python.org

EE461L

SOFTWARE ENGINEERING DESIGN & LABORATORY

Python Crash Course

Guido Van Rossum
Former Benevolent Dictator for Life (BDFL)

- Python 1: 1990
- Python 2: 2000
- Python 3: 2008

Resources

- https://david.goodger.org/projects/pycon/2007/idiomatic/handout.html
- Style guide: https://www.python.org/dev/peps/pep-0008/
- https://docs.python.org/3.7/

```
>>> help()
help> range
...
help> quit
```

First Examples

Lists are mutable

```
# One line comment, like // in Java
# By @gvanrossum
greetings = ["world hello", "nieuwjaar gelukkig", "happy year new"]
for g in greetings:
    words = sorted(g.split())
    print(" ".join(words).title())
```

Output:

Hello World
Gelukkig Nieuwjaar
Happy New Year

```
from math import sqrt
def is_prime(n):
    assert n > 0
    if (n == 2):
        return True
    if(n == 1) or ((n % 2) == 0):
        return False
    for i in range(3, int(sqrt(n)) + 1, 2):
        if (n % i) == 0:
            return False
    return True
print(is_prime(3))
print(is_prime(10))
print(is_prime(119))
```

Output:

True
False
False

First Examples

```
from math import sqrt
def is_prime(n):
    assert n > 0
    if (n == 2):
        return True
    if(n == 1) or ((n % 2) == 0):
        return False
    for i in range(3, int(sqrt(n)) + 1, 2):
        if (n % i) == 0:
            return False
    return True
```

- `from math import sqrt  # sqrt(n)`
- `import math  # math.sqrt(n)`
- **Blocks (or suites) begin with colon**
  - **consistent indentation required – don't mix tabs and spaces!!**
  - convention: 4 spaces per level, no tabs
- `range([start,] stop[, step])`
  - sequence of numbers from start to stop-1

Interactive Mode

```
>>>print('We are the {} who say "{}!"'.format('knight', 'Ni'))
We are the knights who say "Ni!"
```

- Interactive Interpreter
  - AKA REPL – Read Eval Print Loop
  - Runs loop that reads what you type, evaluates, and prints the result
  - Type python3 at command prompt
  - Fun way to try stuff out...

```
>>> # python is opinionated
>>> import this
>>> import antigravity
>>> from __future__ import braces
```

REPL: Try It

```
>>> x = 3
>>> dir(x)
>>> help(x)
>>> x.numerator
>>> (1+3j).conjugate()
>>> range(4)
>>> sum(range(4))
>>> list(range(4))
>>> 5
>>> _ + 3  # last displayed value is _
```

About Python

- Python is: interpreted, dynamically and strongly typed, garbage-collected, general-purpose, object-oriented
- Dynamically typed: type checking done only as code runs
- Strongly typed: No operations incompatible with types

```
>>> if False:
...     1 + "one"  # Line never runs, no TypeError raised
... else:
...     1 + 1
2
>>> 1 + "two"  # This runs - TypeError
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Types

Everything is an object

```
>>> x = 3
>>> type(x)
<class 'int'>
>>> x = 'zebra'
>>> type(x)
<class 'str'>
>>> x = 21.5
>>> type(x)
<class 'float'>
>>> x = 2 + 4j
>>> type(x)
<class 'complex'>
>>> type(3==4)
<class 'bool'>
```

```
>>> type([1, 2, 3])
<class 'list'>
>>> type((1, 2, 3))
<class 'tuple'>
>>> type({'a': 1, 'b':2, 'c':3})
<class 'dict'>
```

Comments

```
# is the one line comment, like // in Java

"""
Multi-line comments
Lie between quotation marks
This is a haiku
"""
```

Triple single or double quotes

Haiku credit: Michael Cooper (Stanford)

Operators

- Arithmetic: +, -, *, /, //, **, %
  - `7 // 3   # 2`
- Comparison: ==, !=, >, >=, <, <=
  - `3 > 2 > 1  # True (3 > 2 and 2 > 1)`
- Logical: and, or, not
  - `not True        # False`
  - `True and False  # False`
  - `True or False   # True (short circuits)`
- Assignment: =, +=, -=, *=, /=, %=, etc.

- Python operators at w3schools

Operators

- Identity (is the same object): is, is not
  - `list1 = [1, 2, 3]`
  - `list2 = [1, 2, 3]`
  - `list1 == list2  # True`
  - `list1 is list2  # False`
  - `list1 is not list2 # True`

