Python Reference Guide

Need a quick reference guide to help you while you hack? This is no substitute for the actual course content, but here's a "cheatsheet" to help jog your memory!

Handy Functions

In [3]:
```
print('This will print some text')
```

```
This will print some text
```

In [90]:
```
# Use string formatting ("f-strings") to insert values
name = 'Ryan'
f'My name is: {name}'
```

Out[90]:
```
'My name is: Ryan'
```

In [31]:
```
# An iterable object containing sequential integers
range(0, 10)

# Iterate by steps of 2
range(0, 10, 2)
```

Out[31]:
```
range(0, 10, 2)
```

In [28]:
```
# Returns the datatype of the value passed in
type(1)
```

Out[28]:
```
int
```

Math Operators

In [2]:
```
# Addition
1 + 1
```

Out[2]:
```
2
```

In [3]:
```
# Subtraction
2 - 1
```

Out[3]:
```
1
```

In [4]:
```
# Multiplication
5 * 5
```

Out[4]:
```
25
```

In [5]:
```
# Division
25 / 5
```

Out[5]: 5.0

In [6]:
```
# Exponents
5 ** 3
```

Out[6]: 125

In [7]:
```
# Modulus (remainder after division)
17 % 5
```

Out[7]: 2

Data Types

In [1]:
```
# Integers
x = 1
type(x)
```

Out[1]: int

In [2]:
```
# Floats
x = 1.0
type(x)
```

Out[2]: float

In [3]:
```
# Strings
x = 'Ryan'
type(x)
```

Out[3]: str

In [4]:
```
# Booleans
x = True
type(x)
```

Out[4]: bool

In [5]:
```
# Byte data
x = bytes(4)
type(x)
```

Out[5]: bytes

Casting Datatypes

In [85]:
```
# String to integer
int('1')
```

Out[85]: 1

In [86]:
```
# Float to integer
int(1.5)
```

Out[86]: 1

In [91]:
```
# Integer to float
float(1)
```

Out[91]: 1.0

In [89]:
```
# Integer to boolean
bool(1)
```

Out[89]: True

In [88]:
```
# String to boolean (Anything other than an empty string is True)
bool('')
```

Out[88]: False

In [92]:
```
# Int to string
str(1)
```

Out[92]: '1'

In [94]:
```
# String to bytes, requires the encoding
bytes('🙂', 'utf-8')
```

Out[94]: b'\xf0\x9f\x99\x82'

String Operators

In [81]:
```
# String concatenation
'Ryan ' + 'Mitchell'
```

Out[81]: 'Ryan Mitchell'

In [83]:
```
# String multiplication
'Python'*6
```

Out[83]: 'PythonPythonPythonPythonPythonPython'

In [84]:
```
# Access a particular character in a string
'Python'[3]
```

Out[84]:
```
'h'
```

Boolean Operators

In [62]:
```
# AND
print(True and True)
print(True and False)
print(False and False)
```

```
True
False
False
```

In [64]:
```
# OR
print(True or True)
print(True or False)
print(False or False)
```

```
True
True
False
```

In [65]:
```
# NOT
print(not True)
print(not False)
```

```
False
True
```

In [75]:
```
# Equality operator
5 == 5
```

Out[75]:
```
True
```

In [74]:
```
# Inequality operator
5 != 4
```

Out[74]:
```
True
```

In [67]:
```
# Less than
4 < 5
```

Out[67]:
```
True
```

In [71]:
```
# Less than or equals to
5 <= 5
```

Out[71]:
```
True
```

In [68]:
```
# Greater than
5 > 4
```

Out[68]: True

In [72]:
```
# Greater than or equals to
5 >= 5
```

Out[72]: True

Control Flow

In [ ]:
```
```

In [24]:
```
if False:
    print('This does not print')
else:
    print('This will print')
```

```
This will print
```

In [26]:
```
# Iterate through items in a range
for i in range(0, 5):
    print('Number: {}'.format(i))
```

```
Number: 0
Number: 1
Number: 2
Number: 3
Number: 4
```

In [77]:
```
# Iterate through items in a list
for i in [1,2,3,4,5]:
    print('Number: {}'.format(i))
```

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

In [27]:
```
i = 0
while i < 5:
    print('Number: {}'.format(i))
    i = i + 1
```

```
Number: 0
Number: 1
Number: 2
Number: 3
Number: 4
```

