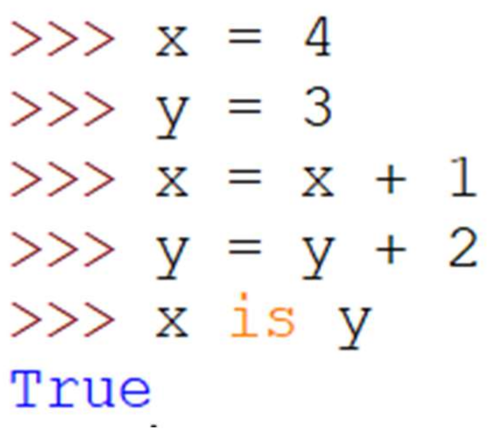
is operator



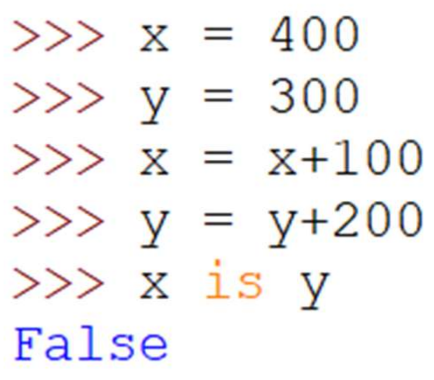
```
>>> x = 2
>>> y = 2
>>> x is y
True
```



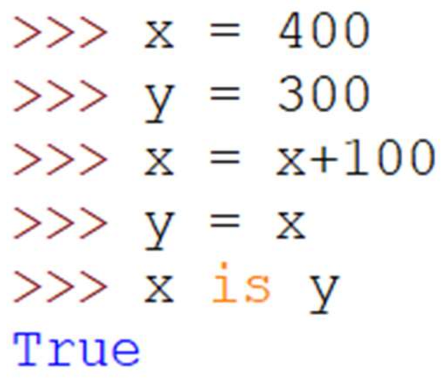
```
>>> x = 2
>>> y = 3
>>> x is y
False
```



```
>>> x = 4
>>> y = 3
>>> x = x + 1
>>> y = y + 2
>>> x is y
True
```



```
>>> x = 400
>>> y = 300
>>> x = x+100
>>> y = y+200
>>> x is y
False
```

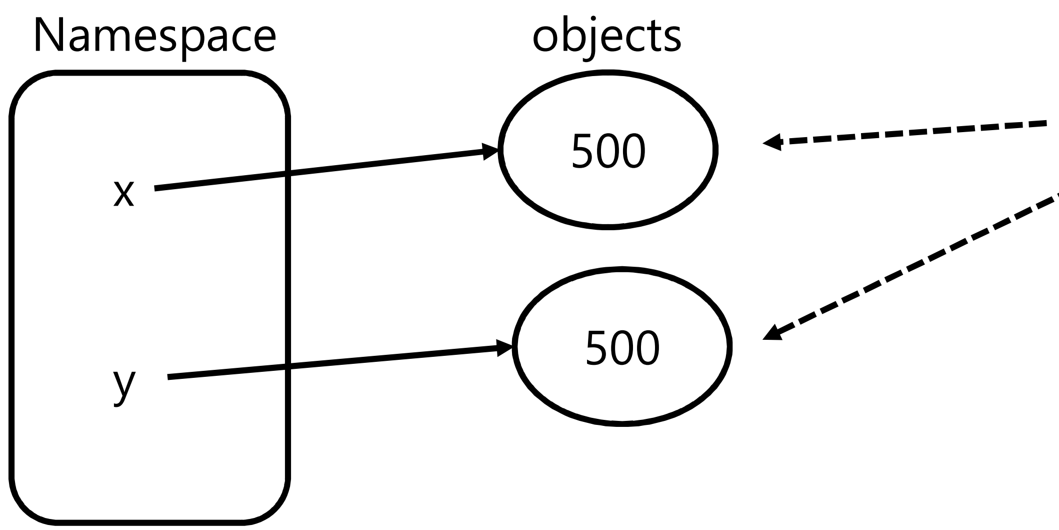


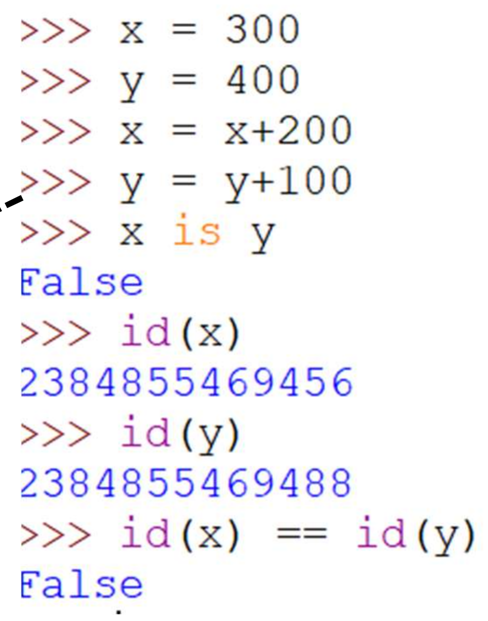
```
>>> x = 400
>>> y = 300
>>> x = x+100
>>> y = x
>>> x is y
True
```

is operator

- Binary operator
  - ➢ Returns true if identity of both operands is same

- What is identity?



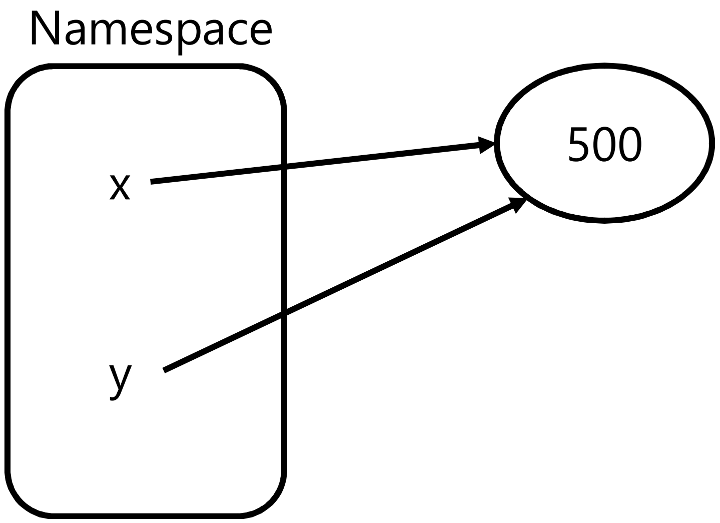


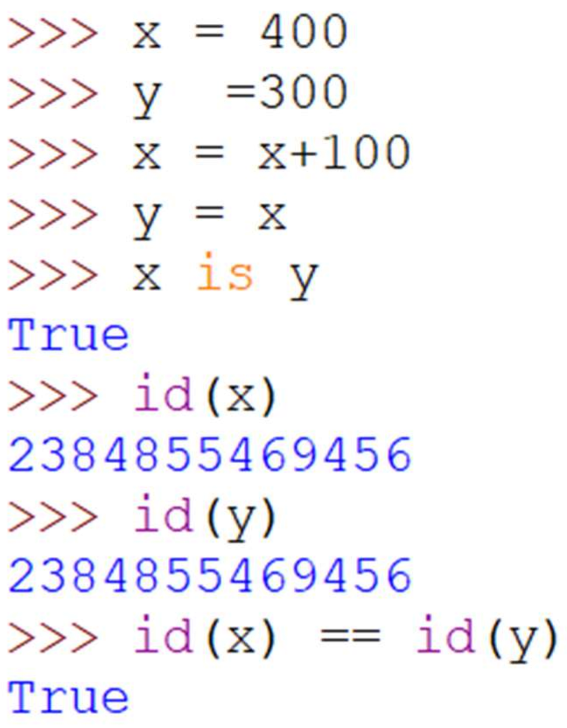
```
>>> x = 300
>>> y = 400
>>> x = x+200
>>> y = y+100
>>> x is y
False
>>> id(x)
2384855469456
>>> id(y)
2384855469488
>>> id(x) == id(y)
False
```

Keyword *is*

- Binary operator
  - Returns true if identity of both operands is same

- What is identity?





```
>>> x = 400
>>> y  =300
>>> x = x+100
>>> y = x
>>> x is y
True
>>> id(x)
2384855469456
>>> id(y)
2384855469456
>>> id(x) == id(y)
True
```