```
>>> id(list1)
4443833672
>>> id(list2)
4443833416
```

- Membership: in, not in
  - `0 in [2, 3, 4]  # False`
  - `4 not in [3, 4, "EE461L"]  # False`
  - `"tho" in "Python"  # True`

Strings

```
# TypeError
word = "rain"
word[1] = 'e'
```

- Python strings are immutable
- No char type – just a string of length 1
- String literals enclosed in ' ' or " "
  - `'don\'t eat spam and eggs'  # "don't eat spam and eggs"`
- Concatenate two strings with +
  - `"hello " + "world" # 'hello world'`
  - `'number' + str(1)  # 'number1'`
  - `'hey' + 3 * 'ya'   # 'heyyayaya'`
- 0-based Indexing
  - `my_string = "Hello Python"`
  - `my_string[0]  # 'H'`
  - `my_string[-1] # 'n'`
  - `my_string[-2] # 'o'`
- Length with len()
  - `s = "hello world"`
  - `len(s)  # 11`

String Slicing

- `s[i:j]` # substring of s from index i (included) to j (excluded)
- `s[i:j:k]` # substring of s from i to j with step k
- omitted first index → 0, omitted second index → size of string

```
s = "Python"
s[0:2]              # 'Py'
s[-2:]              # 'on'
s[:2] + s[2:]       # 'Python'
s[1:4:2]            # 'yh'
s[5:1:-2]           # 'nh'
s[3:40]             # 'hon'
```

Exercise: Reverse the string using slicing

Strings

Method	Description	Example
s.startswith(s2)	Is s2 a prefix of s?	'utece'.startswith('ut') # True
s.endswith(s2)	Is s2 a suffix of s?	'utece'.endswith('ce') # True
s.count(s2)	# of occurrences of s2 in s	'hissing snakes'.count('s') # 4
s.upper(), s.lower()	returns new string with every letter is upper (lower) case	'hello'.upper() # returns 'HELLO'
s.replace(y, z)	return string with all y in s replaced with z	'aggie'.replace('g', '') # 'aie'
s.find(s2), s.rfind(s2)	return index of first (last) occurrence of s2 in s	'green grass'.find('e') # 2 -1 if s2 not found
s.isalpha(), s.isalnum(), s.isdigit(), s.isspace(), s.islower(), s.isupper()	Tests whether characters in s are letters / alphanumeric / digits /spaces ...	''.isalpha() # False '123'.isdigit() # True
s.upper(), s.lower(), s.title()	Returns string which is same as s except all letters are uppercase / lowercase /	'hello world'.title() # 'Hello World'

Strings

- Recall: strings are immutable

```
# Create string "0123...1819"
nums = ""
for n in range(20):
    nums += str(n) # not efficient
print(nums)

# Better
nums = []
for n in range(20):
    nums.append(str(n))
print "".join(nums) # more efficient

# Or use a list comprehension - more on this later
nums = [str(n) for n in range(20)]
print "".join(nums)
```

String Formatting

Curly braces act as placeholders

```
'Hello {} {}'.format('happy', 'coders')
# => 'Hello happy coders'

# Provide values by position or placeholder
'{1} will do {0}'.format('pig', 'that')
'{name} loves {food}'.format(name = 'Elvis',food = 'spam')
# => 'Elvis loves spam'

'Double {} to get {}'.format(5, 5*2)
# => 'Double 5 to get 10'

"{:06.2f}".format(3.14159) # => '003.14' (C-style)
```

Input

- User input via `input()` function
  - `name = input("Please enter your name: ")`
    - `print(f"Hello, {name}!")`
  - `age = int(input("Please enter your age: "))`
- Cast a string to an int with `int()`

```
>>> age = int(input('Please enter your age: '))
Please enter your age: thirty
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'thirty'>>>

