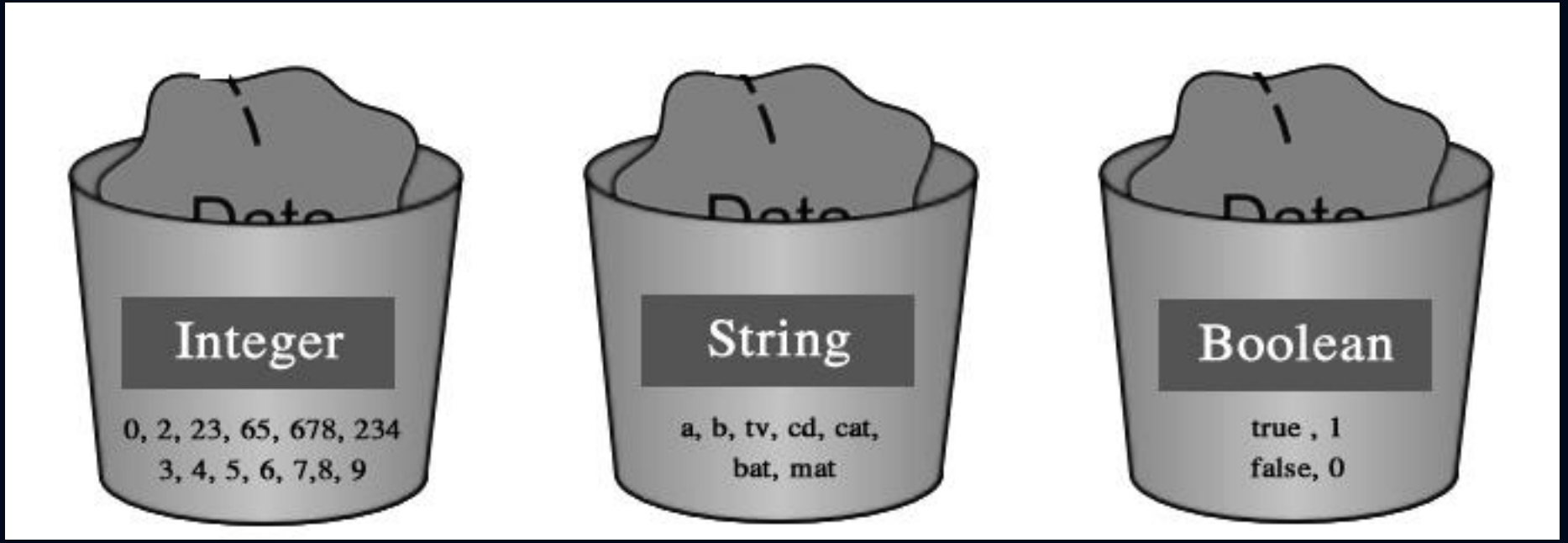
Python  Programming

Day 2: Data Types, Identifiers and Operators

Data Types, Identifiers and Operators

- Data Types
- Identifiers and Keywords
- Operators and Expression
- Comments

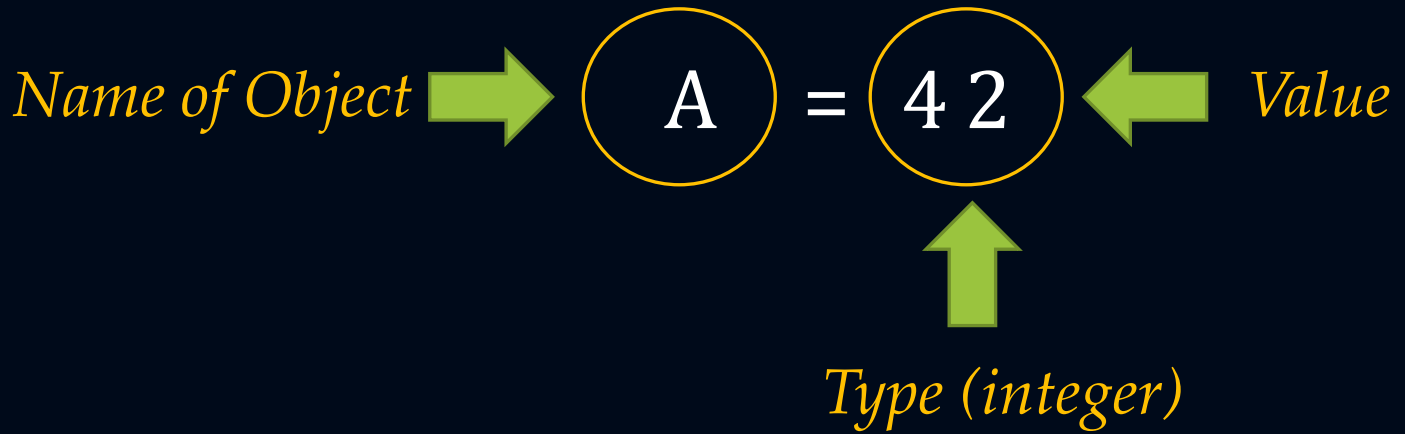
Data Types



A data type is an attribute that specifies the type of data that the object can hold and the operations that can be performed on it.

Data Types

Each object has an **identity**, a **type** (which is also known as its class), and a **value**.

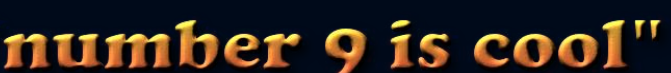



```
>>> type(A)
<class 'int'>
>>> id(A)
507082528
```


Data Types

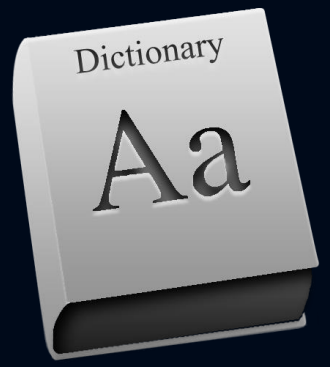
The following list briefly introduces some of Python's data types:

- **Numbers** represents data that you want to do math with.
- **Strings** represents text characters and binary data.
- **Sequences** represents lists of related data that you might want to sort, merge, and so on.

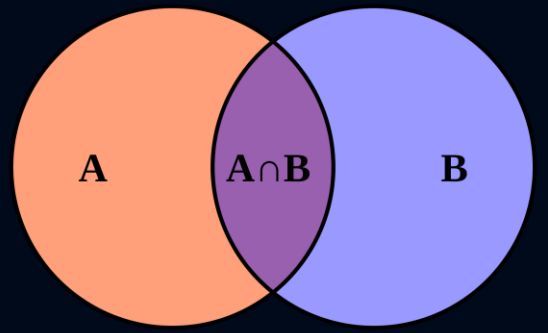


Data Types

Dictionaries are collections of data that associate a unique key with each value.



Sets are for doing set operations (finding the intersection, difference, and so on) with multiple values.



Files are for data that is or will be stored as a document on a computer.

Compare two objects

```
def compare (a, b):
    #compare two objects
    if a is b:
        # a and b are the same object
        print ("Both objects are the same")
    if a == b:
        # a and b have the same value
        print ("Both objects have same value")
    if type(a) is type(b):
        # a and b have the same type
        print ("Both objects have same type")
```

The built-in function id() returns the identity of an object as an integer.

The 'is' operator compares the identity of two objects.

The built-in function type() returns the type of an object

Compare Objects

Syntax: isinstance(object, classinfo)

Returns true if the object argument is an instance of the *classinfo* argument, or of a (direct, indirect or virtual) subclass thereof.

```
>>> isinstance(5, int)
True
>>> isinstance('Hello', str)
True
>>> isinstance('4.34', float)
False
>>> isinstance(4.34, float)
True
>>> isinstance(5, int) and isinstance('Hello', str)
True
```


Type conversion

Each Python type comes with a built-in command that attempts to convert values of another type into that type

int() str() float()

bool() list()

The convert command takes any value and converts it to an instance of itself, if possible, or complains otherwise