Logical/Boolean Operators

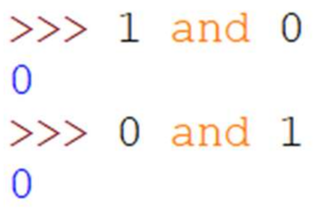
Operator	Operation	Result	Remark
and (conditional and)	x **and** y	If x is false, then x, else y	• Short-circuit operator • Only evaluates the second argument if the first one is true
or (conditional or)	x **or** y	If x is false, then y, else x	• Short-circuit operator • Only evaluates the second argument if the first one is false
not (unary negation)	**not** x	If x is false, then *True*, else *False*	• Low priority than non-Boolean operators • Ex: not a == b means not (a==b)

https://en.wikipedia.org/wiki/Short-circuit_evaluation

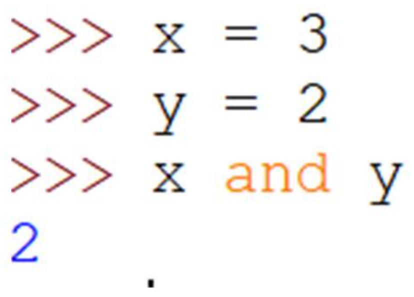
Demo in IDLE

and Operator

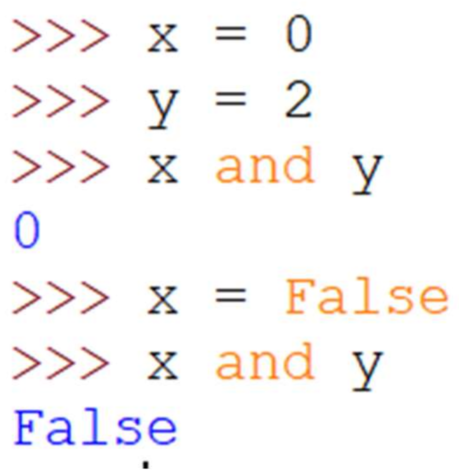
x **and** y: if x is false, then x, else y



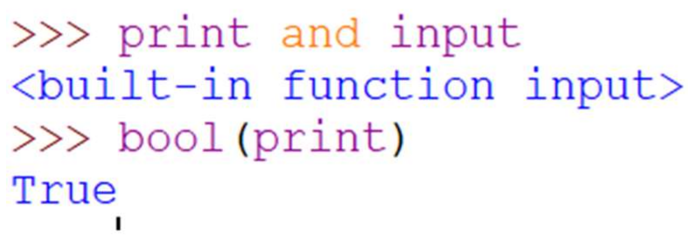
```
>>> 1 and 0
0
>>> 0 and 1
0
```



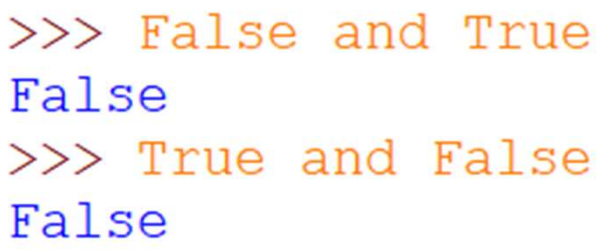
```
>>> x = 3
>>> y = 2
>>> x and y
2
```



```
>>> x = 0
>>> y = 2
>>> x and y
0
>>> x = False
>>> x and y
False
```



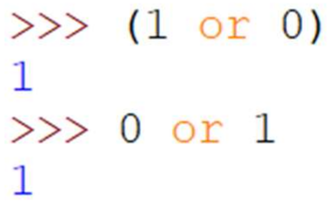
```
>>> print and input
<built-in function input>
>>> bool(print)
True
```



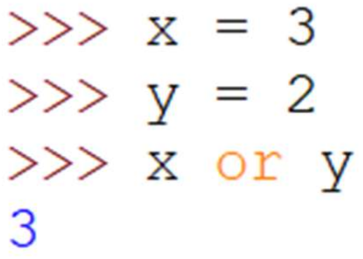
```
>>> False and True
False
>>> True and False
False
```

or Operator

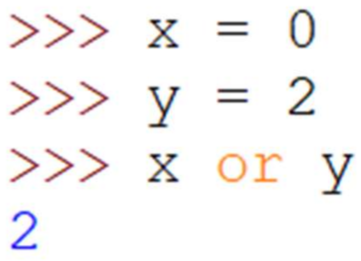
x **or** y: if x is false, then y, else x



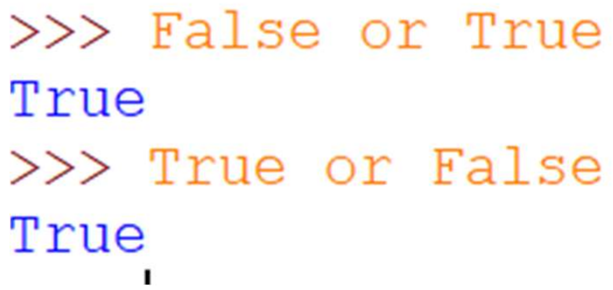
```
>>> (1 or 0)
1
>>> 0 or 1
1
```



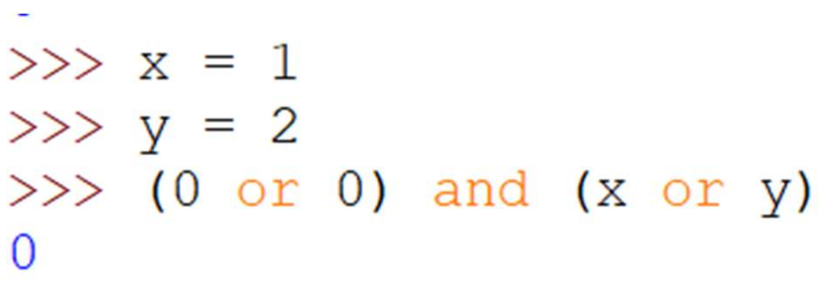
```
>>> x = 3
>>> y = 2
>>> x or y
3
```



```
>>> x = 0
>>> y = 2
>>> x or y
2
```



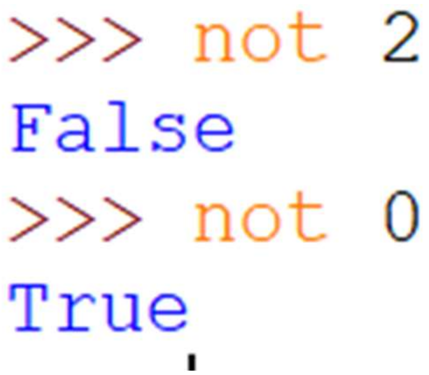
```
>>> False or True
True
>>> True or False
True
```



```
>>> x = 1
>>> y = 2
>>> (0 or 0) and (x or y)
0
```

not Operator

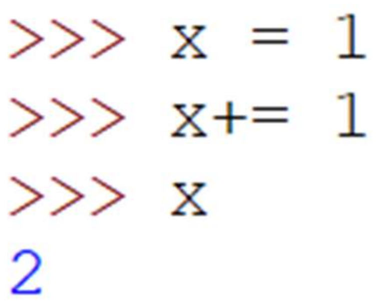
not x: If x is false, then *True*, else *False*



```
>>> not 2
False
>>> not 0
True
```

Augmented Assignment Operators

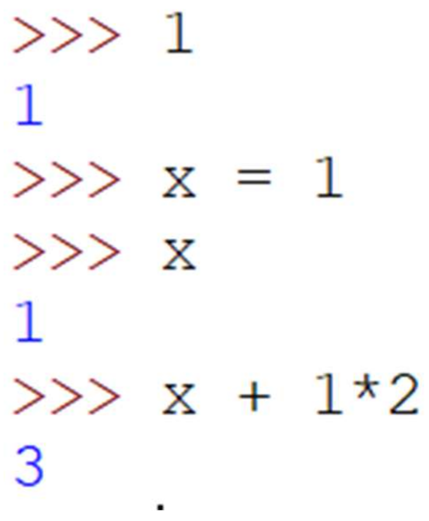
Operation	Description
x += y	x = x + y
x *= y	x = x * y
x /= y	x = x / y
x // = y	x = x //y
x ** = y	x = x**y



```
>>> x = 1
>>> x+= 1
>>> x
2
```

Expressions

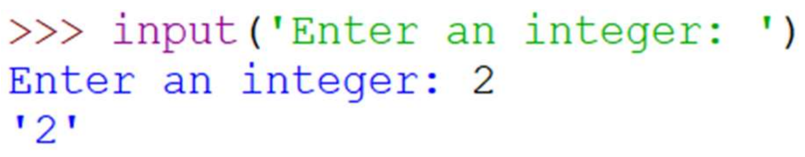
- Expressions
  - A piece of syntax evaluated to some value
  - Combination of operators and operands
    - Value is an expression
    - Variable is an expression
    - Combination of values, variables and operators is also an expression



```
>>> 1
1
>>> x = 1
>>> x
1
>>> x + 1*2
3
```