# Handle the exception
while True:
    try:
         age = int(input('Please enter your age: '))
         break
    except ValueError:
         print('That was not a valid number. Try again...')
```

Lists

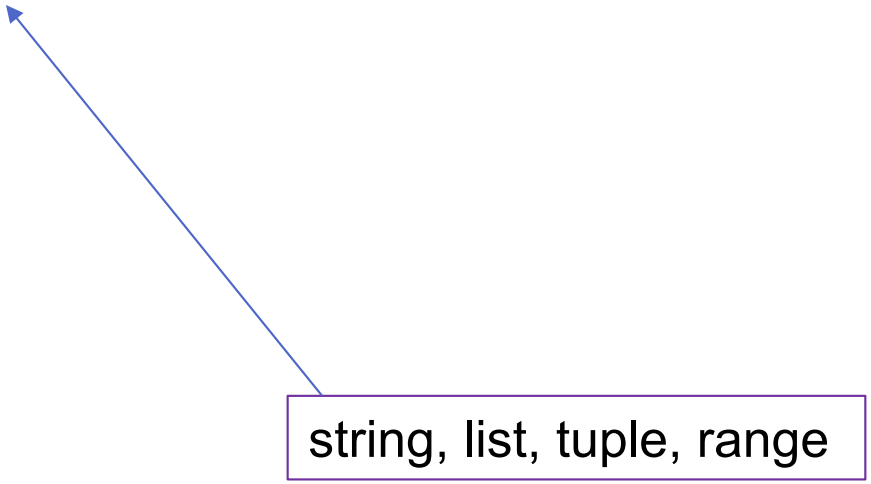
- Lists are mutable

```
# Create new lists
empty = [ ]
letters = ['a', 'b', 'c']
stuff = ['a', "hello", True, [1, 2], 3]
stuff[3]    # what's this?
stuff[3][1] # what's this?

# extend(): append elements of a list
>>> letters.extend(['d', 'e'])
>>> letters
['a', 'b', 'c', 'd', 'e']

# append(): add item to end of list
list_1 = [1, 2]
list_1.append(3) # [1, 2, 3]
```

List Indexing and Slicing

- Like strings and all built-in sequence types, lists can be indexed and sliced

```
numbers = [1, 2, 3, 4]
>>> numbers[1]
2
>>> numbers[1:-1]
[2, 3]
>>> numbers[::2]
[1, 3]
>>> numbers[1:3:-1]
[]
```



List Exercises

```
numbers = [0, 1, 2, 'three', 4, 5, 6, 7, 8, 9]
```

Try these:

- `numbers[0:4]`
- `numbers[:4]`
- `numbers[4:]`
- `numbers[2:-2]`
- `numbers[0:9:2]`
- `numbers[::2]`
- `numbers[::-1]`

Copy of a List

- `copy = numbers[:]` # shallow copy
  - `copy == numbers` # True
    - logical equivalence, like Java's .equals()
  - `copy is numbers` # False
    - same object test: like Java's ==
  - New list containing references to objects in original list

```
>>> list1 = [[1, 2], 3]
>>> copy1 = list1[:]
>>> copy1[0].append(4)
>>> list1
[[1, 2, 4], 3]
>>> copy1
[[1, 2, 4], 3]
```

Also shallow copy:

```
import copy
copy2 = copy.copy(list1)
```

- **`a_list = numbers`** **# not a copy - two names for one object**
  - **`a_list == numbers`** **# True**
  - **`a_list is numbers`** **# True**

Deep Copy

```
# import the copy module
>>> import copy
>>> list1 = [1, [2, 3], 4]
>>> list2 = copy.deepcopy(list1)
>>> list1[1].append(8)
>>> list1
[1, [2, 3, 8], 4]
>>> list2
[1, [2, 3], 4]
```

List Operations

For list s:

```
L[i] = x          # item with index i replaced with x
L[i:j] = t        # slice replaced with items of list/tuple/string t
del L[i:j]        # remove items in slice
del L[i:j:k]      # remove items in slice
del L[i]          # remove item at index i
L.clear()         # remove all items from L
L.insert(i, x)    # insert x at index i
L.pop(i)          # returns and removes item at index i (or last item)
L.remove(x)       # remove first occurrence of x from L
L.reverse()       # reverses L's elements
L.count(x)        # returns number of occurrences of x
L.sort()          # sorts L's items in place
L.index(x)        # return index of first occurrence of x
len(L)            # number of items in L
```

List Examples

```
fruits = ['apple', 'pear', 'peach', 'apple']
fruits.remove('apple')      # ['pear', 'peach', 'apple']

if 'peach' in fruits:
    print("found peach")    # output: found peach

fruits.append('kiwi')       # ['pear', 'peach', 'apple', 'kiwi']
del fruits[3]               # ['pear', 'peach', 'apple']

fruits.pop()                # => 'apple'
fruits.sort()               # ['peach', 'pear']

squares = []
for i in range(10):
    squares.append(i*i)
print("squares are:", squares)
```

Ouput: squares are: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Queues

- Can use a list as a queue – first item added is first removed
  - not efficient – inserts/pops from beginning of list are slow
  - all other elements being shifted
- Instead: use collections.deque which provides fast queue operations

