

# 1.1 Syllabus

* Numeric types
* String Fundamentals
* Input and Output

• Comments

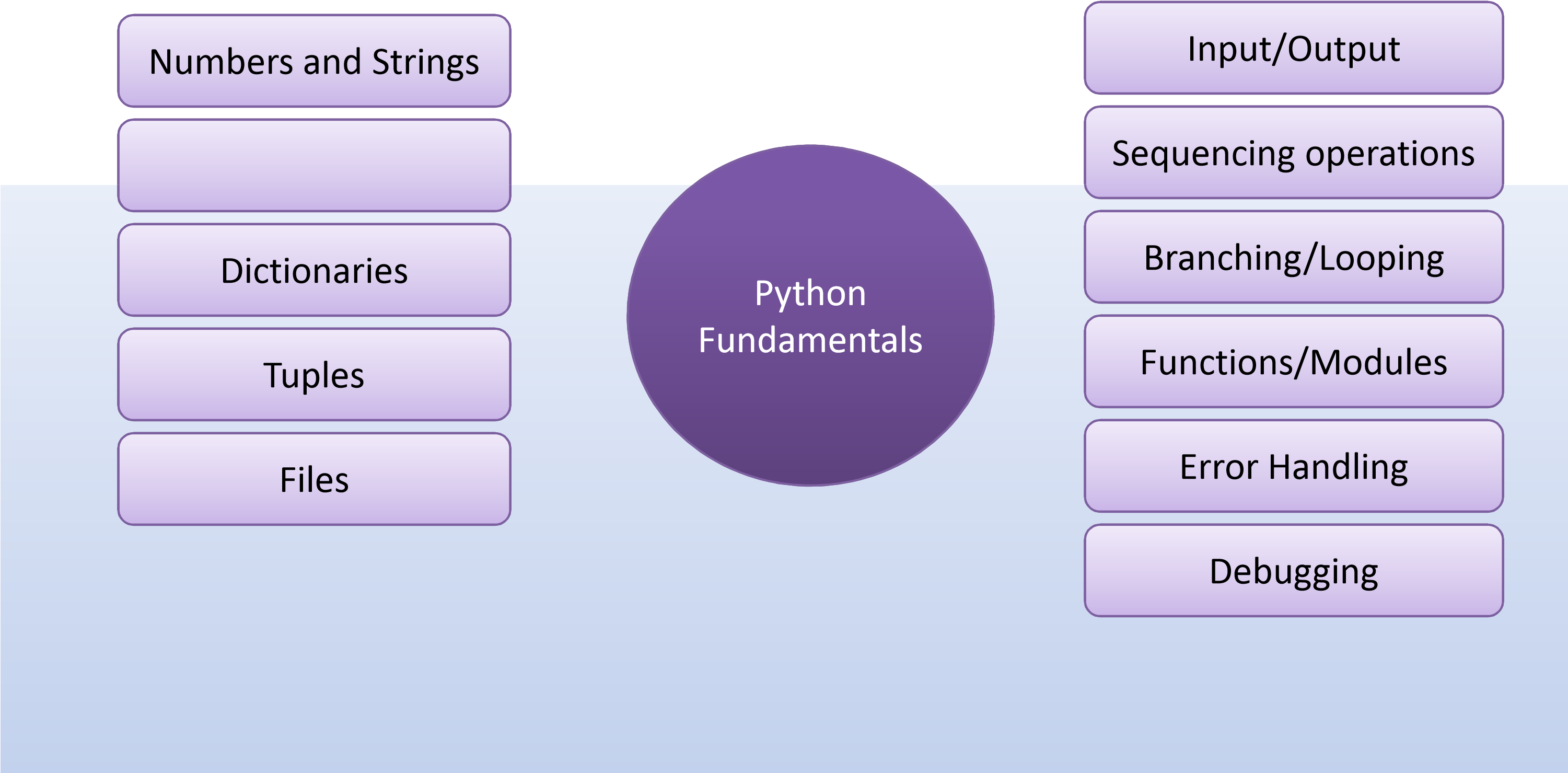


•

Errors

# Let’s Begin…

Information types Actions



# Mutables and Immutables

* Immutables (numbers, strings, tuples, frozensets)
  + None of the object types in the immutable category support in-place changes, though we can always run expressions to make new objects and assign their results to variables as needed.
* Mutables (lists, dictionaries, sets, bytearray)
  + Conversely, the mutable types can always be changed in place with operations that do not create new objects. Although such objects can be copied, in-place changes support direct modification.

# Working with Numbers

• Built-in numeric tools

* Operators: +, -, \*, /, >>, \*\*, etc.
* Mathematical functions: pow, abs, round, int, etc.

– Utility modules: random, math, etc.

|  |  |
| --- | --- |
| **Operators** | **Description** |
| **if y else z** | Ternary selection (x is evaluated only if y is true) |
| **or y** | Logical OR (y is evaluated only if x is false) |
| **and y** | Logical AND (y is evaluated only if x is true) |
| **not x** | Logical negation |
| **in y, x not in y** | Membership (iterables, sets) |
| **< y, x <= y, x > y, x >= y** | Magnitude comparison, set subset and superset; |
| **== y, x != y** | Value equality operators |
| **| y** | Bitwise OR, set union |
| **^ y** | Bitwise XOR, set symmetric difference |
| **& y** | Bitwise AND, set intersection |
| **<< y, x >> y** | Shift x left or right by y bits |
| **is y, x is not y** | Object identity tests |

# Basic Operators and Expressions

# Basic Operators and Expressions

|  |  |
| --- | --- |
| **Operators** | **Description** |
| **x + y** | Addition, concatenation; |
| **x – y** | Subtraction, set difference |
| **x \* y** | Multiplication, repetition; |
| **x % y** | Remainder, format; |
| **x / y, x // y** | Division: true and floor |
| **−x, +x** | Negation, identity |
| **˜x** | Bitwise NOT (inversion) |
| **x \*\* y** | Power (exponentiation) |
| **x[i]** | Indexing (sequence, mapping, others) |
| **x[i:j:k]** | Slicing |
| **x(...)** | Call (function, method, class, other callable) |
| **(...)** | Tuple, expression, generator expression |
| **[...]** | List, list comprehension |
| **{...}** | Dictionary, set, set and dictionary comprehensions |
| **x.attr** | Attribute reference |

Precedence of Common Operators

**Precedence**

**Operator**

Evaluated first. If the parentheses are nested, t

()

he

expression in the innermost pair is evaluated first

. If

there are several pairs of parentheses “on the same



level” (i.e., not nested), they are evaluated left

to right.

Evaluated second. If there are several, they are

\*\*

evaluated

right to left.

/ // \* %

Evaluated third. If there are several, t

hey are evaluated

left to right.

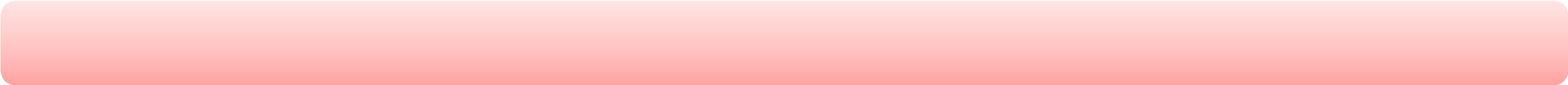
Evaluated last. If there are several, they are e

+

-

valuated

left to right.

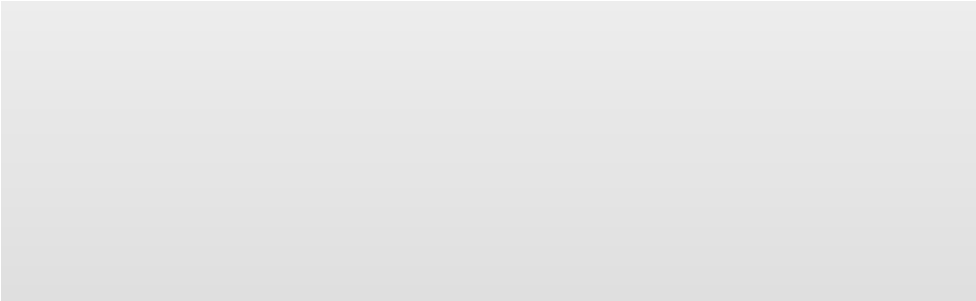


Refer:

https://docs.python.org/3/reference/expressions.htm

l#comparisons

Comparisons: Normal and Chained



x =

2

>>>

>>>

3

y =

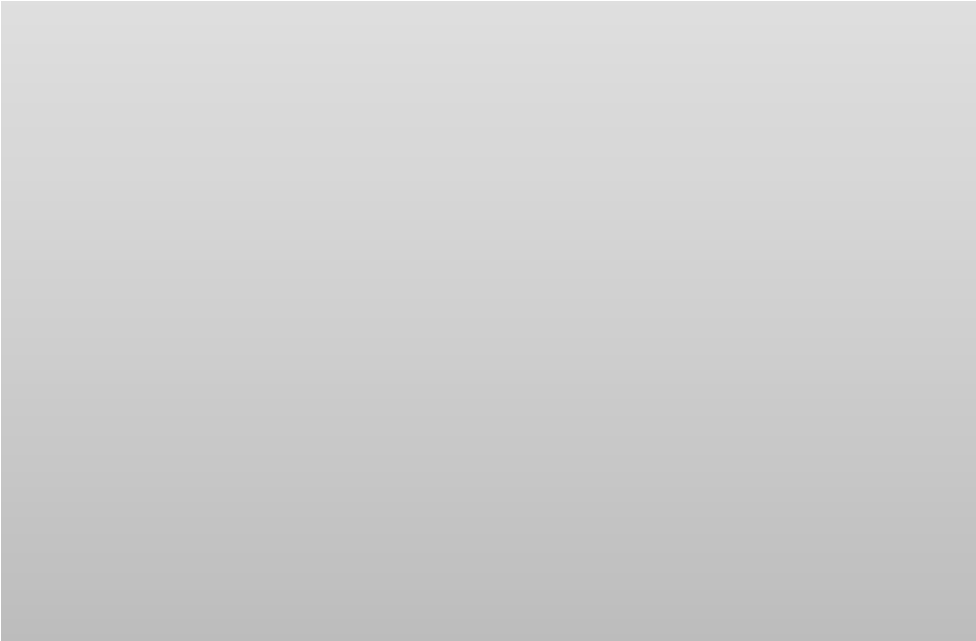
6

z =

>>>

x < y > z

>>>



False

1

>>>

x <

False

>>>

x

2

1

x >

>>>

True

(3.3)

int

>>>

3

3

int(1.1 + 2.2) ==

>>>

True



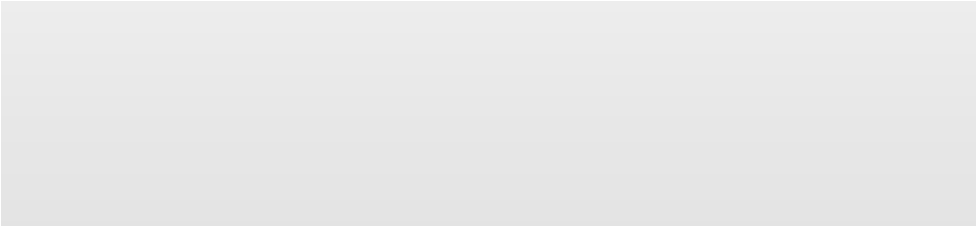
Normal



Chained

# Division

* X/Y : Classic and True Division – Keeps reminder
* X//Y: Floor Division – Truncates reminder



>>>

X =

10

>>>

Y =

3

>>>

X / Y



3.3333333333333335

>>>

X // Y

3

>>>

X /

3.0

3.3333333333333335

>>>

X /

3.0

3.3333333333333335

>>>

X //

3.0

3.0

>>>

X / float(Y

)