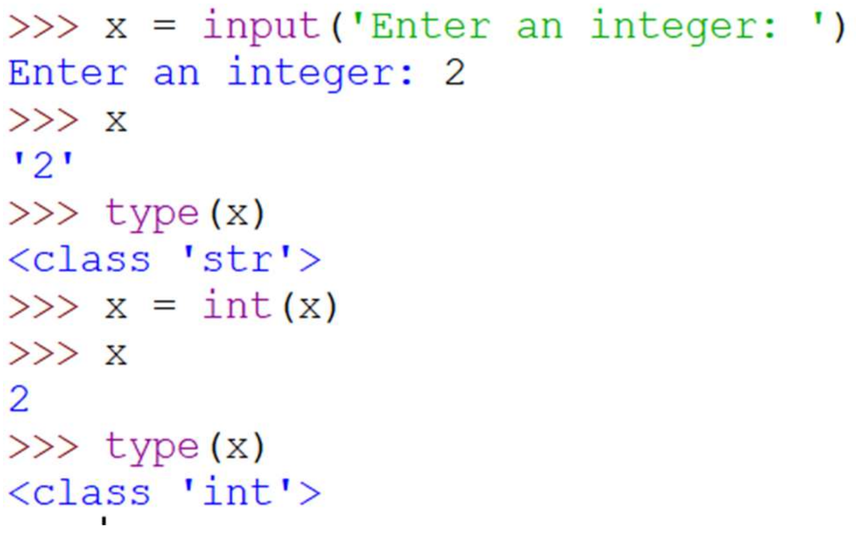
Standard IO: Input

- Input

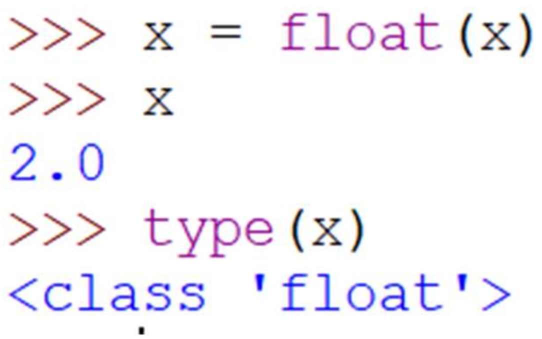


```
>>> input('Enter an integer: ')
Enter an integer: 2
'2'
```

- Type Casting
  - ➢ Conversion of one type to other
  - ➢ Example:

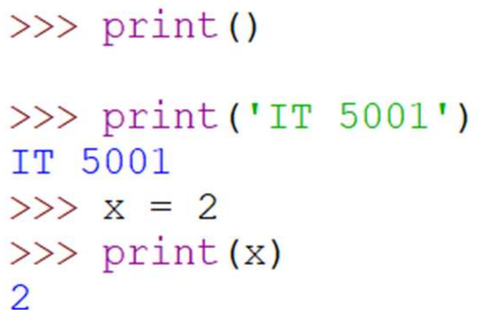


```
>>> x = input('Enter an integer: ')
Enter an integer: 2
>>> x
'2'
>>> type(x)
<class 'str'>
>>> x = int(x)
>>> x
2
>>> type(x)
<class 'int'>
```

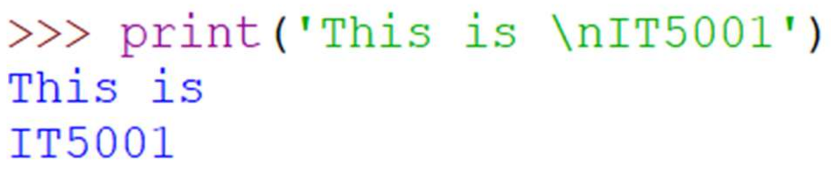


```
>>> x = float(x)
>>> x
2.0
>>> type(x)
<class 'float'>
```

Standard IO: Output



```
>>> print()

>>> print('IT 5001')
IT 5001
>>> x = 2
>>> print(x)
2
```



```
>>> print('This is \nIT5001')
This is
IT5001
```

Precedence

Operator	Description
( )	Parenthesis
**	Exponentiation
+x, -x	x unchanged, x negated
*, /, //, %	Multiplication, division, floor division, remainder
+, -	Addition, Subtraction
in, not, <, <=, >, >=, ==, !=	Membership, comparison and identity tests
not x	Boolean NOT
and	Boolean AND
or	Boolean OR

Precedence

(4-5) * 3 – 7 % 4 ** 2 / 3

-1 * 3 - 7 % 4 ** 2 / 3

-1 * 3 - 7 % 16 / 3

-3 - 7 % 16 / 3

-3 - 7 / 3

-3 - 2.333334

-5.2333334

Operator
( )
**
+x, -x
*, /, //, %
+, -
in, not, <, <=, >, >=, ==, !=
not x
and
or

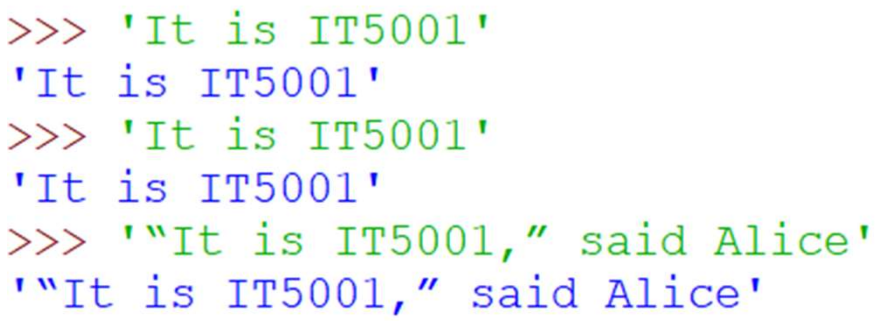
Equal precedence:
Association is from left to right

Strings

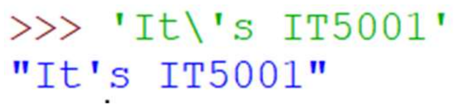
- Strings are **indexed** **sequence of characters**
- Example Strings
  - It is IT5001
  - It’s IT5001
  - “It is IT5001,” said Alice
  - C:\new\IT5001

Strings

- Single quotes:
  - Example
    - It is IT5001
    - “It is IT5001,” said Alice
    - It’s IT5001



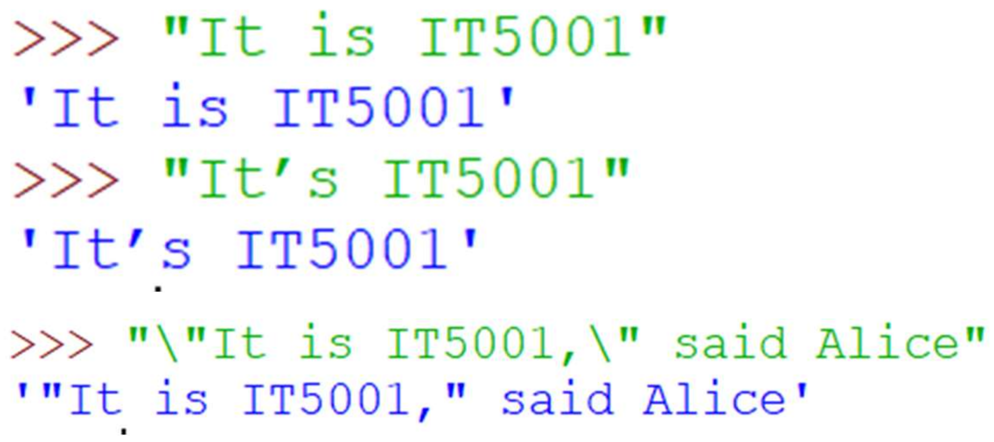
```
>>> 'It is IT5001'
'It is IT5001'
>>> 'It is IT5001'
'It is IT5001'
>>> '“It is IT5001,” said Alice'
'“It is IT5001,” said Alice'
```



```
>>> 'It\'s IT5001'
"It's IT5001"
```