```
>>> from collections import deque
>>> beatles = deque(['John', 'Paul', 'George','Ringo'])
>>> beatles
deque(['John', 'Paul', 'George', 'Ringo'])
>>> beatles.append('Pete')
>>> beatles
deque(['John', 'Paul', 'George', 'Ringo', 'Pete'])
>>> beatles.popleft()
'John'
>>> beatles
deque(['Paul', 'George', 'Ringo', 'Pete'])
```

List Comprehensions

- Another way to create a list
- Basic syntax:
  - `[fn(x) for x in <iterable>]`
  - `[fn(x) for x in <iterable> if cond(x)]`

```
>>> squares = [i*i for i in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehensions

- Consider these sets:

$$S = \{x^2 : x \text{ in } \{0 \ldots 9\}\}$$
$$V = (1, 2, 4, 8, \ldots, 2^{12})$$
$$M = \{x \mid x \text{ in } S \text{ and } x \text{ even}\}$$

- You can define them in Python with a list comprehension:

```
>>> S = [x**2 for x in range(10)]
>>> S
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> V = ??? # Exercise
>>> V
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
>>> M = [x for x in S if x % 2 == 0]
>>> M
[0, 4, 16, 36, 64]
```

List Comprehensions

```
>>> sentence = 'The quick brown fox jumped over the lazy dog'
>>> words = sentence.split()
>>> print(words)
['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy',
'dog']

>>> wordInfo = [[w.upper(), w.lower(), len(w)] for w in words]
>>> for i in wordInfo:
...     print(i)
...
['THE', 'the', 3]
['QUICK', 'quick', 5]
['BROWN', 'brown', 5]
['FOX', 'fox', 3]
['JUMPED', 'jumped', 6]
['OVER', 'over', 4]
['THE', 'the', 3]
['LAZY', 'lazy', 4]
['DOG', 'dog', 3]
```

Exercise

- Define a list comprehension that uses `list1` to produce the list `[1, 3, 5, 7]`

```
list1 = [0, 1, 2, 3]
```

- Define a list comprehension that uses `list2` to produce the list `['A', 'O', 'P']`

```
list2 = ['apple', 'orange', 'pear']
```

Lists & Strings

```
list('Hello') # => ['H', 'e', 'l', 'l', 'o']

# split string into list
'spam and eggs'.split() # => ['spam', 'and', 'eggs']

# split string into list with different delimiter
'1-1-2020'.split(sep='-') # => ['1', '1', '2020']

# join list elements to create a string
'HI'.join(['George', 'John', 'Ringo'])
# => 'GeorgeHIJohnHIRingo'
```

Tuples

- Ordered, **immutable**, non-homogeneous collection
- Similar to lists except:
  - Tuple elements enclosed in parentheses not square brackets
  - Tuples are immutable
- Index and slice just like strings and lists

```
t = (1, 2, 'three', 4)
t[1] # => 2
t[::-1] # => (4, 'three', 2, 1)
t[2] = 3 # TypeError: 'tuple' object does not support item
assignment
t = ( ) # empty tuple
t = (1,) # must include comma or t is just an int
```

Tuple Unpacking

```
>>> t = (1, 2, 3, 4)
>>> (s1, s2, s3, s4) = t
>>> s1
1
>>> s2
2
>>> t = 2,
>>> t
(2,)
```

```
>>> t = 1, 2, 3
>>> t
(1, 2, 3)
>>> x1, x2, x3 = t
>>> x1, x2, x3
(1, 2, 3)
>>> x1, x2, x3 = 4, 5, 6
>>> x1, x2, x3
(4, 5, 6)
```

Swapping with tuple assignment:

```
>>> a = "spam"
>>> b = "eggs"
>>> a, b = b, a
>>> a, b
('eggs', 'spam')
```

Tuple Unpacking

```
def some_math(a, b, c):
    return a + b - c
nums = (2, 3, 1)
some_math(*nums) # => 4
```

* unpacks the tuple
same as `some_math(2, 3, 1)`

Tuple Unpacking

```
list(enumerate(['hello', 'world', 'hi'])
# => [(0, 'hello'), (1, 'world'), (2, 'hi')]

for i, word in enumerate(['hello', 'world', 'hi']):
    print(i, word)
```