3.3333333333333335

>>>

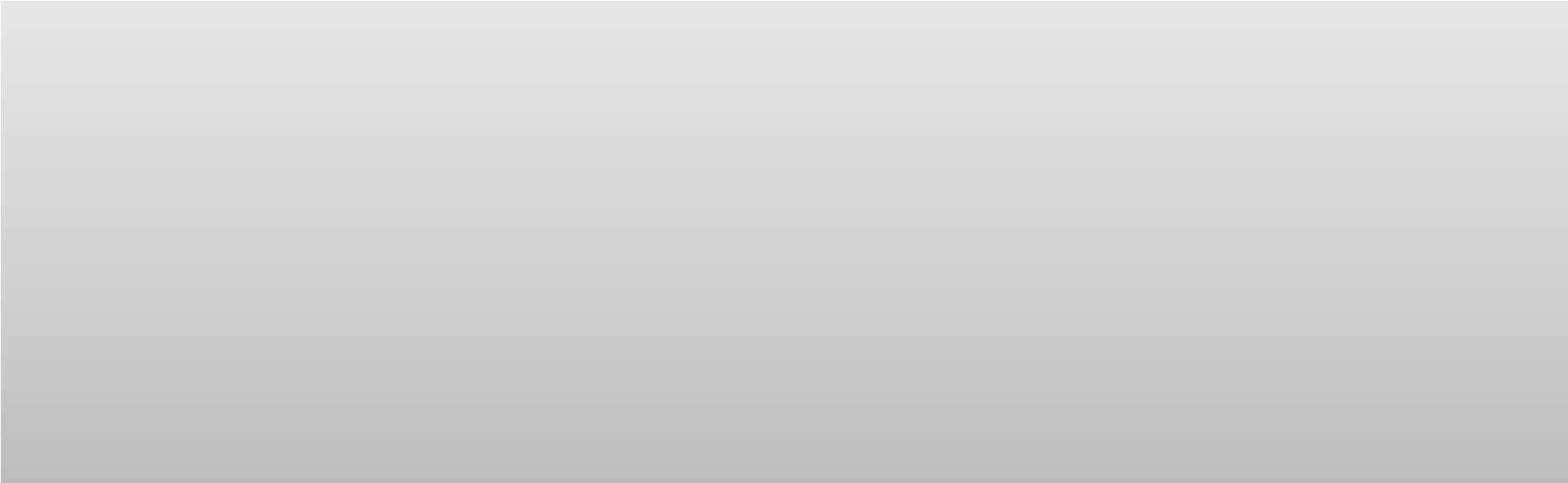
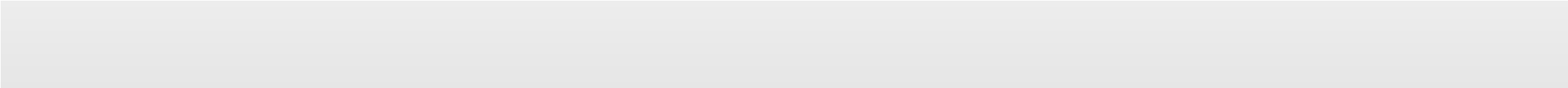
X // float(Y

)

3.0

# Literals and Conversions

* Python integers can be coded in hexadecimal, octal, and binary notation, in addition to the normal base-10 decimal
* Appropriate conversions can also be made



>>> 0

o1, 0x

01, 0b11111011

(1

, 1,

251)

>>>

oct(64), hex(64), bin

(64)

(

'0o100', '0x40', '0b1000000'

)

>>>

int('64'), int('100', 8), int('40', 16), int('1000000',

2)

(64

, 64, 64,

64)

>>>

eval('64'), eval('0o100'), eval('0x40'), eval('0b1000000'

)

(64

, 64, 64,

64)

>>>

'{0:o}, {1:x}, {2:b}'.format(64, 64,

64)

'100, 40, 1000000'

>>>

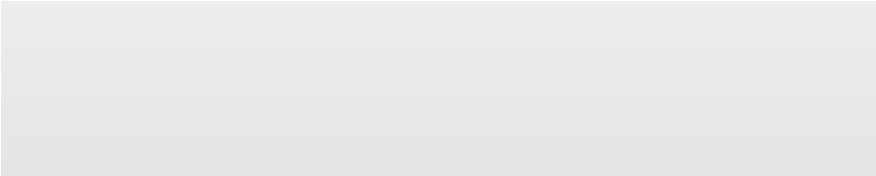
'%o, %x, %x, %X' % (64, 64, 255,

255)

'100, 40, ff, FF’

# Bitwise Operations

* Bitwise operators treat integers as strings of binary bits
* This is handy when dealing with network packets, serial ports and packed binary data produced by a C program



>>>

0b0001

X =



99

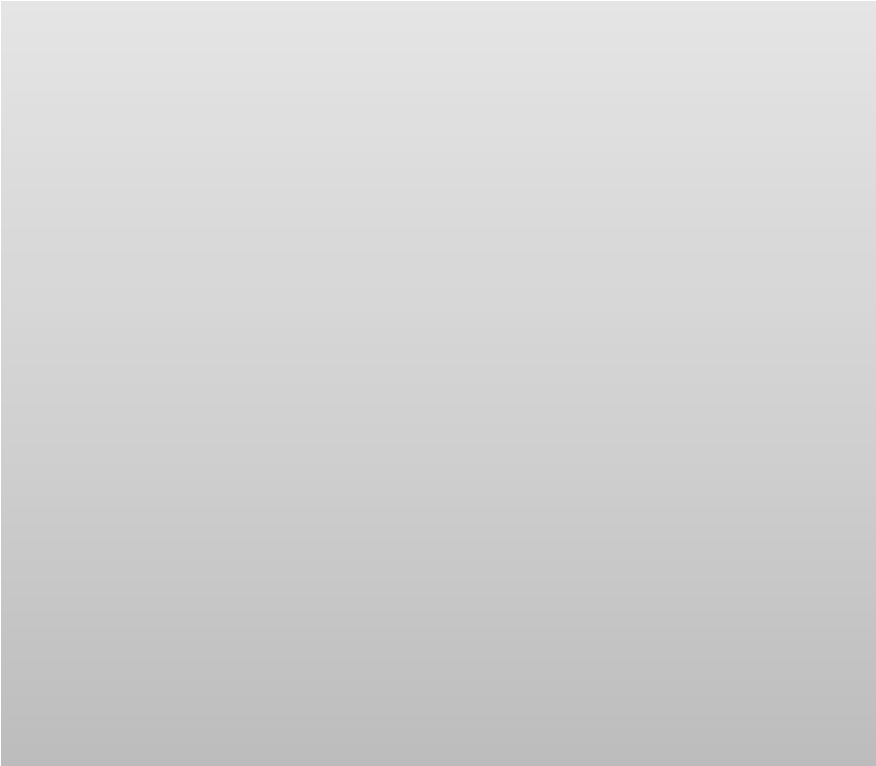
>>>

X =

>>>

bin(X), X.bit\_length(), len(bin(X))

-2



>>>

X

1

>>>

X <<

2

4

>>>

bin (X <<

2)

'0b100'

>>>

bin (X |

0b010)

'0b11'

1)

>>>

bin (X & 0b

'0b1'

>>>

X =

0

>>>

bin (X & 0b

1)

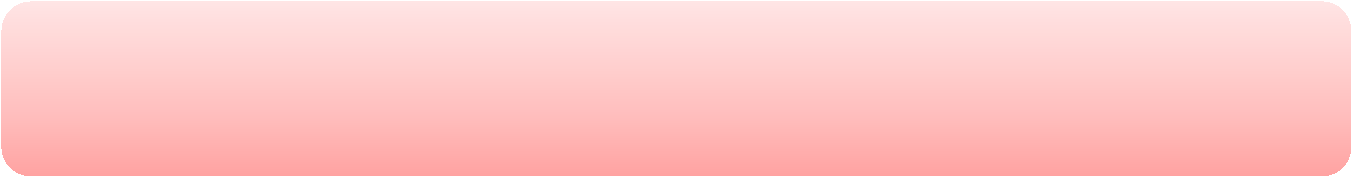
'0b0'



(

'0b1100011', 7,

7)



Allows you to query the number of bits required to

represent the number in concern

# Built-in Numeric Tools

• The built-in **math** module provides numerous number processing tools as below:



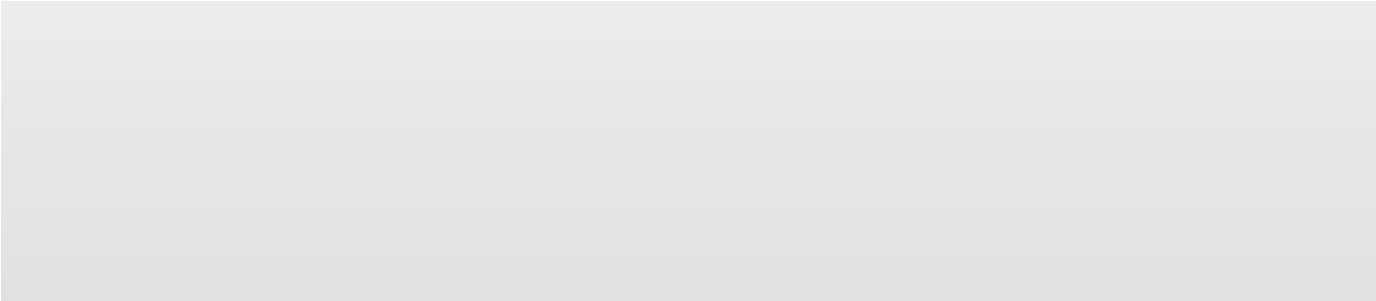
importmath

>>>

>>>

math.pi

3.141592653589793



>>>

abs

(-42.0)

42.0

>>>

min(3, 1, 7, 9), max(4, 7, 8,

2)



>>>

math.e

2.718281828459045

180)

>>>

math.sin(2\*math.pi/

0.03489949670250097

>>>

math.sqrt(144), math.sqrt

(2)

(12.0, 1.4142135623730951)

>>>

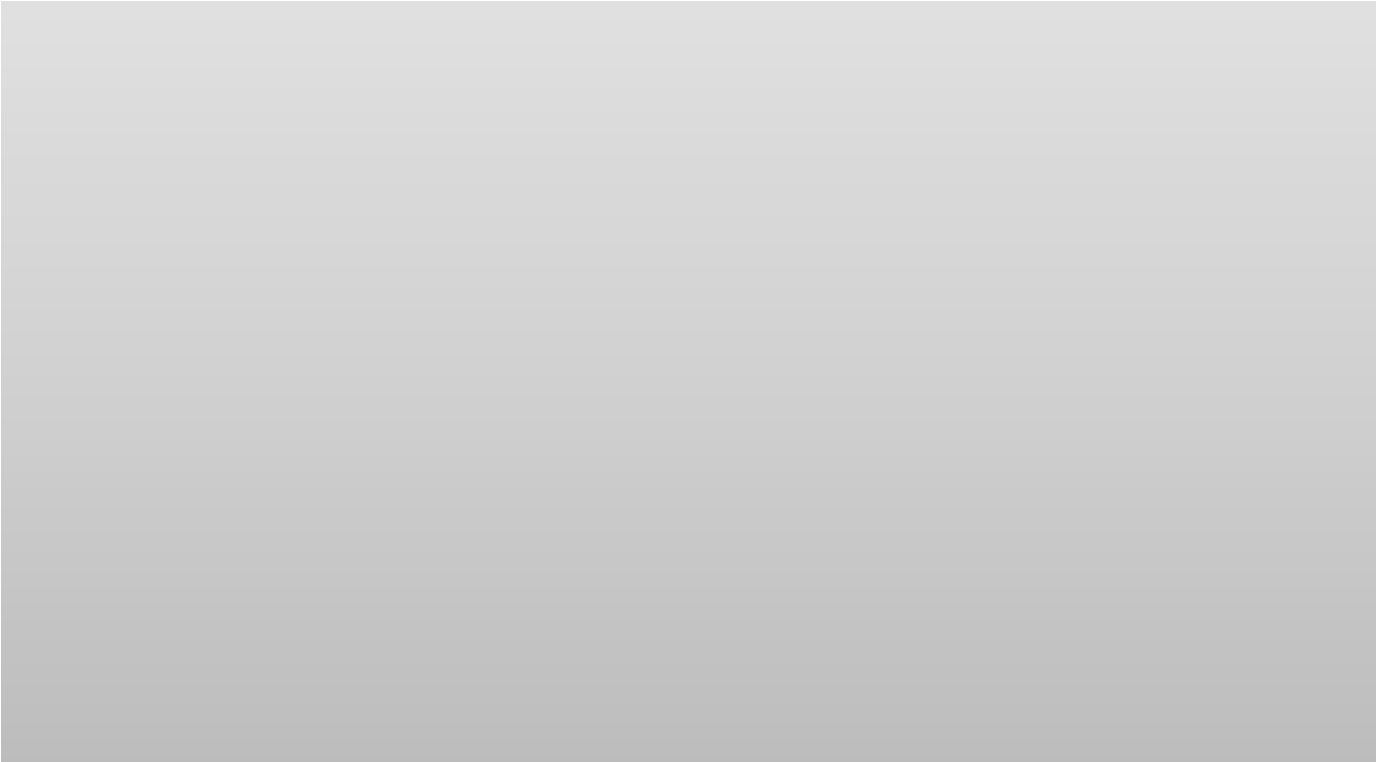
pow

(2,4), 2\*\*4, 2.0\*\*6.00

(16

, 16,

64.0)



(1

8)

,

>>>

math.floor(2.567), math.trunc

(-5.345)

(2

,

-5)

>>>

int(2.345), int

(-3.65)

(2

,

-3)

>>>

round

(2.768)

3

>>>

'%.1f' %

4.556

'4.6'

>>>

'{0:.2f}'.format

(5.567)

'5.57'

>>>

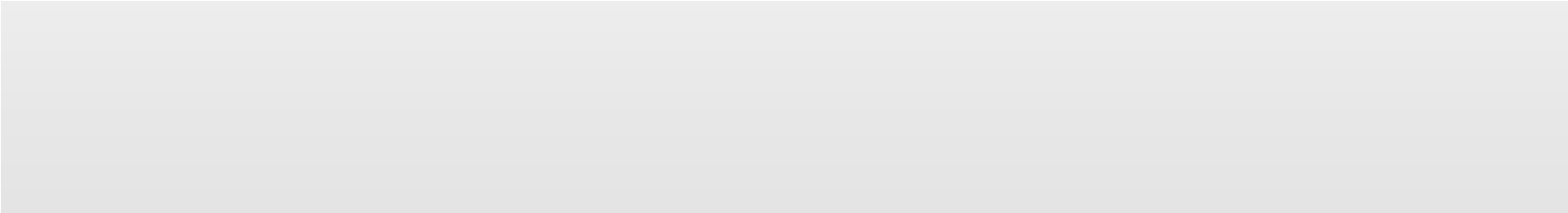
pow(324534,

0.5)