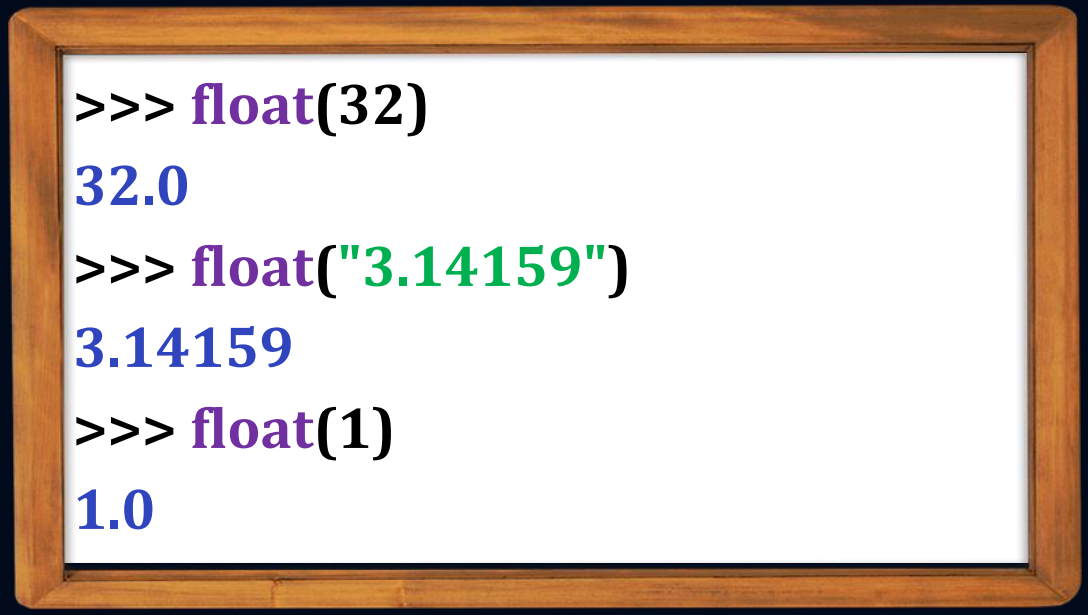
Type conversion - Integer

int can also convert floating-point values to integers, but remember that it truncates the fractional part:

```
>>> int ("32")
32
>>> int (-2.3)
-2
>>> int (3.9999)
3
>>> int ("hello")
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    int("hello")
ValueError: invalid literal for int () with base 10: 'hello'
```


Type conversion - Float

The **float()** command converts integers and strings to floating point numbers:



Type conversion - String

The **str()** command converts any argument given to it to type string:

```
>>> str (32)
'32'
>>> str(3.14149)
'3.14149'
>>> str(True)
'True'
>>> str(true)
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    str(true)
NameError: name 'true' is not defined
```


Type conversion - Boolean

Python assigns Boolean values to values of other types.

For numerical types like integers and floating-points, zero values are false and non-zero values are true.

For strings, empty strings are false and non-empty strings are true.

```
>>> bool(1)
True
>>> bool(0)
False
>>> bool("Hi")
True
>>> bool("")
False
>>> bool(3.14159)
True
>>> bool(0.0)
False
```


Type conversion – Class Practice

- **Convert “Hello world” to a list object**
- **Convert (4/3) to integer**
- **Convert 50 to string**

Mutable Objects

If an object's value can be modified, the object is said to be **mutable**. If the value cannot be modified, the object is said to be **immutable**.

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Any object that does not support item assignment is said to be immutable.

Identifiers and Keywords

A Python identifier is a name used to identify a variable, function, class, module, or other object.

For the above statement in Python it means you are *binding a name* (x) *to an object* (5). You can have multiple names for the same object.

Naming rules

- Names must start with either a letter or an underscore character (_).
- You can't use any of Python's **reserved words** or keywords.
- Names are **case-sensitive**. num is different from NUM and nUm.
- By convention, most Python programmers use lowercase for names that stand for values.
- It's a Good Idea to use meaningful names.

Naming rules

If you give a variable an illegal name, you get a syntax error:

```
>>> 76trombones = "big parade"
SyntaxError: invalid syntax
>>> more$ = 1000000
SyntaxError: invalid syntax
>>> class = "Computer Science 101"
SyntaxError: invalid syntax
```