Strings

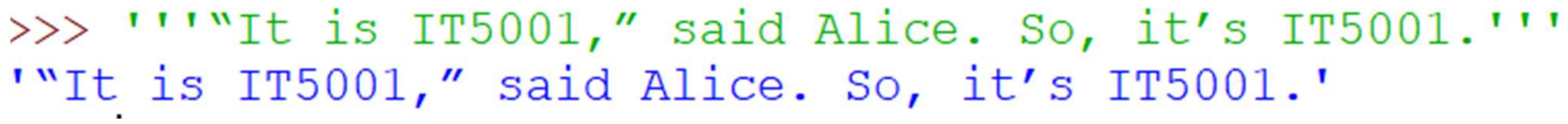
- Double quotes
  - ➢ Example:
    - o It is IT5001
    - o It’s IT5001
    - o “It’s IT5001,” said Alice.



```
>>> "It is IT5001"
'It is IT5001'
>>> "It’s IT5001"
'It’s IT5001'
>>> "\"It is IT5001,\" said Alice"
'"It is IT5001," said Alice'
```

Strings

- Triple Quotes and Triple Double Quotes
  - Doesn’t require escape character for single quote and double quotes within strings
  - Support multiline strings



```
>>> '''“It is IT5001,” said Alice. So, it’s IT5001.'''
'“It is IT5001,” said Alice. So, it’s IT5001.'
```

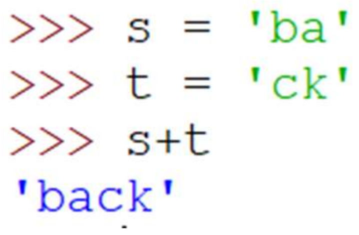
String Manipulations

- String Operators
- Built-in String Functions
- String Indexing and Slicing
- Built-in String Methods
- String Formatting

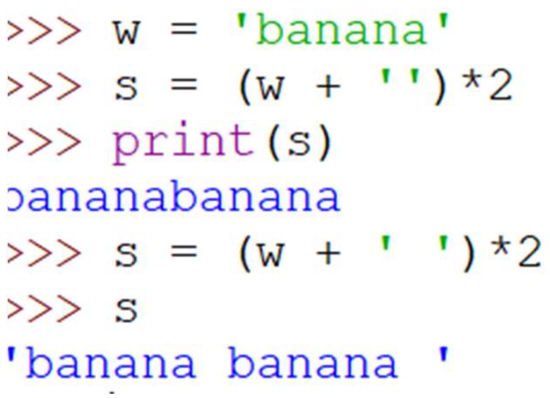
String Operators

Operator	Operation	Result	Example
+	x + y	Concatenates strings x and y	'This is ' + 'IT5001' = 'This is IT5001'
*	x * c	A new string with 'c' copies of string x, where c is integer	'Hi'*2 = 'HiHi'
in	x in y	Returns True if string x is in string y	'Hi' in 'Hi IT5001' → True
not in	x not in y	Returns True if string x is not in string y	'Hi' not in 'Hi IT5001' → False

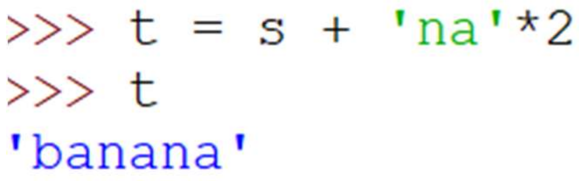
String Operators



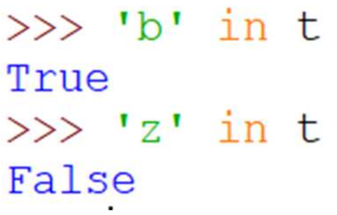
```
>>> s = 'ba'
>>> t = 'ck'
>>> s+t
'back'
```



```
>>> w = 'banana'
>>> s = (w + '')*2
>>> print(s)
bananabanana
>>> s = (w + ' ')*2
>>> s
'banana banana '
```



```
>>> t = s + 'na'*2
>>> t
'banana'
```



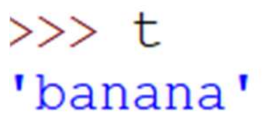
```
>>> 'b' in t
True
>>> 'z' in t
False
```

Built-in String Functions

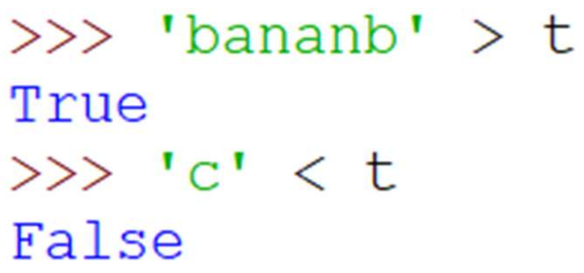
Function	Return Value	Example
len()	Length of the string	len('Hi') = 2
chr(*i*)	A string representing a character whose Unicode point is the integer $i, 0 < i < 1114111$ - Returns a single character string for an input integer	chr(123) = '{'
ord()	ASCII value of character (string with) - Returns integer value for an input single character string	ord('{') = 123
str()	Returns string representation of an object	str(2.5) = '2.5'

https://docs.python.org/3/library/functions.html

Lexicographical Ordering

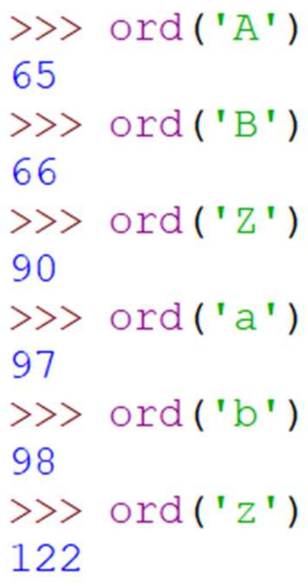


```
>>> t
'banana'
```

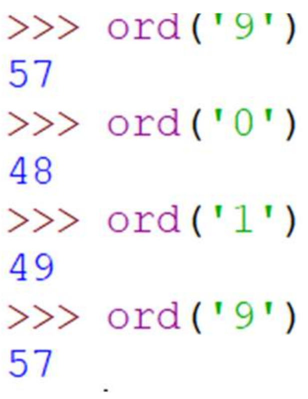


```
>>> 'bananb' > t
True
>>> 'c' < t
False
```

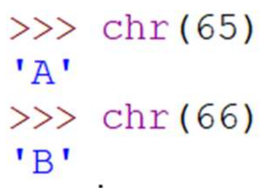
Unicode of Characters



```
>>> ord('A')
65
>>> ord('B')
66
>>> ord('Z')
90
>>> ord('a')
97
>>> ord('b')
98
>>> ord('z')
122
```



```
>>> ord('9')
57
>>> ord('0')
48
>>> ord('1')
49
>>> ord('9')
57
```



```
>>> chr(65)
'A'
>>> chr(66)
'B'
```

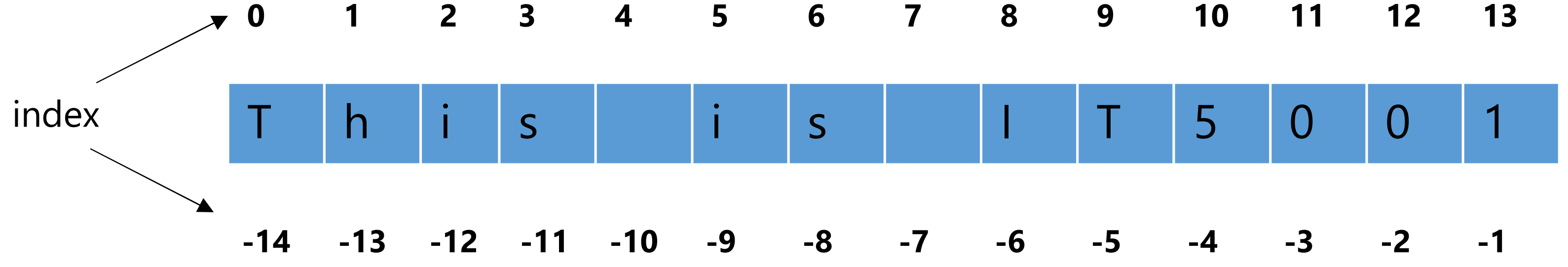
lexicographical ordering: first the first two letters are compared, and if they differ this determines the outcome of the comparison; if they are equal, the next two letters are compared, and so on, until either sequence is exhausted.