569.6788569009736

# Built-in Numeric Tools

• The standard library **random** module helps working with randomization



importrandom

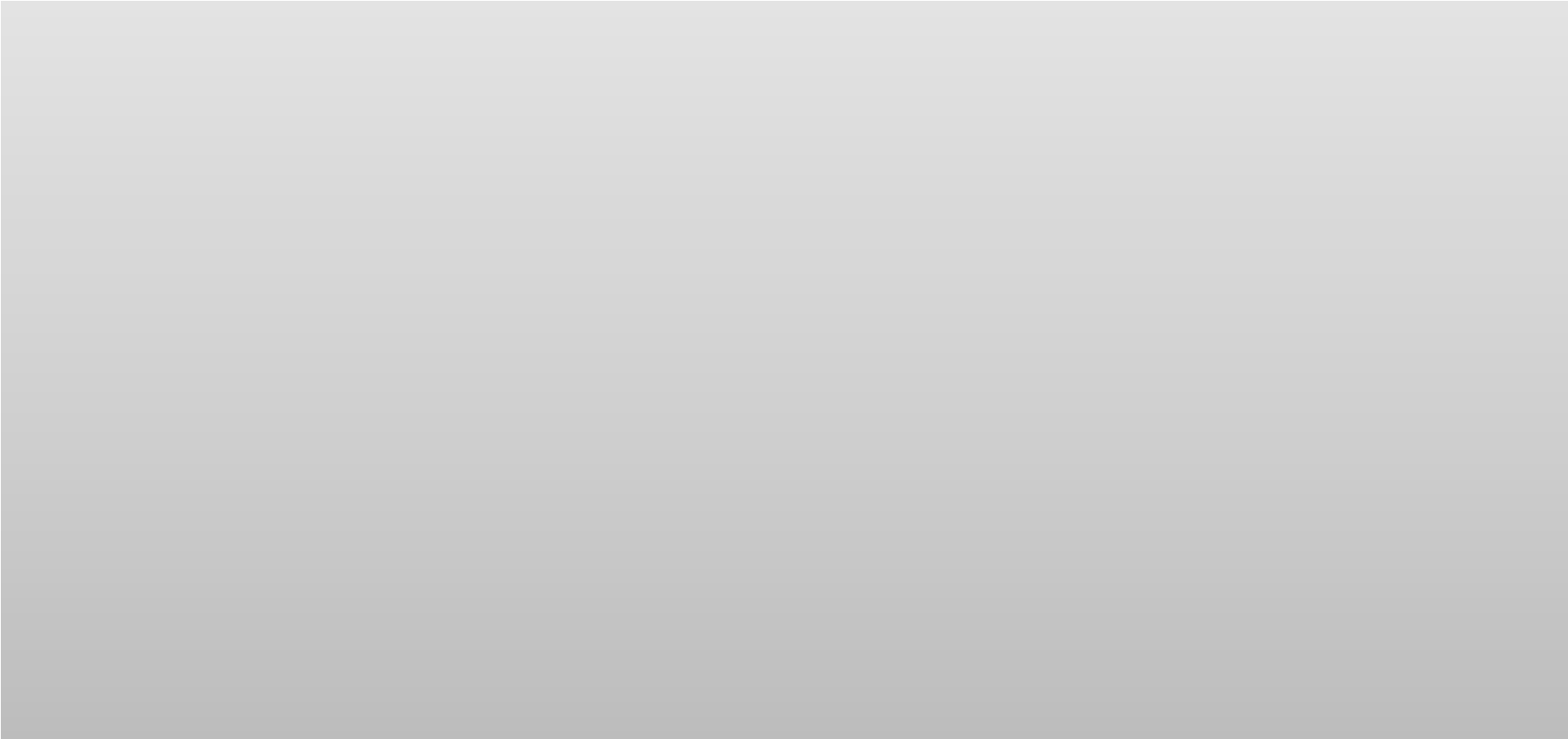
>>>

>>>

random.random

()

0.7550863994655612



>>>

random.random

()

0.2043633056246169

>>>

random.randint

(1,10)

4

>>>

random.randint(1,

10)

9

>>>

random.choice(['StarWars', 'StarTrek', 'Gravity'

])

'StarTrek'

>>>

random.choice(['StarWars', 'StarTrek', 'Gravity'

])

'StarWars'

>>>

suits= ['hearts', 'clubs', 'diamonds', 'spades'

]

>>>

random.shuffle(suits

)

>>>

suits

[

'diamonds', 'spades', 'hearts', 'clubs'

]

# Sets

* Besides decimals, Python 2.4 also introduced a new collection type, the *set—an unordered* collection of unique and immutable objects that supports operations corresponding to mathematical set theory
* By definition, an item appears only once in a set, no matter how many times it is added
* Accordingly, sets have a variety of applications, especially in numeric and database-focused work

# Sets



)

x = set('abcde'

>>>

y = set('dsfgh'

>>>

)

x

>>>

'a', 'c', 'b', 'e', 'd'

}

{

>>>

y

}

{

'h', 's', 'd', 'g', 'f'

x -y

>>>

}

'a', 'c', 'b', 'e'

{

>>>

x | y

}

{

'a', 'c', 'b', 'e', 'd', 'g', 'f', 'h', 's'



Creating a set



Using operator on sets



x & y

>>>

}

{

'd'

>>>

x ^ y

{

'a', 'c', 'b', 'e', 'g', 'f', 'h', 's'

}

>>>

x > y, x < y

(

False, False

)

>>>

'e' in x

True

>>>

'e' in 'elephant'

True

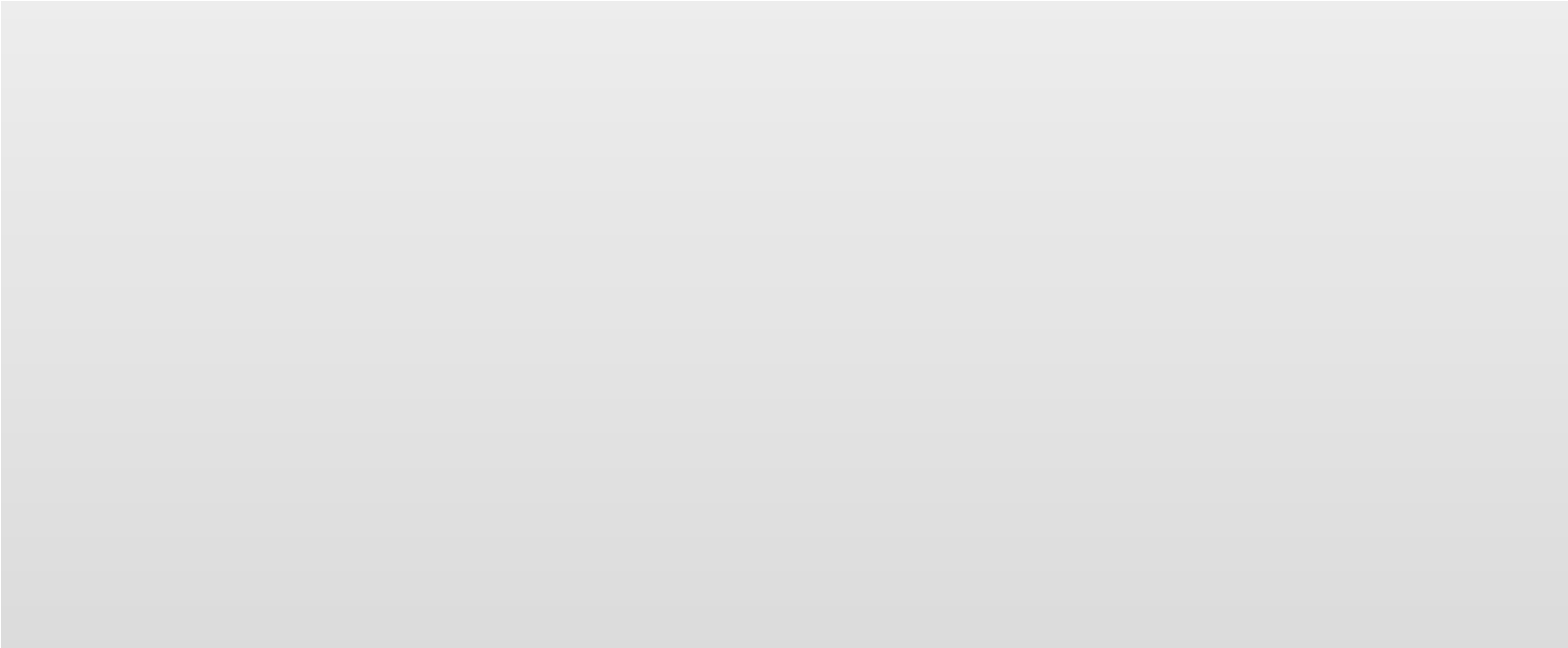
>>> 22

in [11, 22,

33]

True

# Sets



)

>>>

z = x.intersection(y

>>>

z

}

{

'd'

>>>

z.add('PY'

)

>>>

z

{

'PY', 'd'

}

>>>

z.update(set(['HUGO', 'BOSS'

]))

z

>>>

{

'HUGO', 'PY', 'd', 'BOSS'

}



Functions related to sets



>>>

z.remove

(

'd'

)

>>>

z

{

'HUGO', 'PY', 'BOSS'

}

>>>

foritemin set ('abc'): print(item\*

3)

aaa

ccc

bbb

>>>

s = set([1, 2,

3])

>>>

s | set([5 ,

6])

{1

, 2, 3, 5,

6}

>>>

s.union([6, 7,

8])

{1

, 2, 3, 6, 7,

8}

>>>

s.intersection((1, 3,

5))

{1

,

3}

>>>

s.issubset(range(-5,

5))

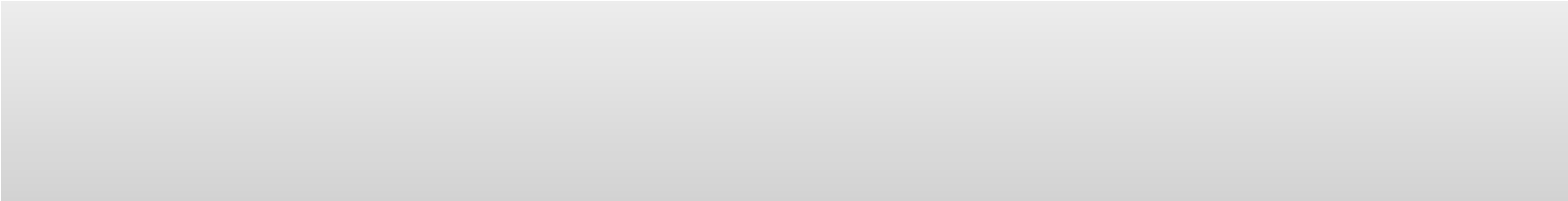
True

# String Basics

* From a functional perspective, strings can be used to represent just about anything that can be encoded as text or bytes.
* In the text department, this includes symbols and words (e.g., your name), contents of text files loaded into memory, Internet addresses, Python source code, and so on.
* Strings can also be used to hold the raw bytes used for media files and network transfers, and both the encoded and decoded forms of non-ASCII Unicode text used in internationalized programs.