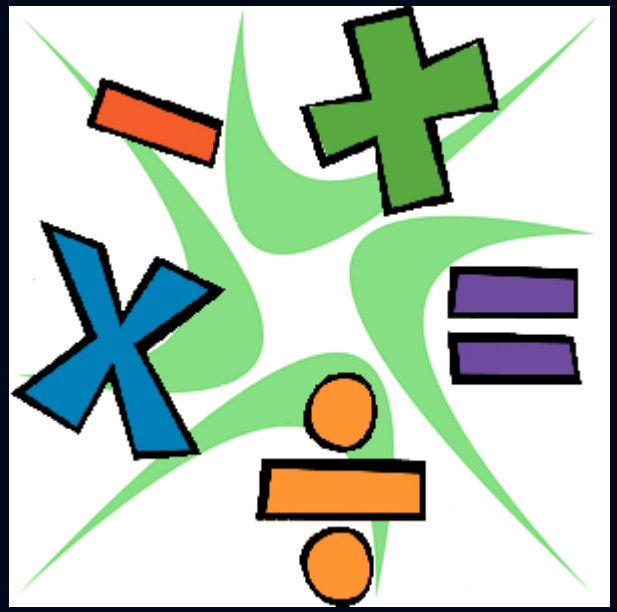

Keywords

These are words that have specific meanings to Python. Below is a table of **keywords** in python

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	not	or	pass
print	raise	return	try	while	with
yield					

Operators and Expressions

Operators are special symbols that represent computations like addition and multiplication. The values the operator uses are called operands



Operators - Arithmetic operators

Python understands a variety of math symbols.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder
()	Grouping

A variable is replaced with its value before operation is performed
E.g. z = x + y

Operators - Arithmetic operators

Some of these operators also work on data types other than numbers, but they may work differently.

```
>>> [1,2,3]*2
[1, 2, 3, 1, 2, 3]
>>> [1,2,3]+[4,5,6,7]
[1, 2, 3, 4, 5, 6, 7]
>>> [1,2,3]-[1,2,1]
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    [1,2,3]-[1,2,1]
TypeError: unsupported operand type(s) for -: 'list' and 'list'
```


Operators - Arithmetic operators

Some of these operators also work on data types other than numbers, but they may work differently.

```
>>> [1,2,3]*2
[1, 2, 3, 1, 2, 3]
>>> [1,2,3]+[4,5,6,7]
[1, 2, 3, 4, 5, 6, 7]
>>> [1,2,3]-[1,2,1]
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    [1,2,3]-[1,2,1]
TypeError: unsupported operand type(s) for -: 'list' and
'list'
```

You can't use an operator with two incompatible data types. E.g. “Hello”+2

Arithmetic Operators – Class Practice

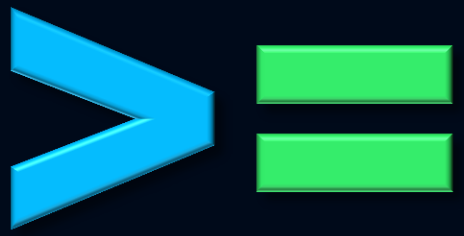
Try to evaluate the following numerical expressions in your head, then use the Python interpreter to check your results:

```
>>> 5 % 2
>>> 9 % 5
>>> 15 % 12
>>> 12 % 15
>>> 6 % 6
>>> 0 % 7
>>> 7 % 0
```


Comparison operators



Comparison operators test the relative sizes of two pieces of data and give either True or False as the result.



Comparison operators

Python can compare values of most other data types, too. When you compare items by using these operators, the result is either True or False

```
>>> 'a' < 'b'
True
>>> 'z' > 'a'
True
>>> 'Z' > 'a'
False

>>> [1] < [2]
True
>>> [1,2,3] > [1,2]
True
>>>
```


Boolean operators

Python has three operators that test whether expressions are true or false. These are called Boolean operators

- **and** stops testing when it encounters a false condition.
- **or** stops testing when it encounters a true condition.
- **not** returns ***True*** if the expression is false and vice versa

Boolean operators

- Python tests an expression with **and** and **or** operators from left to right and returns the last value tested.

- These operators don't return True and False unless the expressions themselves use comparison operators.

```
>>> '1' and 1 and 'one'
'one'
>>> '1' or 1 or 'one'
'1'
>>> (2<3) or (5>6)
True
```