String Indexing and Slicing

- Strings are represented as compact arrays

string_example = ‘This is IT5001’

Indexing:



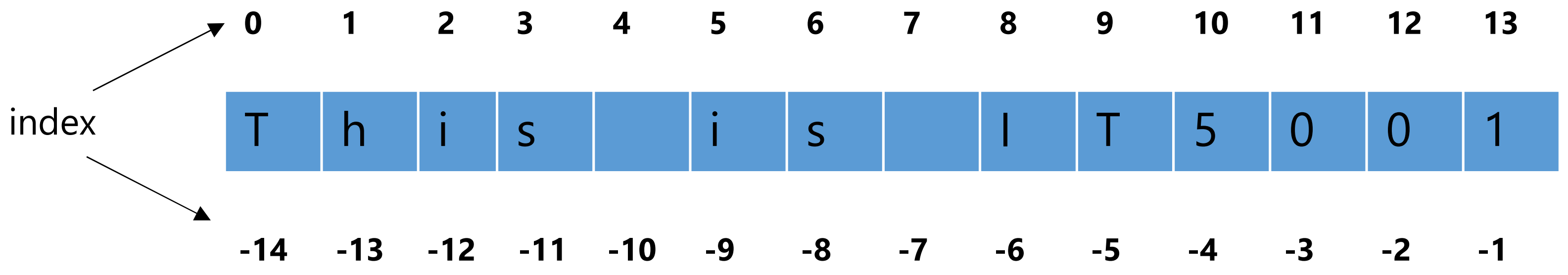
Slicing:

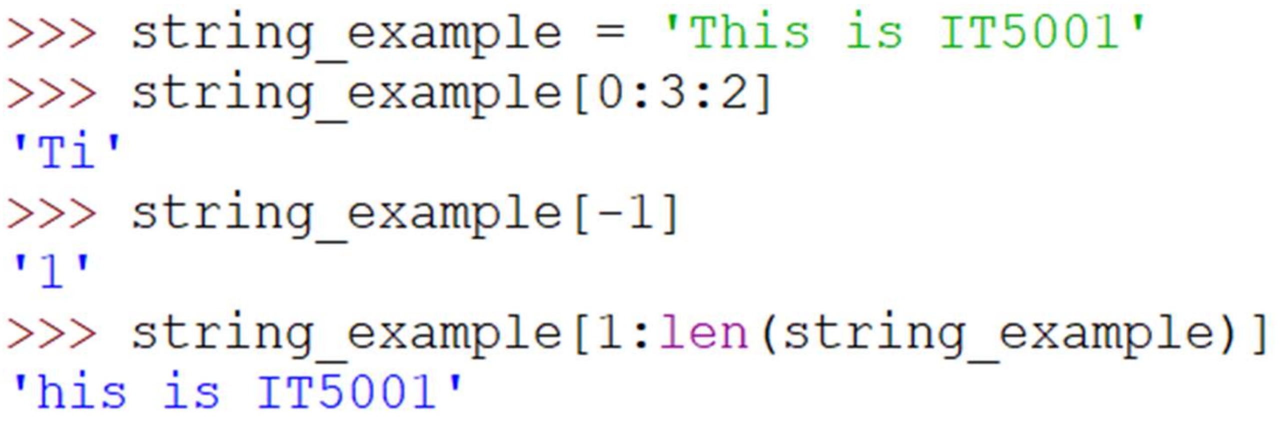
string_example[start : end : stride]

String Indexing and Slicing

string_example = ‘This is IT5001’

Indexing:



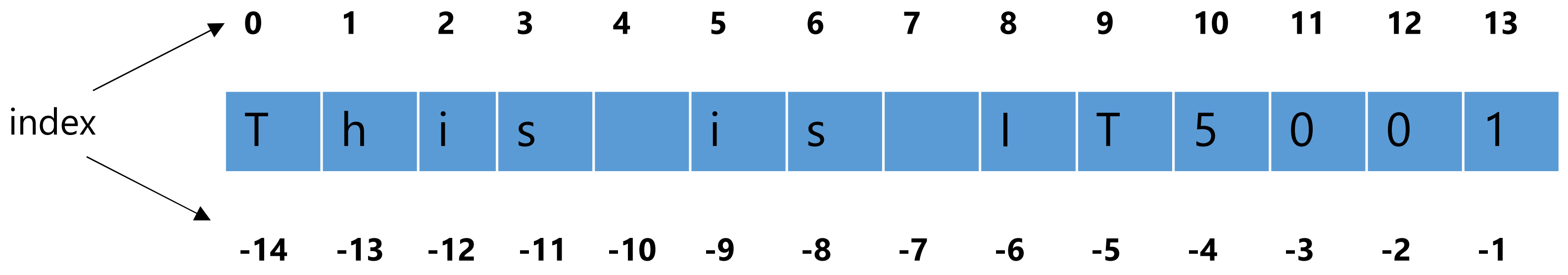


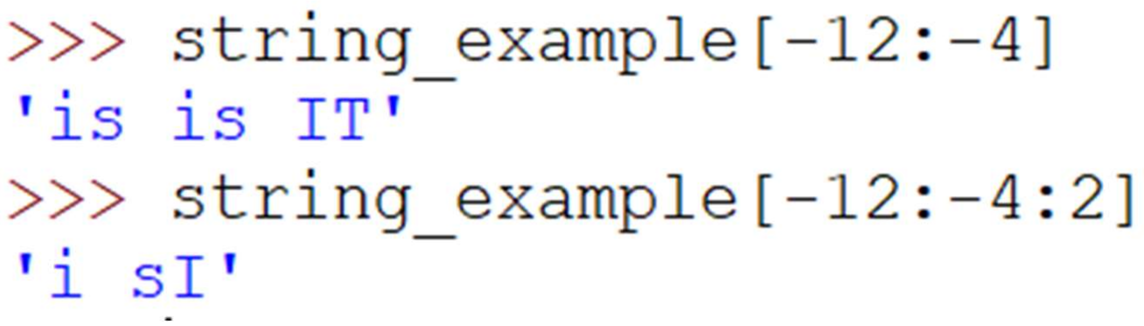
```
>>> string_example = 'This is IT5001'
>>> string_example[0:3:2]
'Ti'
>>> string_example[-1]
'1'
>>> string_example[1:len(string_example)]
'his is IT5001'
```

String Indexing and Slicing

string_example = ‘This is IT5001’

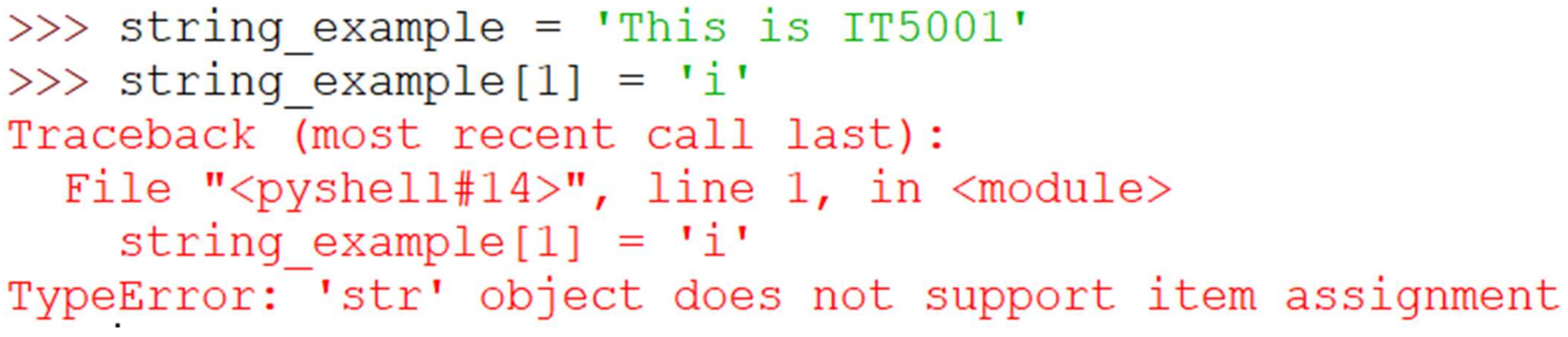
Indexing:





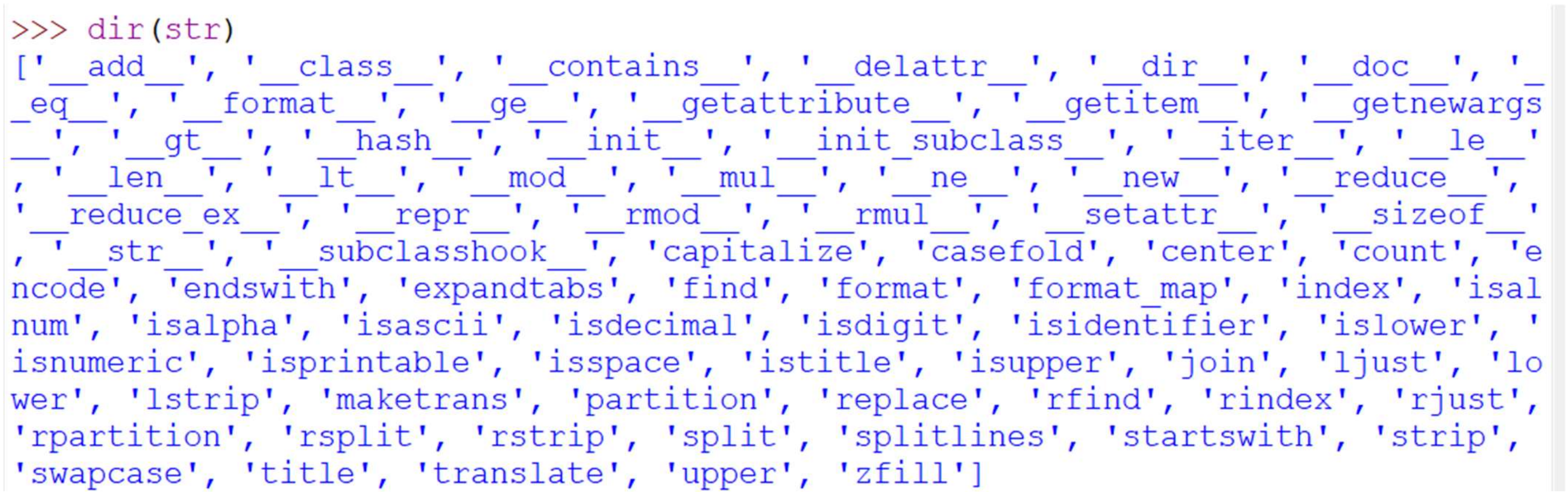
```
>>> string_example[-12:-4]
'is is IT'
>>> string_example[-12:-4:2]
'i sI'
```

Immutability of Strings



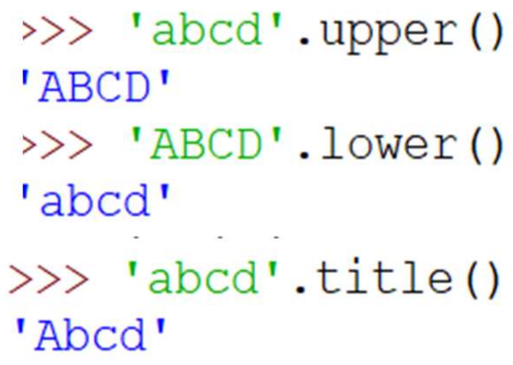
```
>>> string_example = 'This is IT5001'
>>> string_example[1] = 'i'
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    string_example[1] = 'i'
TypeError: 'str' object does not support item assignment
```

String Methods



```
>>> dir(str)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '_
_eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs
__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__'
, '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__'
, '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'e
ncode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isal
num', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', '
isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lo
wer', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip',
'swapcase', 'title', 'translate', 'upper', 'zfill']
```

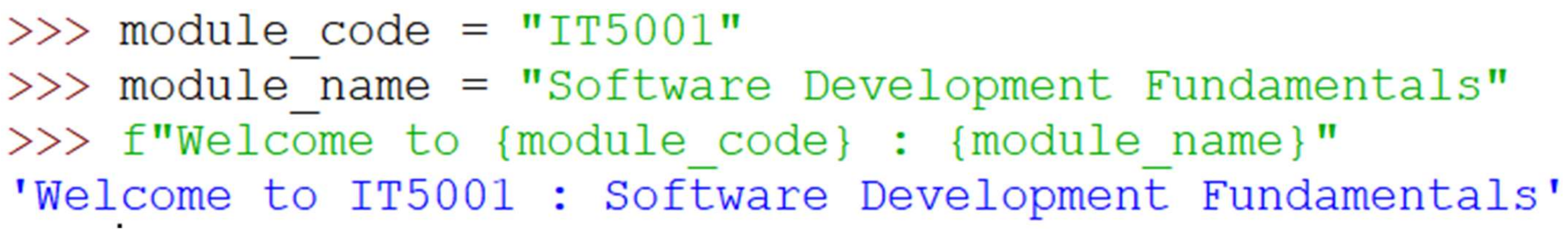
- Case Conversion
  - ➢ upper, lower, title, etc.



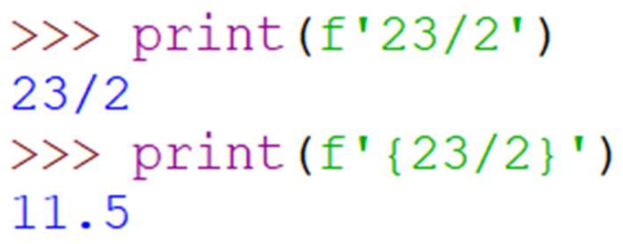
```
>>> 'abcd'.upper()
'ABCD'
>>> 'ABCD'.lower()
'abcd'
>>> 'abcd'.title()
'Abcd'
```

f-strings

- f-strings
  - ➢ Strings prefixed with ‘f’
  - ➢ ‘f’ stands for formatted strings
  - ➢ Expressions can be embedded in strings
    - o Expressions evaluated at run time.
  - ➢ Contains replacement fields, delimited by curly braces



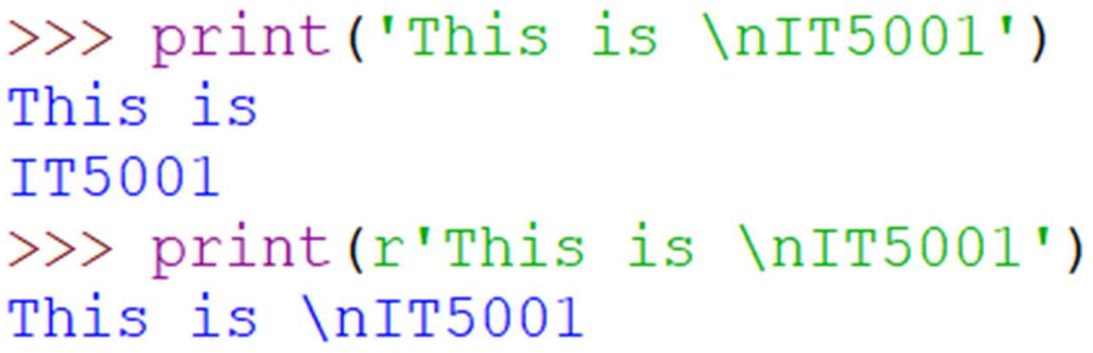
```
>>> module_code = "IT5001"
>>> module_name = "Software Development Fundamentals"
>>> f"Welcome to {module_code} : {module_name}"
'Welcome to IT5001 : Software Development Fundamentals'
```



```
>>> print(f'23/2')
23/2
>>> print(f'{23/2}')
11.5
```

Raw Strings

- Raw Strings
  - ➢ Strings prefixed with literal ‘r’



```
>>> print('This is \nIT5001')
This is
IT5001
>>> print(r'This is \nIT5001')
This is \nIT5001
```