# Using Single and Double Quotes

• In Python, single and double quoted strings are the same



>>>

'Python'

'Python'

>>>

"Python"

'Python'



>>>

title = "Life"' of '"Pi“

**#Intrinsic Concatenation**

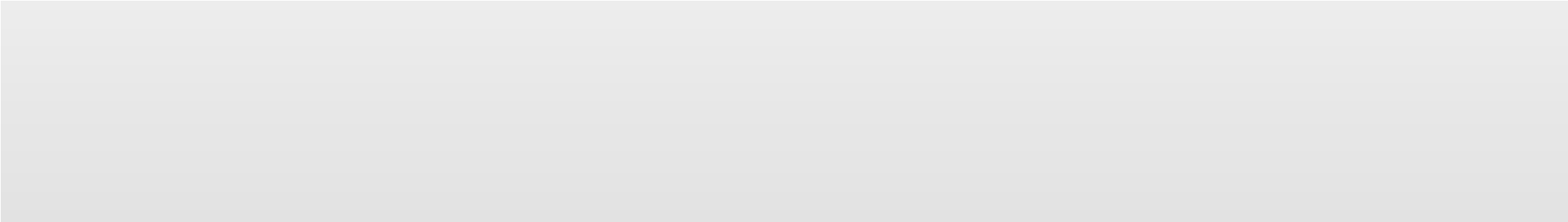
>>>

title

'Life of Pi'

# Triple Quoted Strings

• Triple quoted string literal, sometimes called the block string is a syntactic convenience for writing multi-line text data



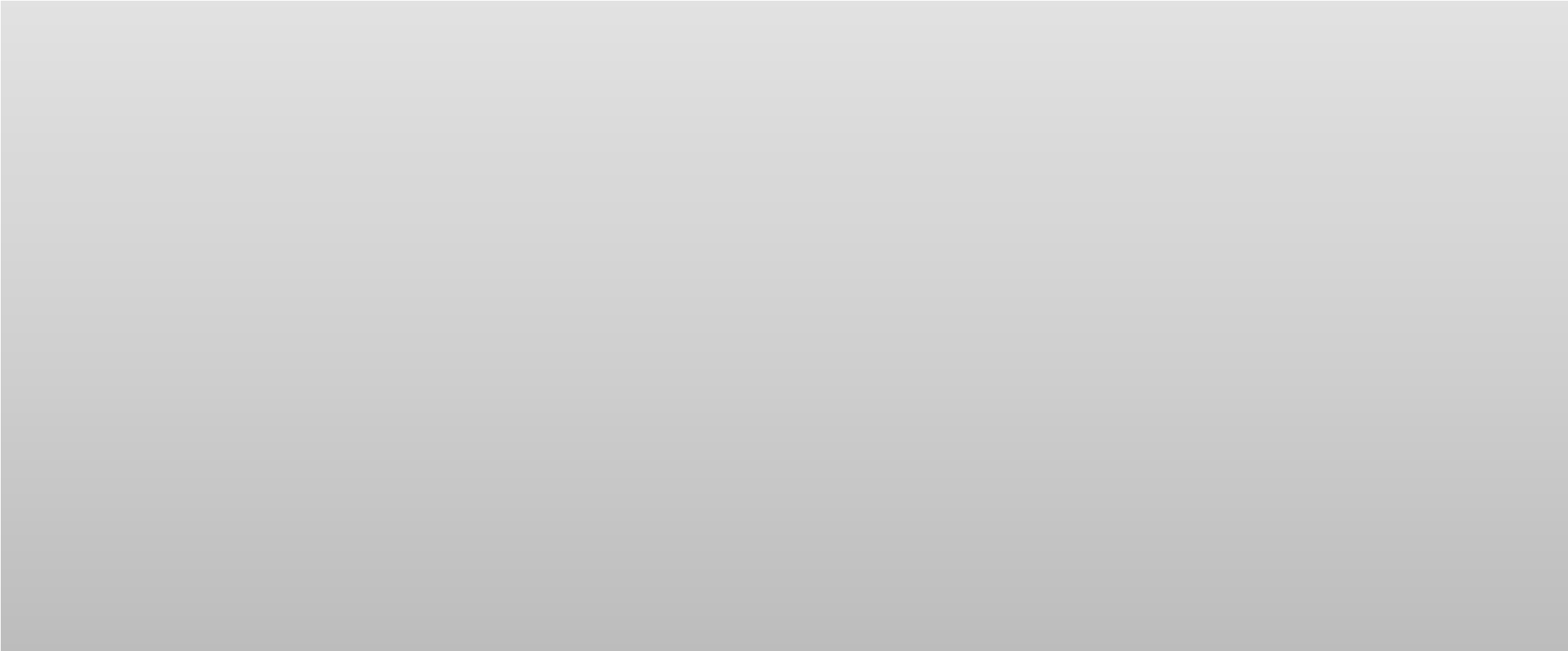
>>>

mantra = """Always

Look

At

The



Brighter

Side

Of

Life"""

>>>

mantra

'Always\nLook\nAt\nThe\nBrighter\nSide\nOf\nLife'

>>>

print(mantra

)

Always

Look

At

The

Brighter

Side

Of

Life

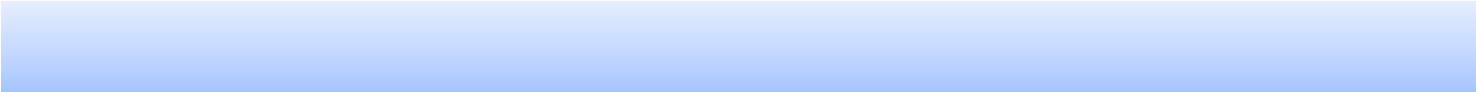
# Escape Sequences

|  |  |
| --- | --- |
| **Escape** | **Meaning** |
| \*newline* | *Ignored (continuation line)* |
| \\ | Backslash (stores one \) |
| \' | Single quote (stores ') |
| \" | Double quote (stores ") |
| \a | Bell |
| \b | Backspace |
| \n | Newline (linefeed) |
| \f | Form feed |
| \r | Carriage return |
| \t | Horizontal tab |
| \v | Vertical tab |

• Backslashes are used to introduce special character coding known as escape sequences

# Raw String

• Some times escape sequences can lead to problems, such as in this example:



myfile= open('C:\new\text.dat', 'w')



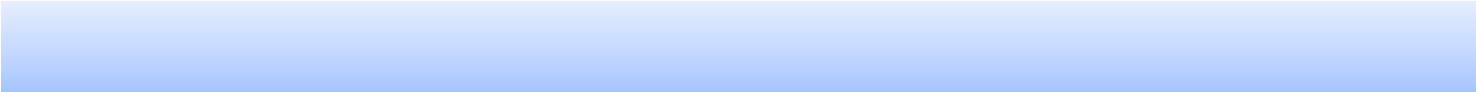
•

By prefixing ‘r’ the string is converted to a raw s

tring

•

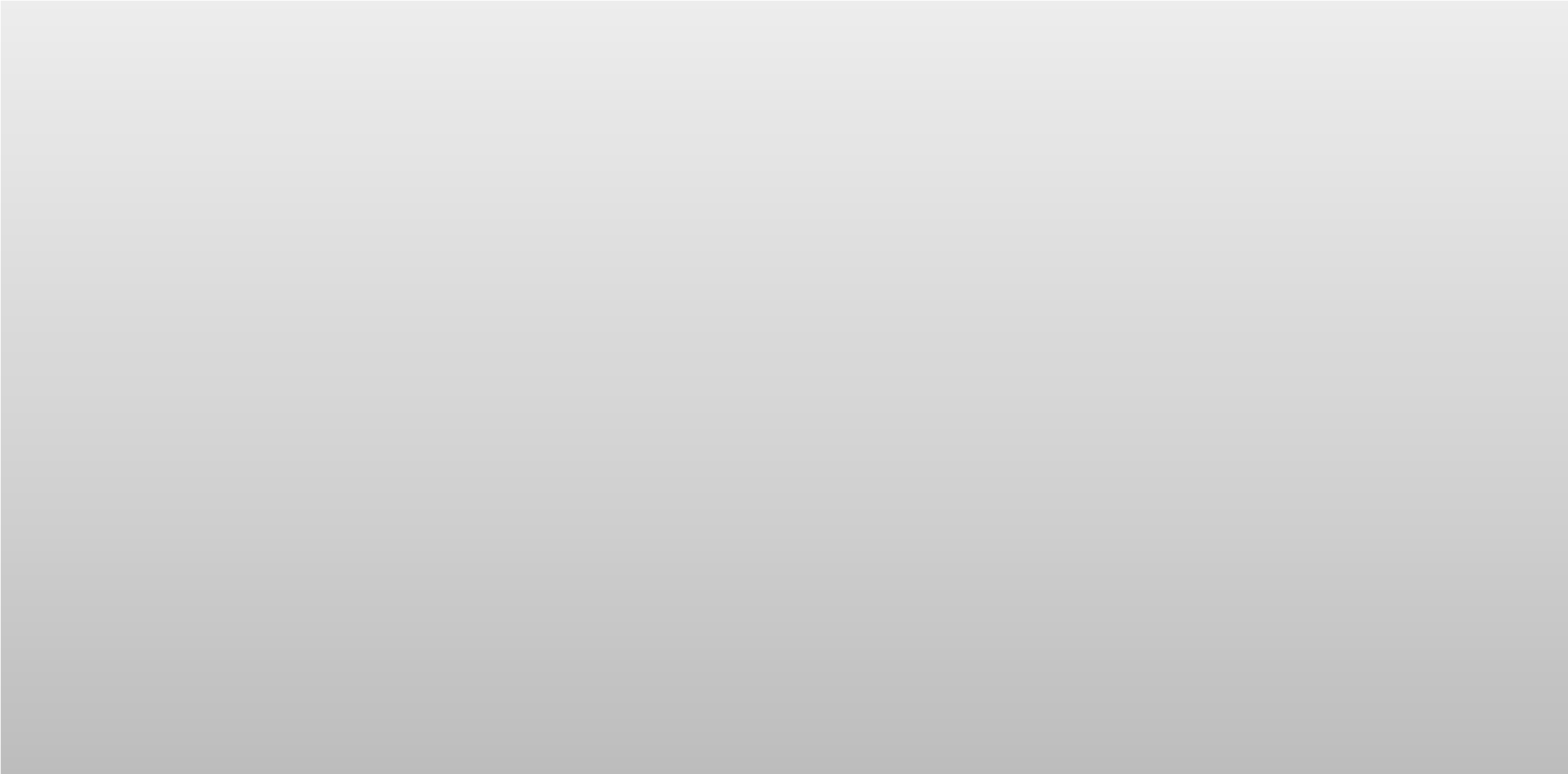
Raw strings ignore the escape sequences literally



myfile= open(r'C:\new\text.dat', 'w')

# Basic String Operations

* Length of the string using **len()**
* Concatenation using the **+** operator
* Repetition using **\*** operator
* Step through using **for**
* Searching for a character or group of characters using **in**



>>>

string = 'OptimusPrime'

>>>

len(string

)

13

>>>

string = string + ' -AutobotLeader'

>>>

string

'OptimusPrime -AutobotLeader'

>>>

'Hi!' \*

8

'Hi!Hi!Hi!Hi!Hi!Hi!Hi!Hi!‘

>>>

for iin string: print(i, end = ' '

)

O p t im u s P r im e -

A u t o b o t L e a d e r

>>>

'p' in string

True

>>>

'x' in string

False

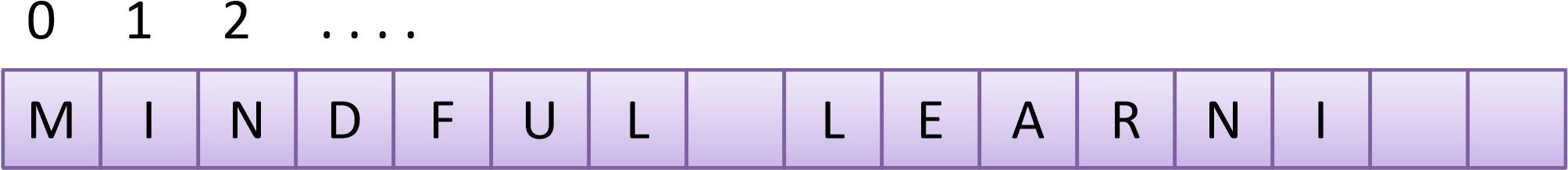
>>>

'Prime' in string

True

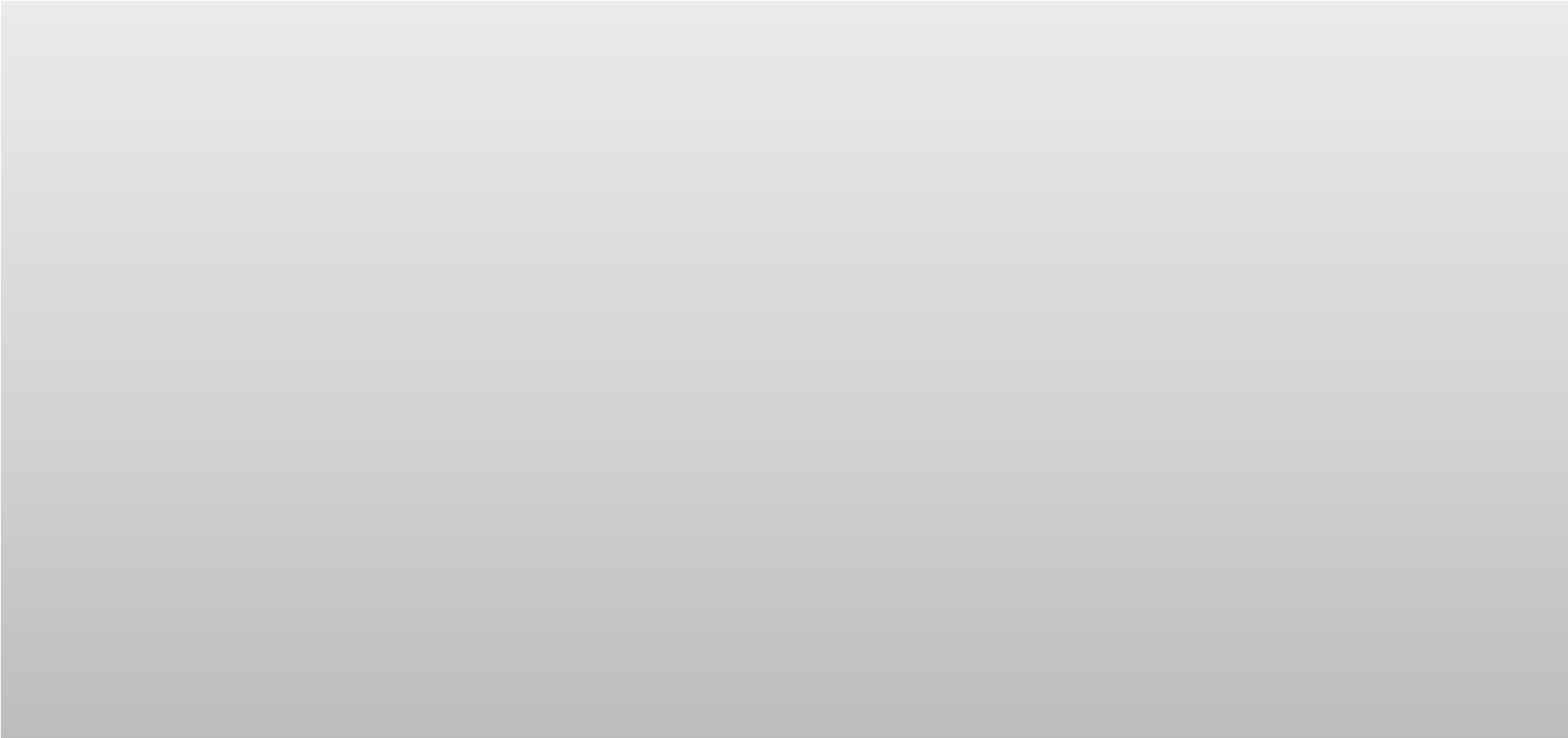
# Indexing and Slicing

• Structure of a string:

 1 . . . .