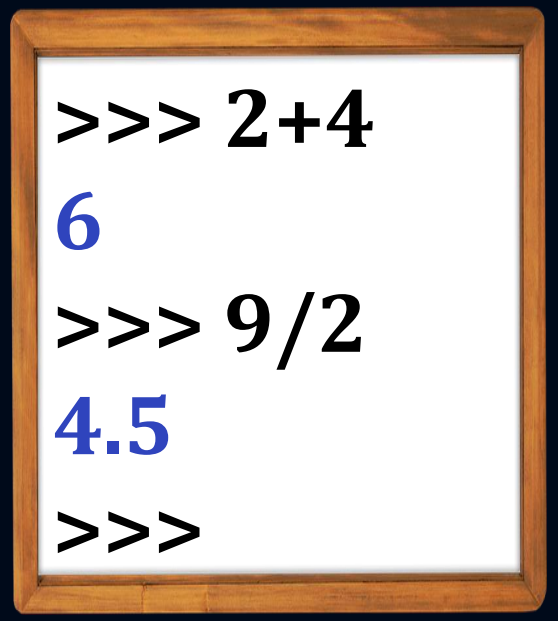

Boolean Operators – Class Practice

Enter the following expressions into the Python shell:

True or False
True and False
not(False) and True
True or 7
False or 7
True and 0
False or 8
"happy" and "sad"
"happy" or "sad"
"" and "sad"

Expressions

An expression is a combination of **values**, **variables**, and **operators**.



Expressions

```
>>> x = 4 + 8
>>> y = 2 * 5
>>> print (x + y)
22
```

The evaluation of an expression produces a value. Therefore expressions can appear on the right hand side of assignment statements.

Comments

Comment makes the source code easier for humans to understand, and are generally ignored by compilers and interpreters.

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

Comments should explain what a code is doing or why it is there

Comments

Program Documentation String.

A docstring is always the first line in a function. It can be more than one line if you begin and end it with three quotation marks

```
>>> def printme(me):
        '''
        Prints its argument.
        '''
>>> help(printme)
Help on function printme in module __main__:

printme(me)
    Prints its argument.
```

Docstring works with Python's help utility to describe a function without accessing the actual file.

Input

This is a way to get data directly from the user. The functions allows a prompt to be given to the user after the value between the parentheses.

```
applicant = input("Enter the applicant's name: ")
interviewer = input("Enter the interviewer's name: ")
time = input("Enter the appointment time: ")

print(interviewer, "will interview", applicant, "at", time)
```


Input

```
x = input("Enter an integer: ")
y = input("Enter another integer: ")
print('The sum of ', x, ' and ', y, ' is ',
x+y)

# wrong output
```

We do not want string concatenation, but integer addition. We need integer operands.

Input

```
xString = input ("Enter an integer: ")
x = int (xString)
yString = input ("Enter another integer: ")
y = int (yString)
Print ('The sum of ', x, ' and ', y, ' is ', x+y)
```


Composition

The process by which the result of one function is used as the input to another.

```
x = int (input("Enter an integer: "))
y = int (input("Enter another integer: "))
Print ('The sum of ', x, ' and ', y, ' is ', x+y)
```


Python  Programming

Tutorials

Exercise 1:

Take the sentence: All work and no play makes Jack a dull boy. Store each word in a separate variable, then print out the sentence on one line using print.

Exercise 2:

Add parenthesis to the expression 6 * 1 - 2 to change its value from 4 to -6.

Exercise 3:

Start the Python interpreter and enter bruce + 4 at the prompt. This will give you an error:
NameError: name 'bruce' is not defined

Now, assign a value to bruce so that bruce + 4 evaluates to 10.

Exercise 4:

Write a program (Python script) named name.py, which asks the user to enter your name, age and your sex and produce the output on the screen.

Exercise 5:

a. Use values between 1 and 10 as input for x and y

```
x = int (input("Enter an integer": "))
y = int (input("Enter another integer: "))
if x < y:
    print(x, "is less than", y)
elif x > y:
    print(x, "is greater than", y)
else:
    print(x, "and", y, "are equal")
```

b. Wrap this code in a function called compare(x, y). Call compare three times: one each where the first argument is less than, greater than, and equal to the second argument

Exercise 6

Write a Python program to add two objects if both objects are an integer type.

Exercise 7

Wrap this code in a function called dispatch(choice).

```
if choice == 'a':
   function_a()
elif choice == 'b':
   function_b()
elif choice == 'c':
   function_c()
else:
   print "Invalid choice"
```

Then define function_a, function_b, and function_c so that they print out a message saying they were called. For example:

```
def function_a():
 print("function_a was called...")
```

Put the four functions (dispatch, function_a, function_b, and function_c into a script named mod02e01.py.

At the bottom of this script add a call to dispatch('b'). Your output should be:
function_b was called...

Next Lecture ...

Python Programming

Day 3: String Manipulation