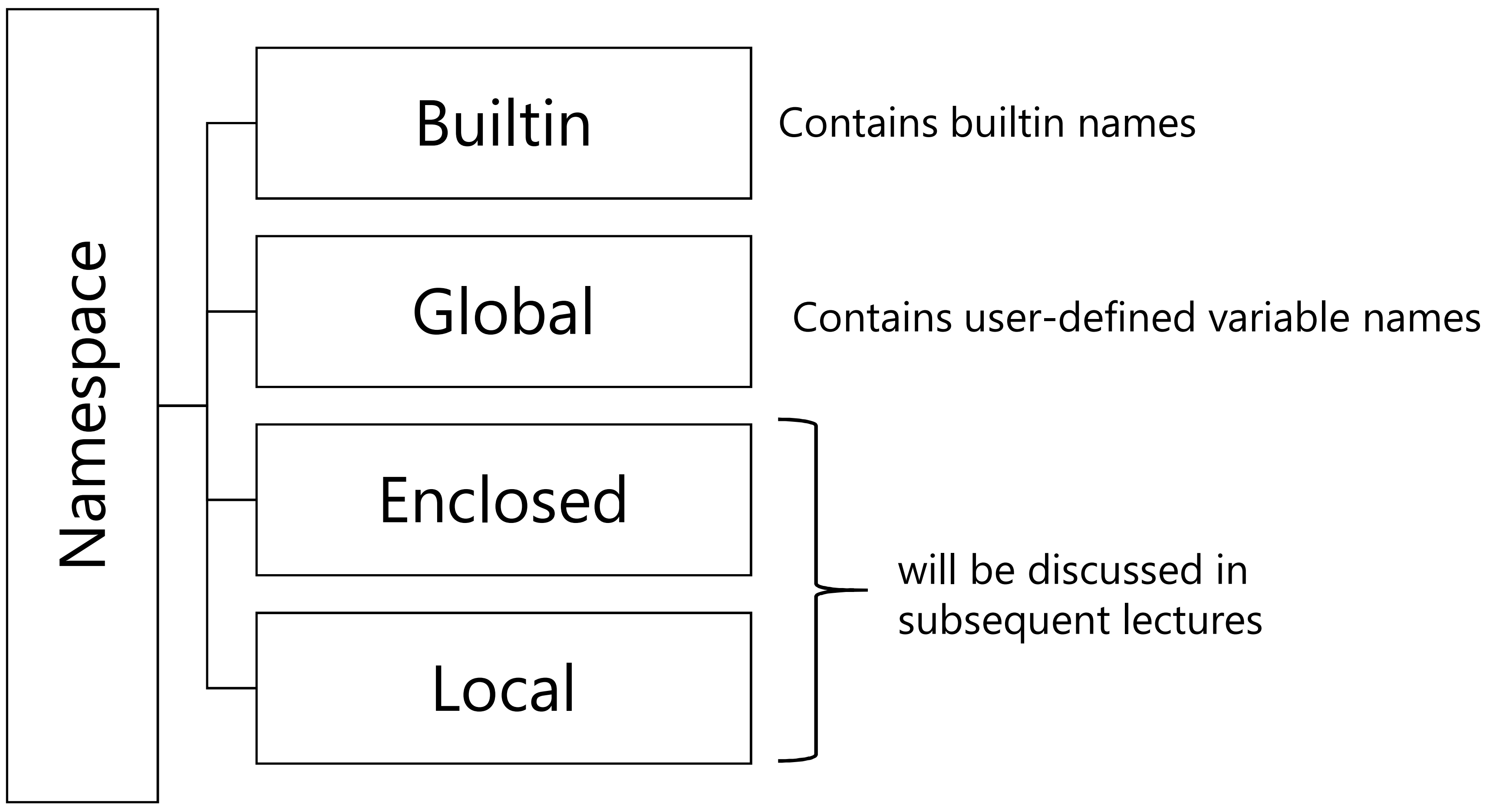
Conclusion

- Numeric and Boolean Types
- Operators and Precedence
- Expressions and Statements
- Strings, String Operators, String Functions, and String Methods
- Immutability
- **Next Class**: Libraries and User-defined Functions

Miscellaneous

Namespaces

Namespaces


Namespace
Builtin
Global
Enclosed
Local
Contains builtin names
Contains user-defined variable names
will be discussed in subsequent lectures

Builtin Namespace

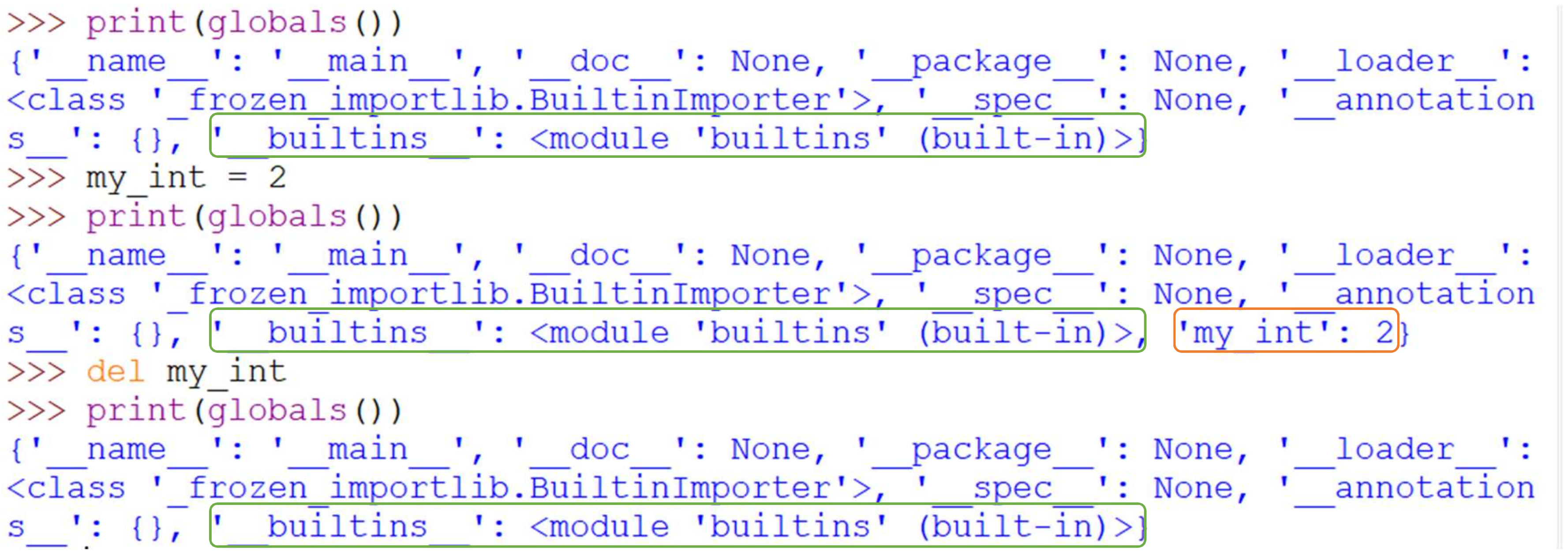
- Contains names of __builtin__ module:
  - Datatypes
    - Int, float, etc.
  - Functions
    - print, input, etc.
  - Exceptions
    - NameError, SyntaxError, etc.

- Check dir(__builtins__)

- Will be created (destroyed) when Python interpreter starts (closes)

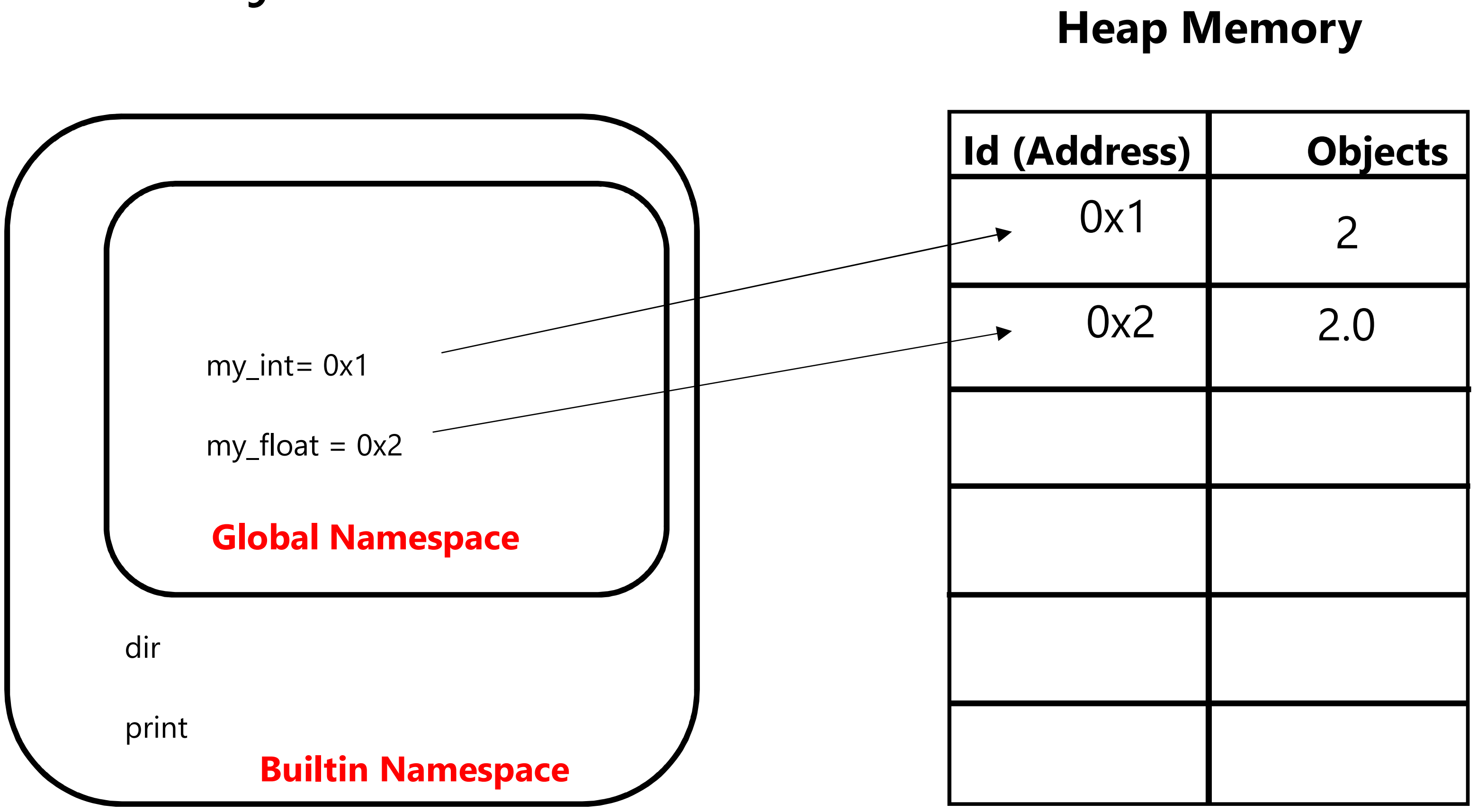
- What if you want to use a name in builtins?