N G

-2 -1



>>>

string = 'Mindful Learning‘

>>>

string[0], string[8:16], string

[0:16:2]

(

'M', 'Learning', 'MnflLann'

)

>>>

string[::4]

**# Skipping charecters**

'MfLn'

>>>

string

[::-1]

'gninraeLlufdniM'

>>>

string

[8:]

'Learning’

>>>

string

[-5:-1]

'rnin'

>>>

string[slice(0, 8)]

**# slice()**

'Mindful ‘

>>>

string[8:0:-1]

**# Reverse from8 downto0**

‘L lufdni’

>>>

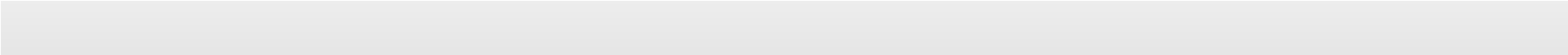
string[0:8:-1]

**# Doesn’twork**

''

# String Conversion Tools

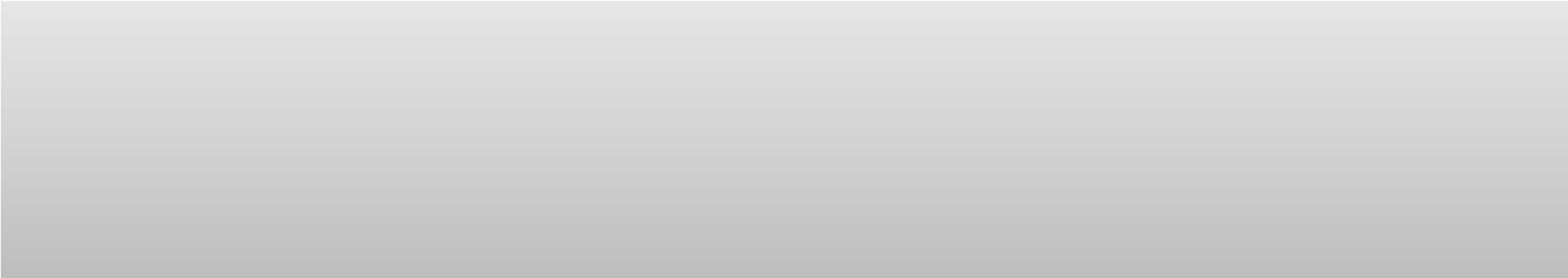
* Convert from/to string using **int(), float()** and **str()**
* Convert to as-code string using **repr()**
* Convert from/to ASCII using **ord()** and **chr()**



5

>>>

'42' +



Traceback(most recent call last):

File "<pyshell#14>", line 1, in <module>

'42' + 5

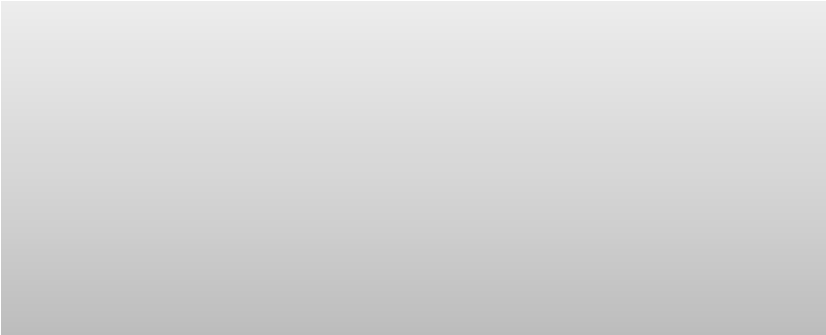
TypeError: Can't convert 'int' object to strimplicitly

>>>

'42' + str

(5)

'425'



>>>

str= '42'

>>>

int(str) +

5

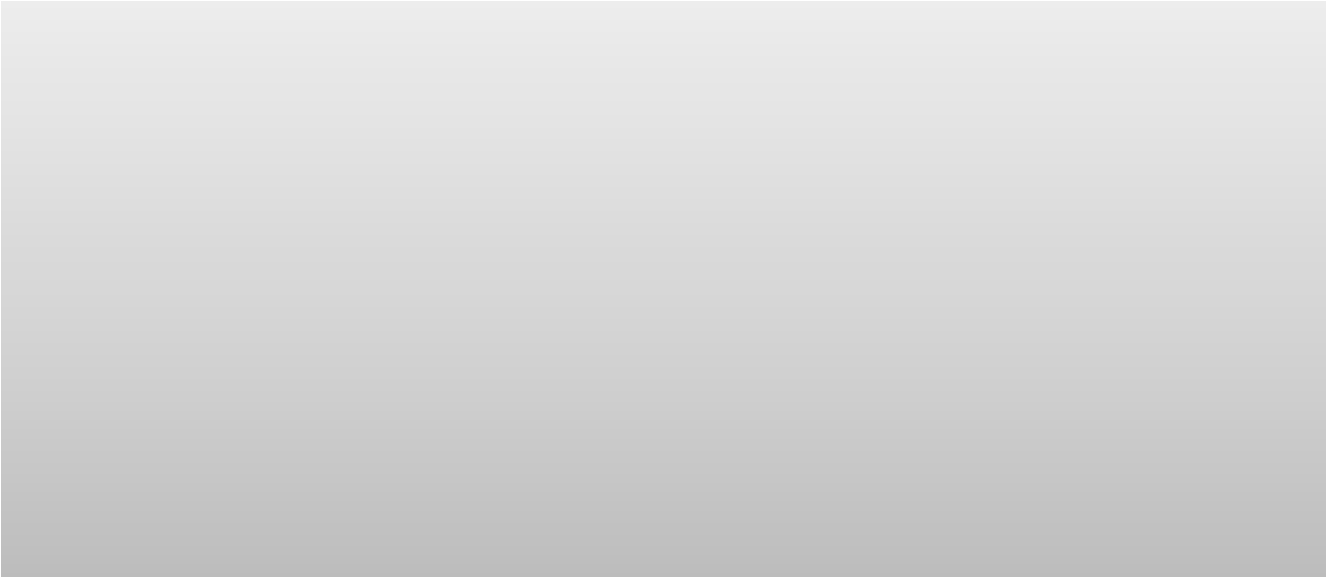
47

repr

>>>

(42)

'42'



>>>

text = '1.234E-10'

)

>>>

float(text

1.234e-10

>>>

repr(text

)

"'1.234E-10'"

>>>

ord('P'

)

80

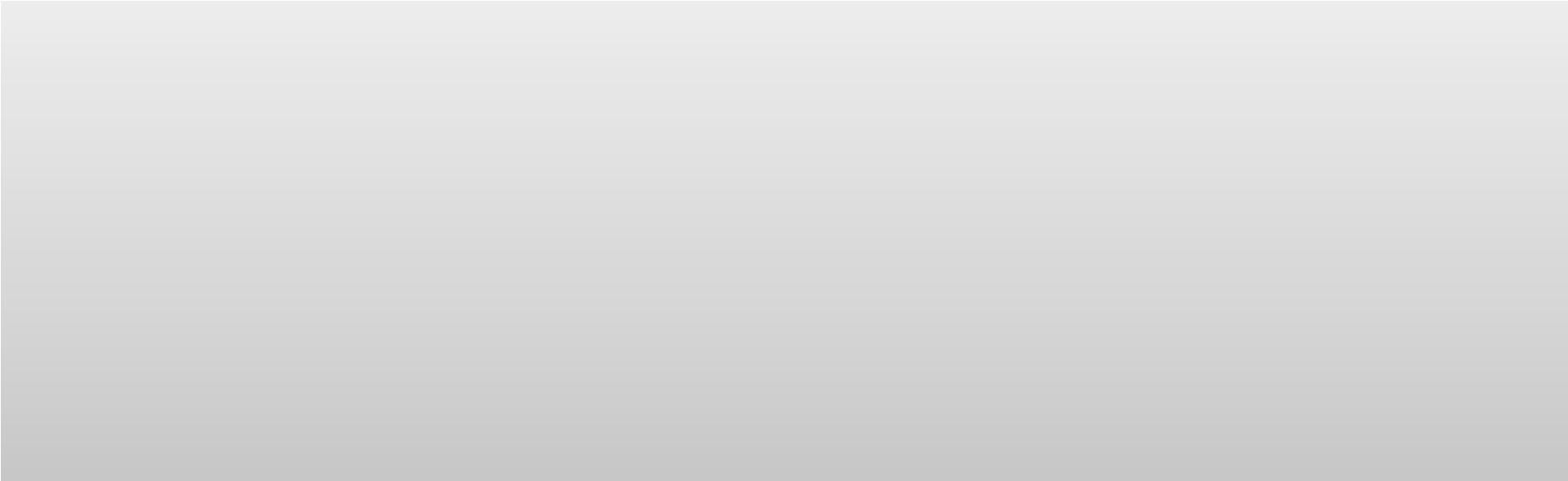
>>>

chr

(80)

'P'

# String Conversion Tools



>>>

string = '42'

>>>

int(string) +

5

47

>>>

string = '42A'

)

>>>

int(string

Traceback(most recent call last):

File "<pyshell#32>", line 1, in <module>

int(string)

ValueError: invalid literal for int() with base 10: '42A'



>>>

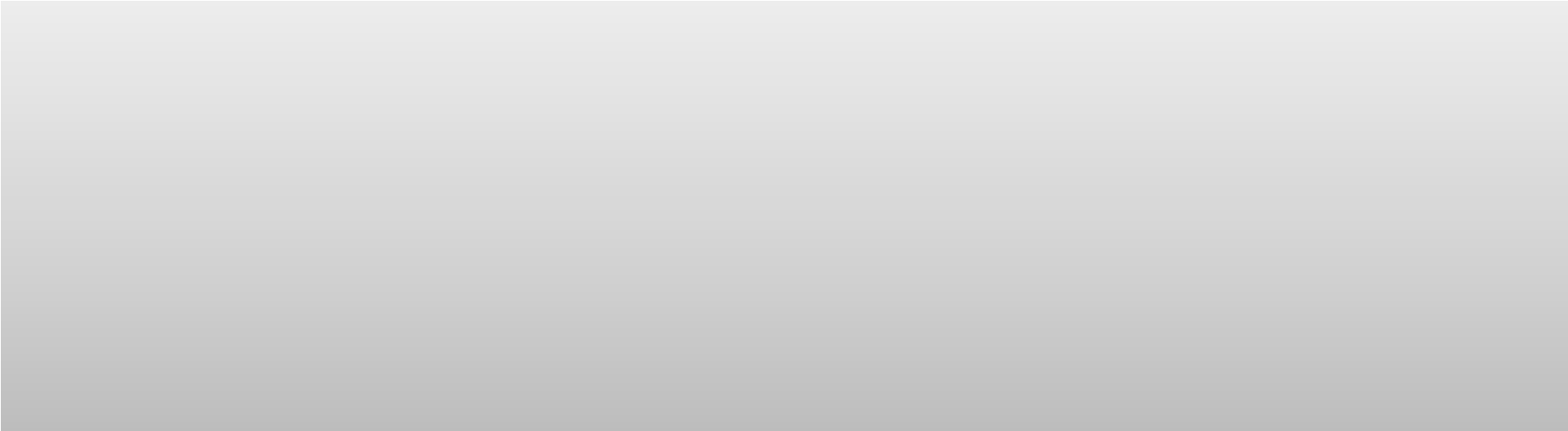
repr

(

string

)

"'42A'"



>>>

string = '6.445‘

>>>

string

+ 3.0

Traceback(most recent call last):