Data Types

Lists

In [36]:
```
aList = [1,2,3,4]
type(aList)
```

Out[36]:
```
list
```

In [51]:
```
# Append item to a list
aList.append(5)
aList
```

Out[51]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 5, 5]
```

In [37]:
```
# List concatenation
[1,2,3] + [4,5,6]
```

Out[37]:
```
[1, 2, 3, 4, 5, 6]
```

In [38]:
```
# List multiplication
[1,2] * 5
```

Out[38]:
```
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

In [43]:
```
# Accessing items in a list

aList = [1,2,3,4,5,6,7,8,9,10]
# Print the item at index 4
print(aList[4])

# Print the items at index 0, up to (not including) index 4
print(aList[0:4])

# If the first index is missing, it's assumed to be 0
print(aList[:4])

# If the last index is missing, it's assumed to go to the end of the list
print(aList[4:])

# Print every other item in the list
print(aList[::2])

# Print every third item in the list
print(aList[::3])
```

```
5
[1, 2, 3, 4]
[1, 2, 3, 4]
[5, 6, 7, 8, 9, 10]
[1, 3, 5, 7, 9]
[1, 4, 7, 10]
```

Dictionaries

In [52]:
```
aDict = {
    'one': 1,
    'two': 2,
    'three': 3,
}
# Add key/value pairs to dictionary
aDict['four'] = 4

# Access values by keys
print(aDict['one'])
```

```
1
```

In [55]:
```
# Print all keys
print(aDict.keys())

# Print all values
print(aDict.values())

# Print all key/value pairs
print(aDict.items())
```

```
dict_keys(['one', 'two', 'three', 'four'])
dict_values([1, 2, 3, 4])
dict_items([('one', 1), ('two', 2), ('three', 3), ('four', 4)])
```

Sets

In [49]:
```
aSet = {1,2,3,4}

# Add an item to a set
aSet.add(5)

# Remove an item from a set
aSet.remove(2)
print(aSet)
```

```
{1, 3, 4, 5}
```

Tuples

In [57]:
```
# Tuples cannot be modified
aTuple = (1,2,3,4)
aTuple
```

Out[57]:
```
(1, 2, 3, 4)
```

Functions

In [10]:
```
# Function with one argument and a return value
def aFunction(anArg):
    return anArg + 1

aFunction(5)
```

Out[10]: 6

In [11]:
```
# Different ways of calling a function with a keyword argument
def aFunction(anArg, optionalArg=1):
    return anArg + optionalArg

print(aFunction(5))
print(aFunction(5, 4))
print(aFunction(5, optionalArg=4))
```

```
6
9
9
```

Classes

In [16]:
```
# A simple class with one attribute and one method
class ParentClass:
    def __init__(self, val):
        self.val = val

    def printVal(self):
        print(self.val)


classInstance = ParentClass('Value for the class attribute "val"')
classInstance.printVal()
```

```
Value for the class attribute "val"
```

In [17]:
```
# Extend the parent class to create a child class
class ChildClass(parentClass):

    def printVal(self):
        print('Child class! {}'.format(self.val))

childInstance = ChildClass('Value for the class attribute "val"')
childInstance.printVal()
```

```
Child class! Value for the class attribute "val"
```

In [18]:
```
# Overriding the parent class constructor
class AnotherChildClass(parentClass):
    def __init__(self):
        super().__init__('A default value')

anotherChildInstance = AnotherChildClass()
anotherChildInstance.printVal()
```

```
A default value
```

File Handling

In [103]:
```
# Open a file for reading
with open('09_01_file.txt', 'r') as f:
    data = f.readlines()
print(data)
```

```
['Beautiful is better than ugly.\n', 'Explicit is better than implicit.\n',
'Simple is better than complex.\n', 'Complex is better than complicated.\n',
'Flat is better than nested.\n', 'Sparse is better than dense.\n', 'Readabili
ty counts.\n', "Special cases aren't special enough to break the rules.\n",
'Although practicality beats purity.\n', 'Errors should never pass silentl
y.\n', 'Unless explicitly silenced.\n', 'In the face of ambiguity, refuse the
temptation to guess.\n', 'There should be one-- and preferably only one --obv
ious way to do it.\n', "Although that way may not be obvious at first unless
you're Dutch.\n", 'Now is better than never.\n', 'Although never is often bet
ter than *right* now.\n', "If the implementation is hard to explain, it's a b
ad idea.\n", 'If the implementation is easy to explain, it may be a good ide
a.\n', "Namespaces are one honking great idea -- let's do more of those!"]
```

In [ ]:
```
with open('test.txt', 'w') as f:
    f.write('Writing a new line\n')
```

In [ ]:
```
with open('test.txt', 'a') as f:
    f.write('Adding a new line to the last one\n')
```