Output:

```
0 hello
1 world
2 hi
```

Dictionaries

- Python's associative array – collection of key-value pairs
- d = { <key>: <value>, <key>: <value>, ... }
- Keys must be immutable type (strings, numbers, tuples)

- Create:
  - `empty = { }`
  - `x = {"one":1, "two":2, "three":3}`
  - `y = dict(one=1, two=2, three=3)`
  - `z = dict([('one', 1), ('two', 2), ('three', 3)])`
  - `x == y == z # => True`

Dictionaries

```
d = {"one":1, "two":2, "three":3}

# Access
d["one"] # => 1
d["five"] # => KeyError

# Mutate
d["two"] = 22 # Change value for existing key
d["four"] = 4 # Add new key-value pair

# length
len(d) # => 4
```

Dictionaries

```
d = {"one":1, "two":2, "three":3}

# Removing
del d['one']
del d['five']          # KeyError
d.pop('three', None) # => 3
d.clear()                # removes all key-value pairs

# Items, Keys and Values
list(d)            # => ['one', 'two', 'three']
list(d.items())  # => [('one', 1), ('two', 2), ('three', 3)]
list(d.keys())   # => ['one', 'two', 'three']
list(d.values()) # => [1, 2, 3]

for key, value in d.items():
    print(key, value)
```

Return value if key not in dictionary

Output:
one 1
two 2
three 3

Dictionaries

```
person={'fname': 'Elvis', 'lname': 'Presley', 'age': 85}

>>> person['fname'][:4]                    # => 'Elvi'
>>> 'fname' in person                      # => True

>>> person.get('fname')                    # => 'Elvis'
>>> person.get('children')                 # => None
```

Control Flow

while loops:

```
a, b = 0, 1
while a < 1000:
    print(a, end=',')
    a, b = b, a+b
```

Output:

0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,

if statements:

```
x = int(input("Please enter an integer: "))
if x < 0:
    print("Negative")
elif x == 0:
    print("Zero")
elif x == 1:
    print("One")
else:
    print("More")
```

Control Flow

for loops:

```
words = ['hello', 'wonderful', 'world']
for w in words:
    print(w, len(w))
```

Output:

hello 5

wonderful 9

world 5

```
for ch in "EE461L":
    print(ch, end = ' ')
```

Output:

E E 4 6 1 L

range() function

- Generates sequence of integers lazily
- range(10) generates 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- range(5, 10) generates 5, 6, 7, 8, 9
- range(-10, -100, -30) generates -10, -40, -70

same rules as slicing!

```
>>> type(range(10))
<class 'range'>
>>> range(10)
range(0, 10)
```

```
>>> sum(range(4))
6
>>> list(range(4))
[0, 1, 2, 3]
```

Example:

```
words = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
    print(i, a[i])
```

Output:

0 Mary
1 had
2 a
3 little
4 lamb

`break` and `continue`

- `break` statement: breaks out of innermost enclosing for or while loop
- `else` clause – executed when `for` loop is exhausted or `while` condition becomes false
  - doesn't execute when loop terminated with break

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop didn't find a factor
...         print(n, 'is a prime number')
...
```

Output:

```
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

`break` and `continue`

- `continue` statement: continues with next iteration of loop

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print("Found an even number", num)
...         continue
...     print("Found a number", num)
```

Output:

```
Found an even number 2
Found a number 3
Found an even number 4
Found a number 5
Found an even number 6
Found a number 7
Found an even number 8
Found a number 9
```

Functions

- The def keyword used to define new function.

```
def my_fun(param1, param2):
    ...
    return something # if omitted, function returns None
```

- Multiple return values

```
def return_fun():
    # Packs these into a tuple that gets returned
    return 1, 2, 'three'

values = return_fun()
print(values) # (1, 2, 'three')

a, b, c = return_fun()
print(b)      # 2
```

Arguments: Mutable vs. Immutable

```
def foo(x):
    x += 1

x = 5
foo(x)
x # 5
```

A new object is created and bound to x, but the scope of `x` is `foo`.