File "<pyshell#36>", line 1, in <module>

string + 3.0

TypeError: Can't convert 'float' object to strimplicitly

>>>

float(string)

+ 3.0

9.445

# Format Specifiers

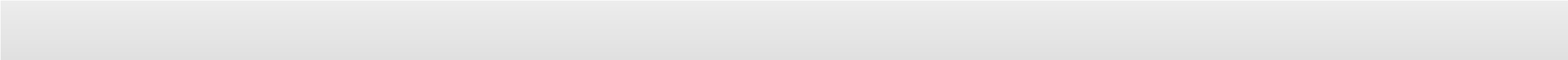
• Format specifiers are special strings that specify how data is represented or printed

•

Examples: %d, %4.3f, %s



Old style!



import sys

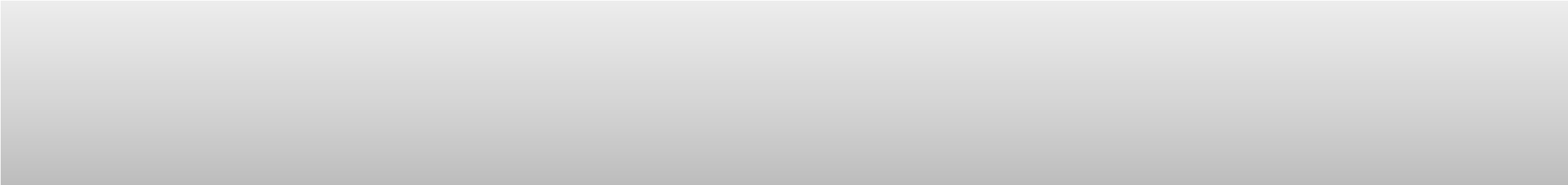
>>>



•

Python has a new style which uses the

**format()**



)

>>>

'{2}, {1}, {0}'.format('a', 'b', 'c'

'c, b, a'

>>>

'Coordinates: {lat}, {long}'.format(lat='37.24N', long='-115.81W'

)

'Coordinates: 37.24N, -115.81W'



}

>>>

'My %(kind)s runs %(platform)s' % {'kind': 'laptop', 'platform': sys.platform

'My laptop runs win32‘

>>>

'My %(kind)-8s runs %(plat)8s' % dict(kind='laptop', plat=sys.platform

)

'My laptop runs win32'

# Format String Syntax



format\_spec::= [[fill]align][sign][#][0][width][,]

[

.precision][type

]

fill ::= <any character>

align ::= "<" | ">" | "=" | "^"

sign ::= "+" | "-" | " "

width ::= integer

precision ::= integer

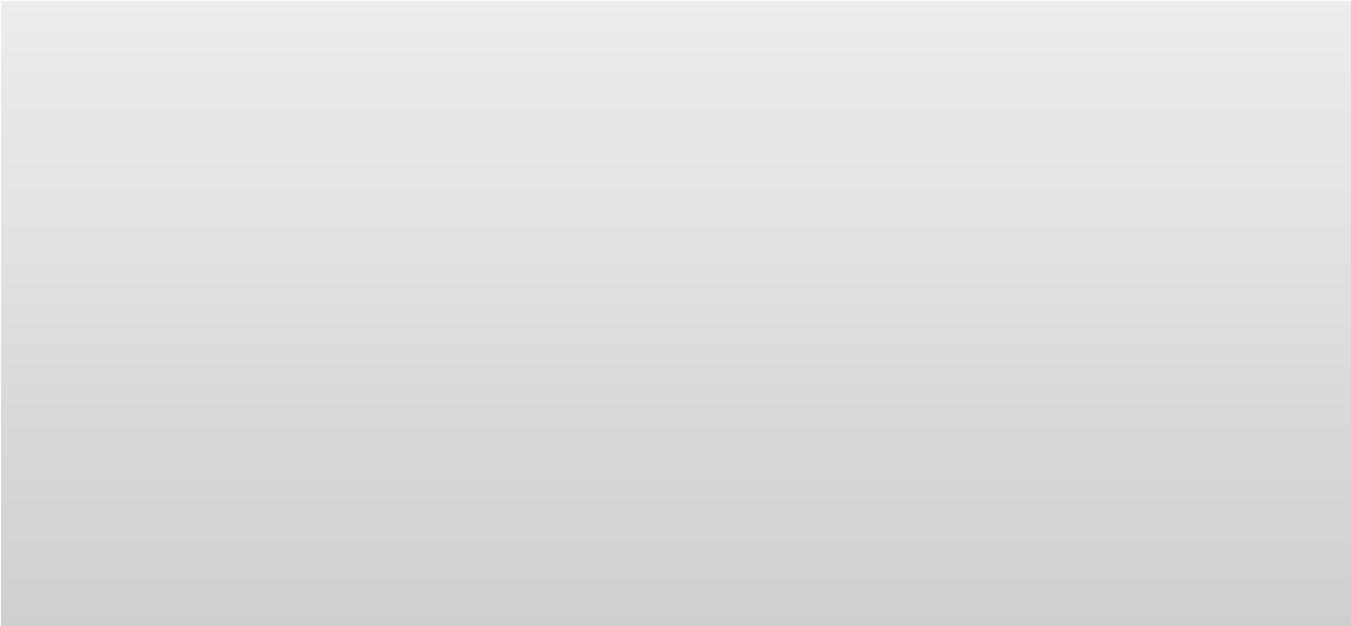


type ::= "b" | "c" | "d" | "e" | "E" | "f"

| "F" | "g" | "G" |

"n" | "o" | "s" | "x" | "X" | "%"

# Examples using **format()**



)

'{:<30}'.format('Mindful Learning'

>>>

'Mindful Learning '

'{:>30}'.format('Mindful Learning'

>>>

)

' Mindful Learning'

)

'{:^30}'.format('Mindful Learning'

>>>

' Mindful Learning '

'{:\*^30}'.format('Mindful Learning'

>>>

)

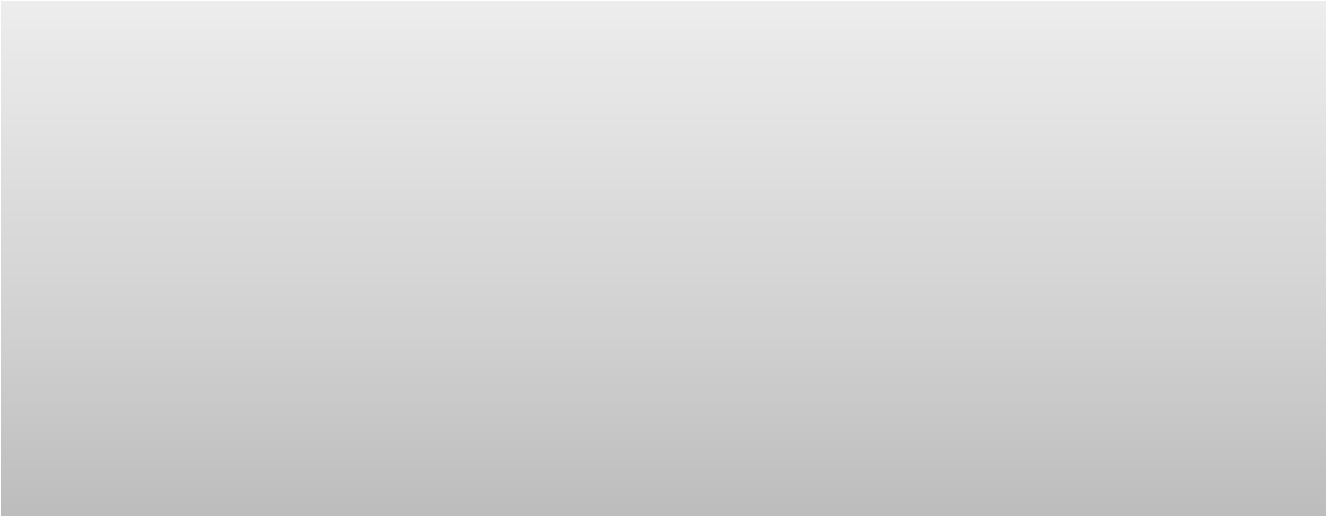
'\*\*\*\*\*\*\*Mindful Learning\*\*\*\*\*\*\*'

(3.14, -3.14)

'{:+f}; {:+f}'.format

>>>

'+3.140000; -3.140000‘



(1234567890)

'{:,}'.format

>>>

'1,234,567,890‘

(296999.2567)

'{:,.2f}'.format

>>>

'296,999.26'

= 200

>>>

runs

>>>

overs =

17

)

>>>

'Average: {:.2%}'.format(runs/overs

'Average: 1176.47%'



>>>

nlist

=

[1, 2,

3]

>>>

'{0:s}', format(nlist

)

(

'{0:s}', '[1, 2, 3]'

)

>>>

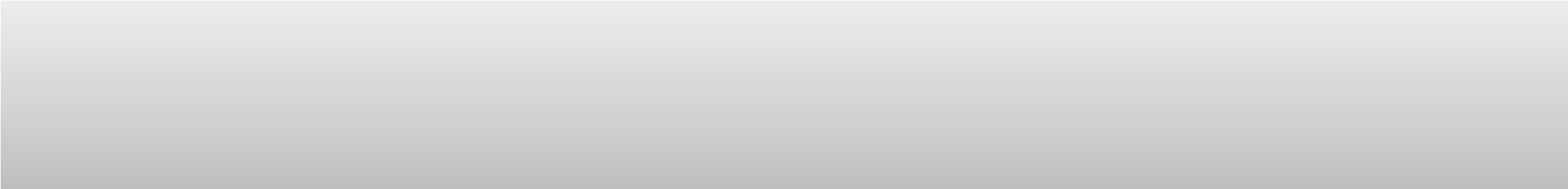
'{0:s}'. format(nlist

)

'[1, 2, 3]'



Why?



>>>

import datetime

>>>

d = datetime.datetime(2010, 7, 4, 12, 15,

58)

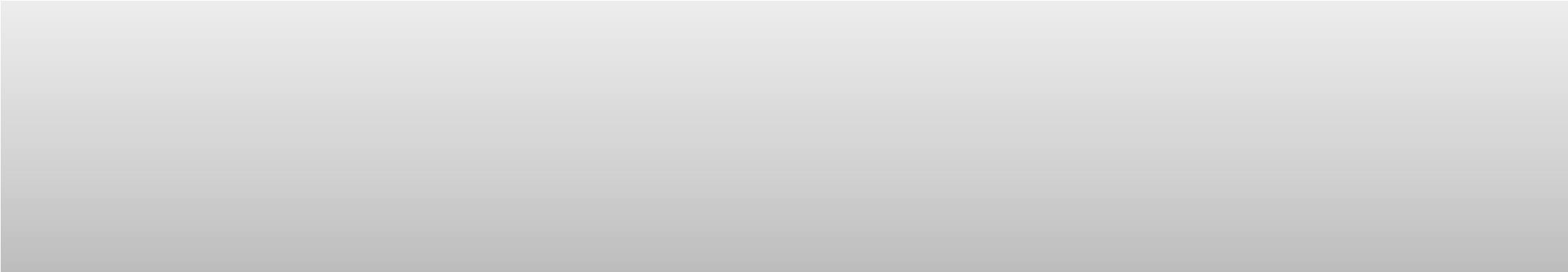
>>>

'{:%Y-%m-%d %H:%M:%S}'.format(d

)

'2010-07-04 12:15:58'

# Examples using **format()**



>>>

X = '{motto}, {0} and {food}'.format(42, motto=3.14, food=[1,

2])

>>>

X

'3.14, 42 and [1, 2]'

Y = X.replace('and', 'but under no circumstances'

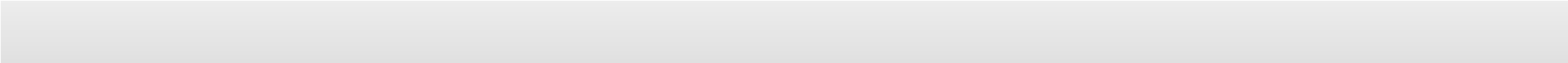
>>>

)

Y

>>>

'3.14, 42 but under no circumstances [1, 2]'



>>>

import sys



>>>

'My {map[kind]} runs

{

sys.platform

}

'.format(sys=sys, map={'kind': 'laptop'

})

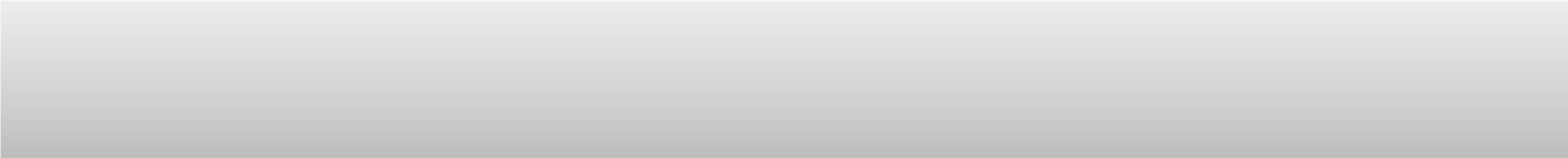
'My laptop runs win32‘

>>>

'{0.platform:>10} = {1[kind]:<10}'.format(sys, dict(kind='laptop'