Global Namespace



```
>>> print(globals())
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__':
<class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotation
s__': {}, '__builtins__': <module 'builtins' (built-in)>}
>>> my_int = 2
>>> print(globals())
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__':
<class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotation
s__': {}, '__builtins__': <module 'builtins' (built-in)>, 'my_int': 2}
>>> del my_int
>>> print(globals())
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__':
<class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotation
s__': {}, '__builtins__': <module 'builtins' (built-in)>}
```

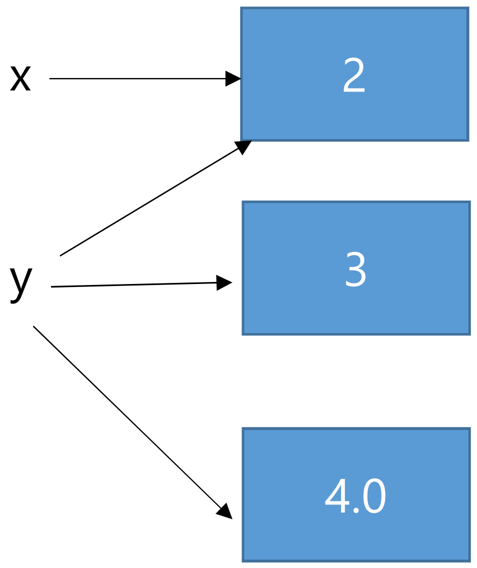
How are objects stored?



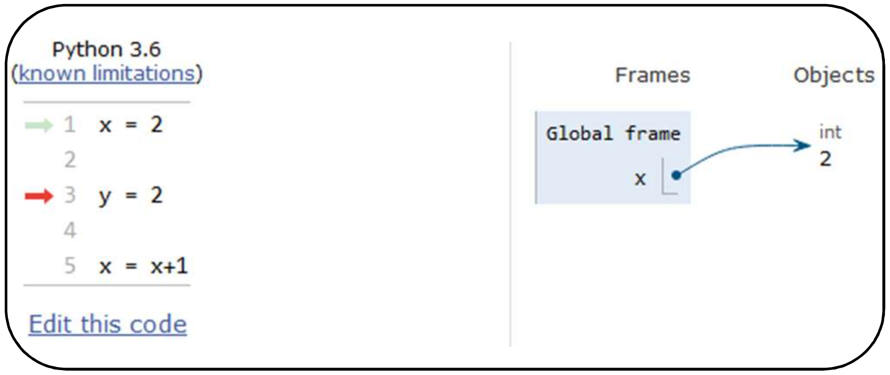
Interpreter first searches names in Global Namespace
If name is not there in global namespace, searches in builtin namespace
If name is not in builtin namespace, throws 'NameError'

How are objects stored?

- x = 2
- y = 2
- y = 3
- y = 4.0



How are objects stored?

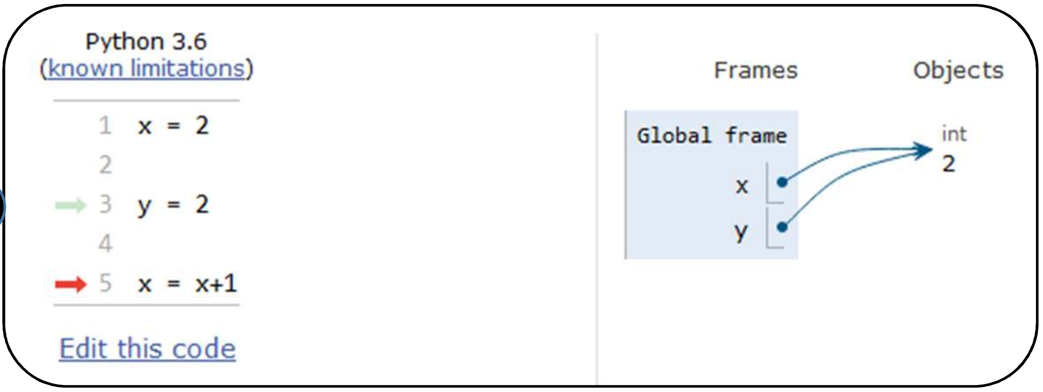


Python 3.6
(known limitations)

```
x = 2

y = 2

x = x+1
```

1

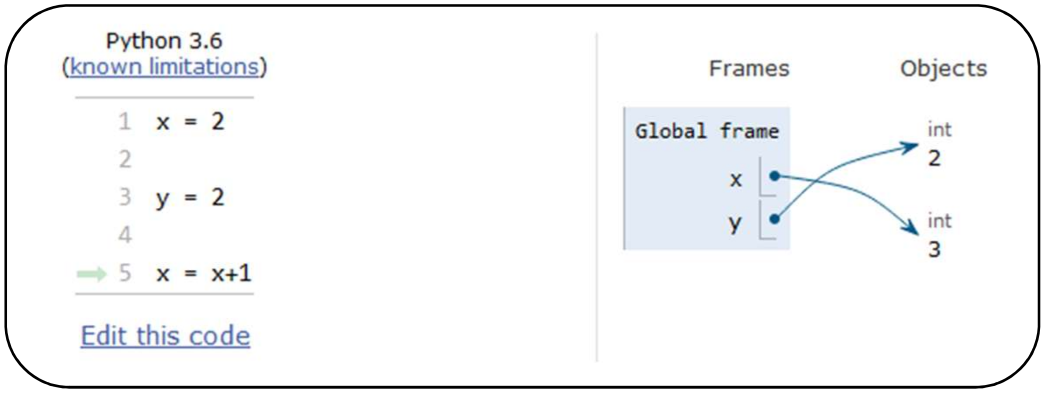
Demo: pythontutor.com



2

Python 3.6
(known limitations)

```
x = 2

y = 2

x = x+1
```



Python 3.6
(known limitations)

```
x = 2

y = 2

x = x+1
```

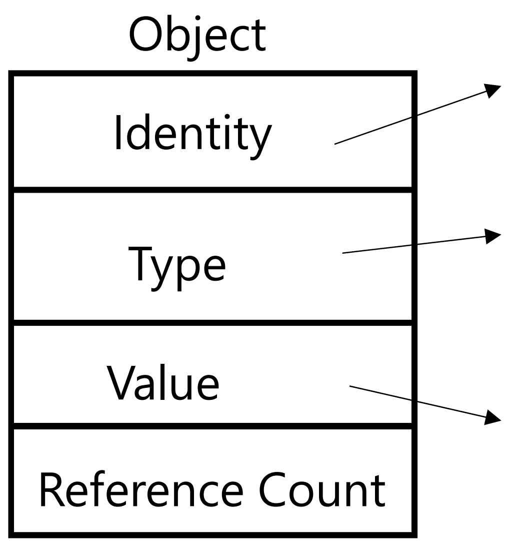
3

Memory Management

- Python does memory management automatically
- Private heap to store objects
- Memory management depends on object type

Data Model: Objects, Values, and Types

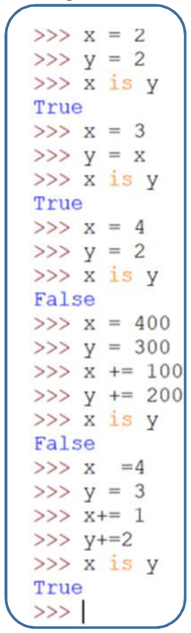
- Objects are Python’s abstraction for data
- Data in program is represented by objects and relation between objects



Object’s address in memory
Can be obtained using *id()*

Specifies the types of object
Can be obtained using *type()*

Immutable : numbers, strings, tuples, etc.
Mutable : lists, dictionaries



```
>>> x = 2
>>> y = 2
>>> x is y
True
>>> x = 3
>>> y = x
>>> x is y
True
>>> x = 4
>>> y = 2
>>> x is y
False
>>> x = 400
>>> y = 300
>>> x += 100
>>> y += 200
>>> x is y
False
>>> x  =4
>>> y = 3
>>> x+= 1
>>> y+=2
>>> x is y
True
>>> |
```

Why is this behaviour?