```
def foo(x):
    x.append(41)

x = [5]
foo(x)
x # [5, 41]
```

Lists are mutable – `x` isn't being rebound to a new object.

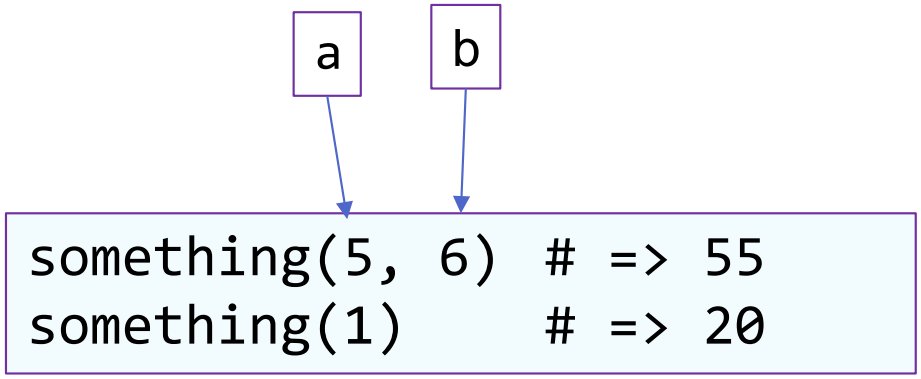
Parameters

- So far: required positional parameters

```
def something(a, b, c):
    return (a+b)*c
```

- Default parameters:

```
def something(a, b = 3, c = 5):
    return (a+b)*c
```



- Keyword arguments:

```
something(1, c = 9, b = 2)# => 27
something(c = 3, a = "hi", b = 'yo') # => 'hiyohiyohiyo'
```

Variadic Positional Arguments

```
def fun(arg1, *args):
    print("arg1: ", arg1) #arg1 is a mandatory parameter
    print("args: ", args) #args is a tuple of positional parms

>>> fun(10, 20, 30)
arg1: 10
args: (20, 30)

>>> fun(1)
arg1: 10
args: ()
```

Arguments

```
def fun(arg1, *args, **kwargs):
  print("arg1: ", arg1) #arg1 is a mandatory parameter
  print("args: ", args) #args is a tuple of positional parms
  print("kwargs: ", kwargs) #kwargs is dictionary of keyword parms

>>> fun(arg1=10, x=1, y=2, z=3)
arg1: 10
args: ()
kwargs: {'x':1, 'y':2, 'z':3}

>>> fun(1)
arg1: 10
args: ()
```

Exceptions

File exc.py:

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero")
    else:
        print("result =", result)
    finally:
        print("executing finally clause")
```

```
>>> from exc import divide
>>> divide(2, 1)
result = 2.0
executing finally clause
>>> divide(2, 0)
division by zero
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
 File "/Users/ece-mve55/Documents/pythonStuff/exc.py", line 3, in divide
    result = x / y
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Exceptions

```
try:
    some_dangerous_code()
except SomeError as e:
    handle_exception(e)
except AnotherError:
    handle_without_binding()
except (OneError, TwoError):
    handle_multiple_errors()
except:
    handle_everything_else()
```

Exceptions

```
>>> def this_fails():
...     x = 1/0
...
>>> try:
...     this_fails()
... except ZeroDivisionError as err:
...     print('Handling run-time error:', err)
...
Handling run-time error: division by zero
```

Objects

- Everything is an object

```
isinstance(2, object)  # => True
isinstance("EE461L", object) # => True
isinstance([6, 5], object) # => True
```

Objects have identity, type and value

- In CPython, identity is memory address of object
  - `id(6) # => 4523233584`
- Objects have type:
  - `type(math) # => <class 'module'>`
  - `type(6) # => <class 'int'>`

Classes

```
class Person:
    """Instantiates a Person object with given name."""

    def __init__(self, first_name, last_name): # like Java constructor
        self.firstname = first_name
        self.lastname = last_name

    def __str__(self):
        return self.firstname + " " + self.lastname

    def getFirstname(self):
        return self.firstname

    def getLastname(self):
        return self.lastname

    def setFirstname(self, newFirst):
        self.firstname = newFirst

    def setLastname(self, newLast):
        self.lastname = newLast

person1 = Person("Elvis", "Presley")
print(person1) # calls the __str__ method on person1
```

Inheritance

```
class SuperHero(Person):
    def __init__(self, firstname, lastname, nick):
        super(SuperHero, self).__init__(firstname, lastname)
        self.nick = nick

    def nick_name(self):
        return "I am {}".format(self.nick)


p = SuperHero("Clark", "Kent","Superman")
print(p.nick_name())                # I am Superman
print(p)                            # Clark Kent
```

SOURCES

stanfordpython.com
docs.python.org/3/tutorial/
realpython.com
Head First Python