))

' win32 = laptop '



>>>

template = '{first} {second}'

>>>

template.format(first = 'Don\'t be afraid of being slow', second = ',be afraid of

staying still');

"Don't be afraid of being slow ,be afraid of staying still"

# String Methods

* S.capitalize() • S.rfind(sub [,start [,end]])
* S.ljust(width [, fill]) • S.find(sub [, start [, end]])
* S.casefold() • S.rindex(sub [, start [, end]])
* S.lower() • S.format(fmtstr, \*args, \*\*kwargs)
* • S.rjust(width [, fill])

S.center(width [, fill])



•

S.lstrip

([

chars

])

•

S.count(sub [, start [, end]])

•

S.maketrans(x[, y[, z]])

•

S.encode([encoding [,errors]])

•

S.partition(sep)

•

S.replace(old, new [, count])

•

S.expandtabs([tabsize])

•

S.isdecimal()

•

S.isalpha()

•

S.isalnum()

•

S.rpartition(sep)

•

S.rsplit([sep[, maxsplit]])

•

S.rstrip([chars])

•

S.split([sep [,maxsplit]])

•

S.endswith(suffix [, start [, end]])

•

S.isdigit()

•

S.splitlines([keepends])

•

S.isidentifier()

•

S.startswith(prefix [, start [, end]])

* S.index(sub [, start [, end]])

# String Methods

* S.islower()
* S.strip([chars])
* S.isnumeric() S.translate(map)

•

S.swapcase()

•

S.isprintable()



•

S.title

()

•

S.isspace()

•

•

S.istitle()

•

S.upper()

•

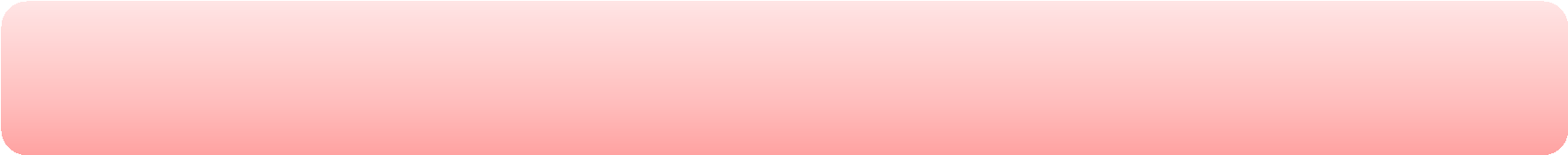
S.isupper()

•

S.zfill(width)

•

S.join(iterable)



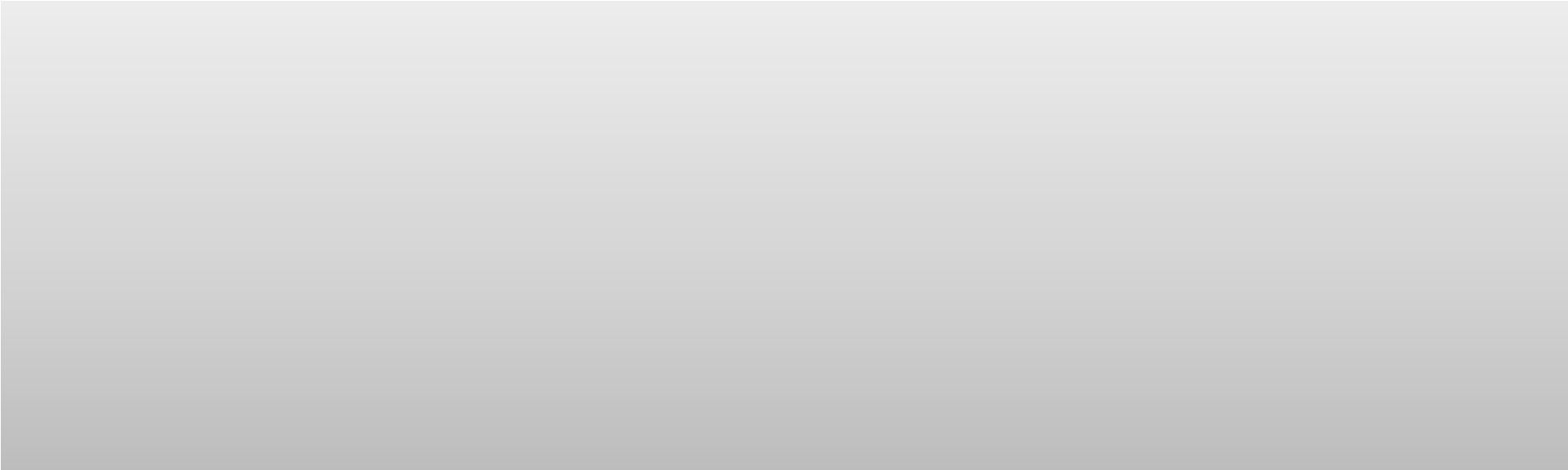
See python documentation for more details

# Changing Strings

* A string is an immutable sequence What changes to make – It cannot be changed in place in a string?



* To change a string, you generally need to build and assign a new string using tools such as concatenation and slicing and then, if desired, assign the result back to the string’s original name



>>>

string = 'Python'

>>>

string[0] = 'J'

Traceback (most recent call last):

File "<pyshell#47>", line 1, in <module>

string[0] = 'J'

TypeError: 'str' object does not support item assignment

>>>

string = 'J' + string

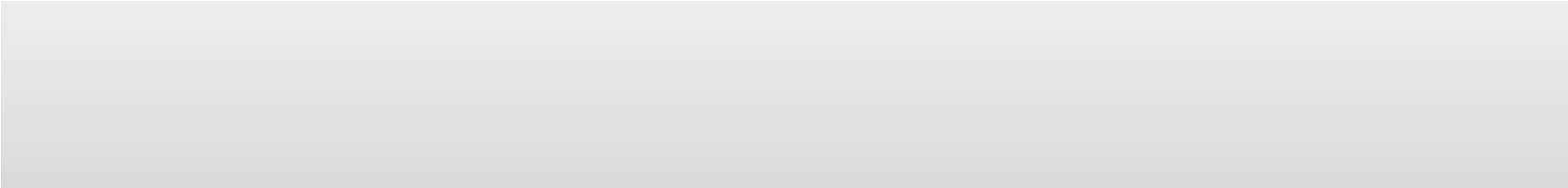
[1:]

>>>

string

'Jython'

# Changing Strings



string = 'PanAir'

>>>

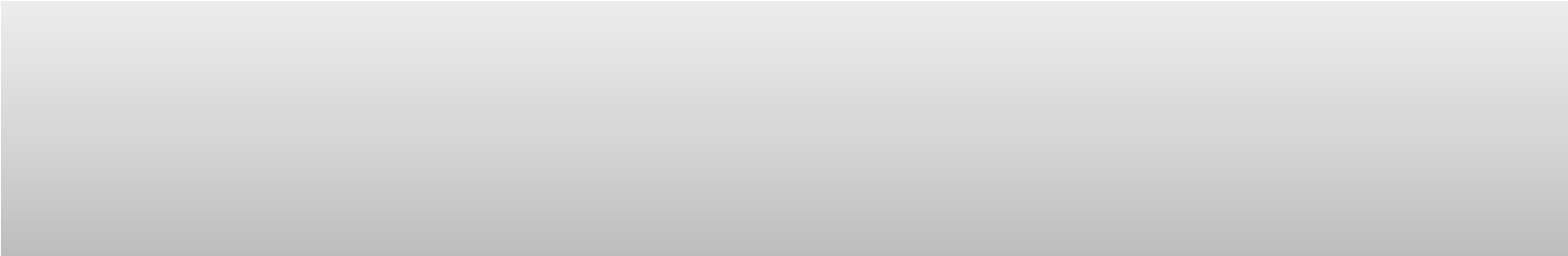
)

string = string.replace('Pan', 'Con'

>>>

>>>

string



)

'Python is %d %s language' % (1, 'amazing'

>>>

'Python is 1 amazing language'

)

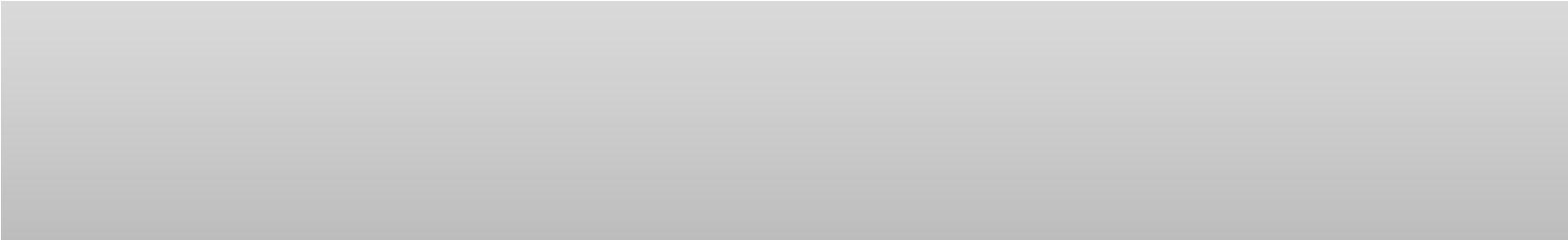
'Python is {0} {1} language'.format(1, 'amazing'

>>>

'Python is 1 amazing language'



Replacing



'ConAir‘

>>>

string = "Astlavista Babe!”

>>>

string.find("vista"

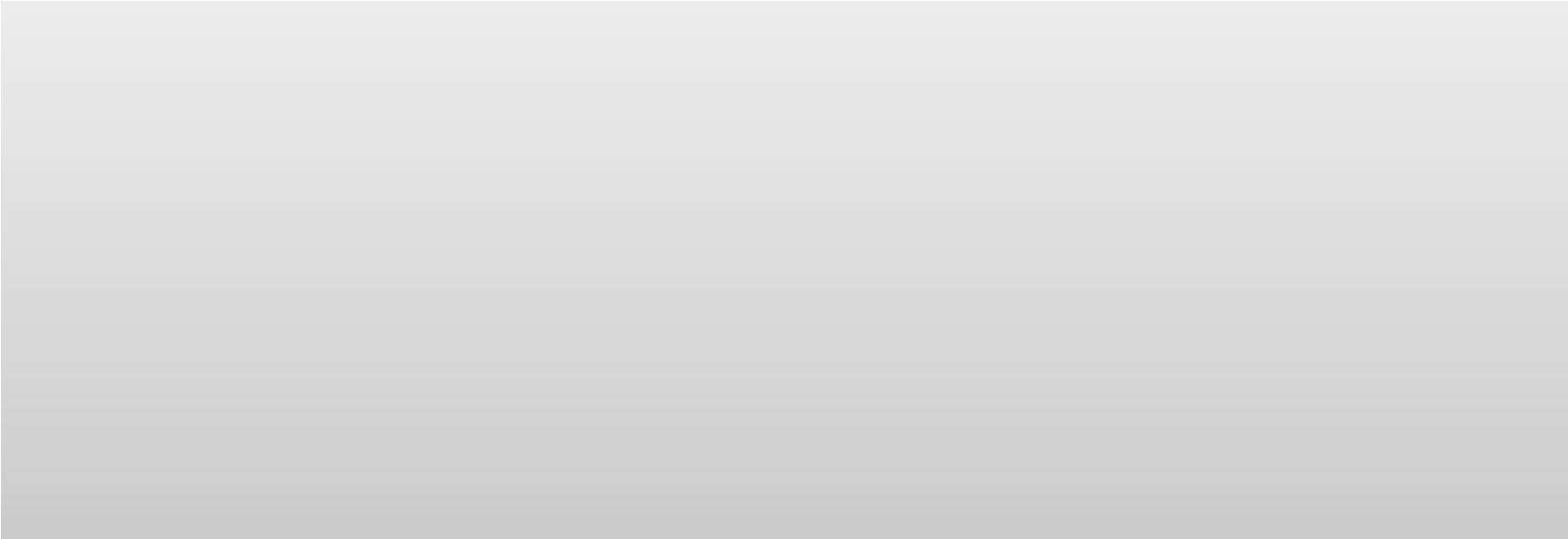
)

5



Finding

# Changing Strings



>>>

string = 'Be mindful when you learn python!'

>>>

where = string.find('mindful'

)

>>>

where

3

()

string.split

>>>

[

'Be', 'mindful', 'when', 'you', 'learn', 'python!'

]

>>>

)

string.split('e'

[

'B', ' mindful wh', 'n you l', 'arnpython!'

]

]

words = ['Live', 'Life', 'King', 'Size'

>>>



Splitting



Joining



)

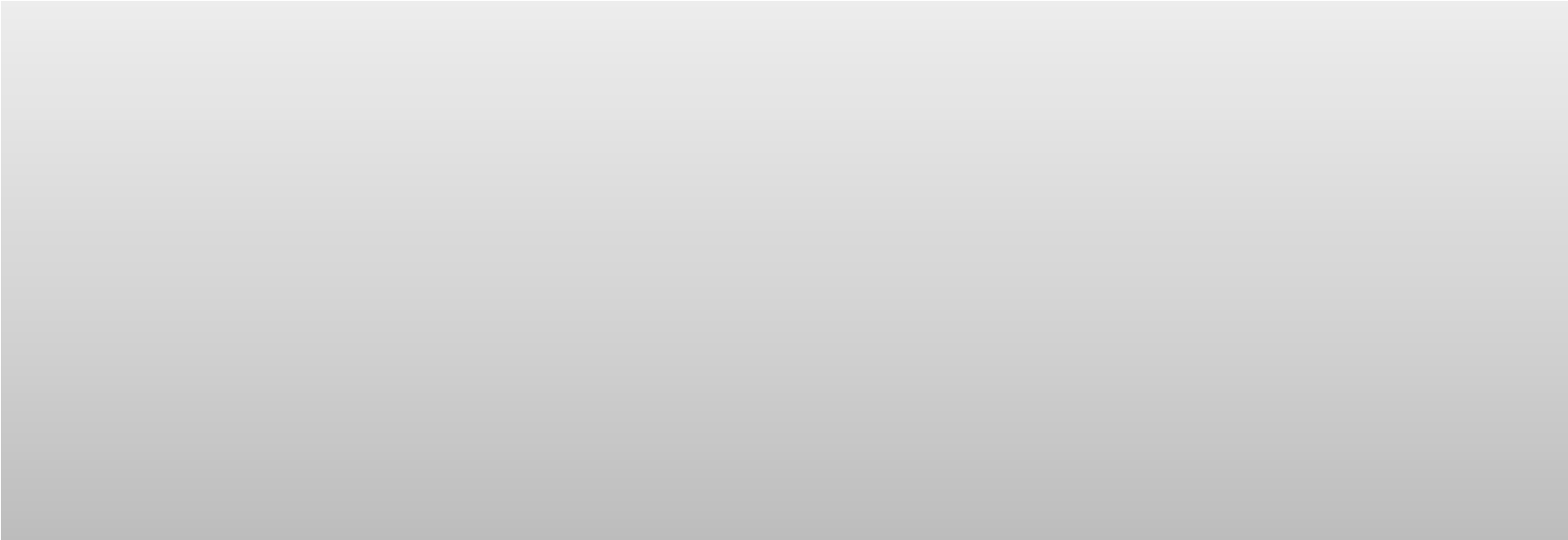
>>>

sentence = ' '.join(words

sentence

>>>

'Live Life King Size'



sentence = 'Silence is Alchemy!'

>>>

>>>

sentence.upper

()

'SILENCE IS ALCHEMY!'

>>>

sentence.startswith('Si'

)

True

>>>

sentence.endswith('!\n'

)

False

>>>

'is' in sentence

True



Refer

https://docs.python.org/2/library/string.html#

for more details

# Input and Output

• Python has two functions designed for accepting data directly from the user:

* input()
* raw\_input()

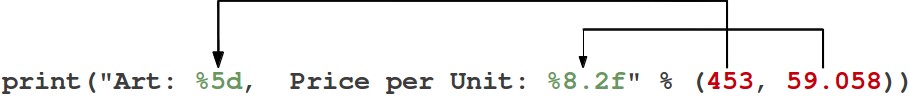


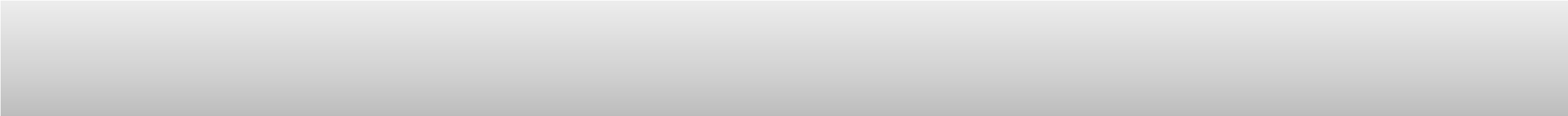
•

The basic way to do output is to use the print()

statement

# **print()**





>>>

print ("Art: %5d, Price per unit: %8.2f" %

(453

, 59.08))

Art: 453, Price per unit: 59.08

Specifying a Final String in **print()**

* By default every **print()** terminates with ’\n’
* It can be changed using **end=** parameter to the print statement

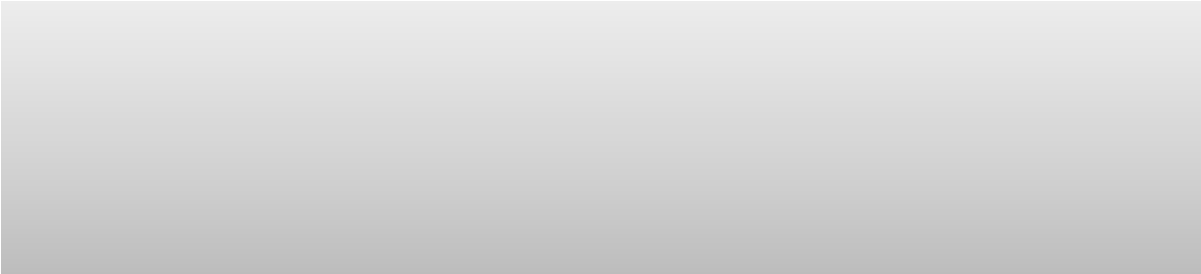


# print01.py

print('I hear and I forget')

print('I see and I remember')

print('I do and I understand')

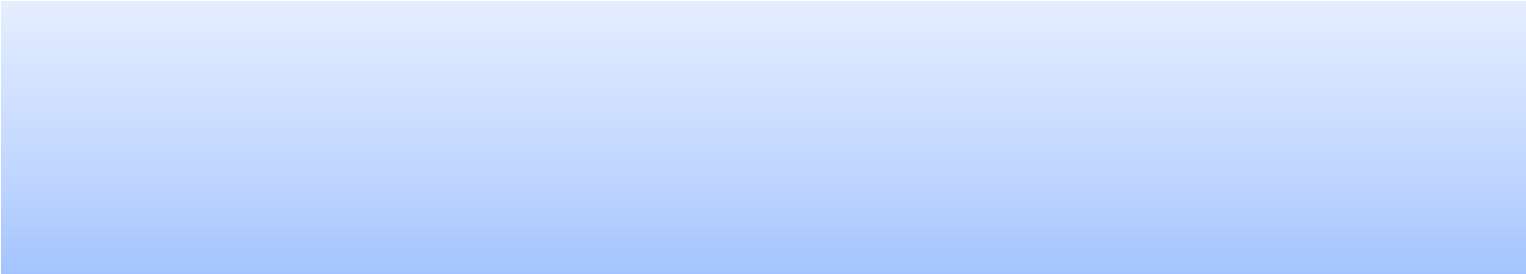


>>>

I hear and I forget

I see and I remember

I do and I understand



# print01.py

print('I hear and I forget', end='.')

print('I see and I remember', end='.')

print('I do and I understand', end='.')

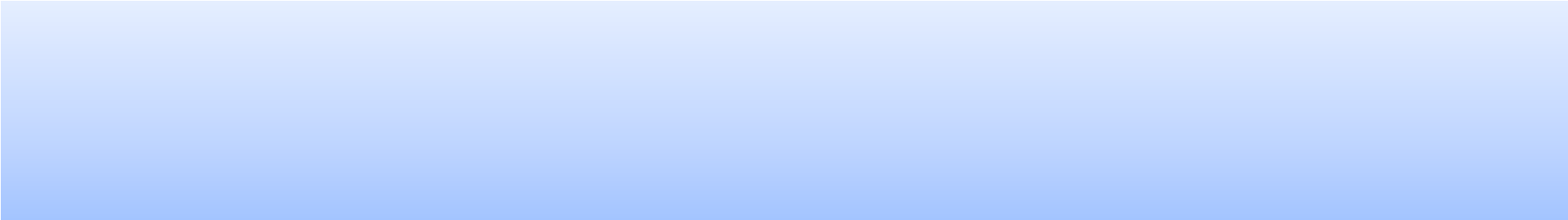


>>>

I hear and I forget.Isee and I remember.Ido and I understand.

# **raw\_input()**

* **raw\_input()** asks the user for a string of data (ended with a newline), and simply returns the string.
* It can also take an argument, which is displayed as a prompt before the user enters the data.



integer1 = raw\_input('Enter the first number: ')

integer1 = int(integer1)

integer2 = raw\_input('Enter the second number: ')

integer2 = int(integer2)

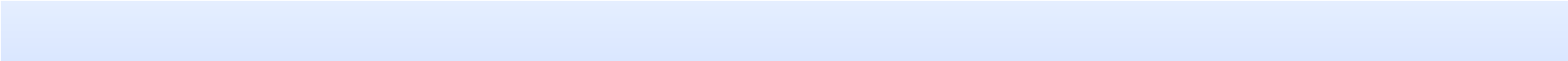


**raw\_input()**

is discarded in python 3.x

# **input()**

• The function then reads a line from input, converts it to a string (stripping a trailing newline) and returns that string



integer1 = input('Enter the first number: ')



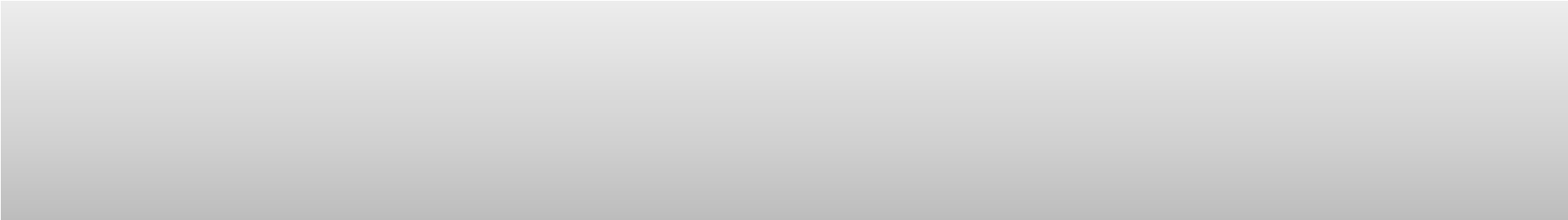
integer1 = int(integer1)

integer2 = input('Enter the second number: ')

integer2 = int(integer2)

sum = integer1 + integer2

print ('Sum is: %d'% sum)



>>>

Enter the first number: 12

Enter the second number: 23

Sum is: 35

## Common Errors

* Variable names starting with number, has symbols such as /, etc
* Variable name having a space or using a



keyword as a variable name

•

Using an uninitialized variable

•

Unbalanced parenthesis

•

Round

-

off errors during floating point

calculations

## Sample Application

* Application: Construction/Survey
* Given the angle of elevation Θ in degrees and the distance from the tree X, write a python program to calculate the height of the tree or a building



Follow problem solving steps starting

from information gathering



Time: 20 mins+ 15 mins

## Solution



# Program to calculate the height of the tree

# The angle and the distance from the tree is provided

# by the user

import math

# Take the inputs from the user

angle = float(input('Enter the angle in degrees: '))

angle = math.radians(angle) # Convert into radians



distance = float(input('Enter the distance in meters : '))

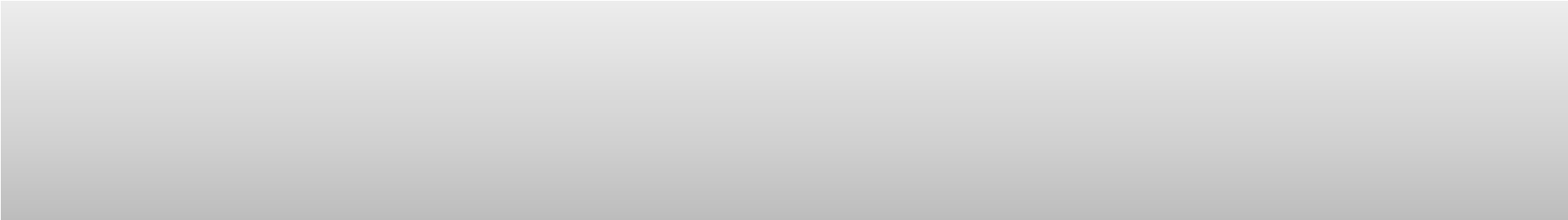
# Calculate the height using the formula y = x tan (r)

height = distance \* math.tan(angle)

# Print the output

print('Height of the tree will be approximately %.2f meters' %

height)



>>>

Enter the angle in degrees: 31.8

Enter the distance in meters : 71

Height of the tree will be approximately 44.02 meters

## Challenge #1

* Application: Linear Algebra
* Write a python program to solve the quadratic equation given its coefficients



Follow problem solving steps starting

from information gathering



Time: 20 mins+ 15 mins

## Challenge #2

• Given the height a person can jump on Earth, say x meters, write a python script that displays how much the same person can jump on various other planets in our solar system and the moon.

– Get inspired by the movie John Carter (2012)



Follow problem solving steps starting

from information gathering



Time: 20 mins+ 15 mins

## Challenge #3

• Given a triangles dimensions x, y and z in meters:

– Validate the dimensions

– Find out the three angles of the triangle