P524: Survey of Instrumentation and Laboratory Techniques

Week 2: Python Scripting

9/3/2024

Python Scripting Language

[Python Cheat Sheet
https://quickref.me/python.html](https://quickref.me/python.html)

Hello World

```
>>> print("Hello, World!")
Hello, World!
```

Variables

```
age = 18      # age is of type int
name = "John" # name is now of type str
print(name)
```

Slicing String

```
>>> msg = "Hello, World!"
>>> print(msg[2:5])
llo
```

Casting

Integers

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

Floats

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
```

Strings

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

Python Operators

Arithmetic operators

```
102 + 37 #Add two numbers with +
102 - 37 # Subtract a number with -
4 * 6 # Multiply two numbers with *
22 / 7 # Divide a number by another with /
22 // 7 # Integer divide a number with //
3 ** 4 # Raise to the power with **
22 % 7 # Returns 1 # Get the remainder  after division with %
```

Numeric comparison operators

```
3 == 3 # Test for equality with ==
3 != 3 # Test for inequality with !=
3 > 1 # Test greater than with >
3 >= 3 # Test greater than or equal to with >=
3 < 4 # Test less than with <
3 <= 4 # Test less than or equal to with <=
```

Logical operators

```
~(2 == 2) # Logical NOT with ~
(1 != 1) & (1 < 1) # Logical AND with &
(1 >= 1) | (1 < 1) # Logical OR with |
(1 != 1) ^ (1 < 1) # Logical XOR with ^
```

List, Tuple, Set

Data Types	
str	Text
int, float, complex	Numeric
list, tuple, range	Sequence
dict	Mapping
set, frozenset	Set
bool	Boolean
bytes, bytearray, memoryview	Binary

Lists

```
list1 = ["apple", "banana", "cherry"]
list2 = [True, False, False]
list3 = [1, 5, 7, 9, 3]
list4 = list((1, 5, 7, 9, 3))
```

Tuple

```
my_tuple = (1, 2, 3)
my_tuple = tuple((1, 2, 3))
```

Set

```
set1 = {"a", "b", "c"}
set2 = set(("a", "b", "c"))
```

Container Types

- **ordered sequences**, fast index access, repeatable values
  - `list` `[1,5,9]` `["x",11,8.9]` `["mot"]` `[]`
  - `tuple` `(1,5,9)` `11,"y",7.4` `("mot",)` `()`
  - *Non modifiable values (immutables)* — *expression with only comas →* `tuple`
  - `str` `bytes` *(ordered sequences of chars / bytes)* `""` `b""`
- **key containers**, no *a priori* order, fast key access, each key is unique
  - **dictionary** `dict` `{"key":"value"}` `dict(a=3,b=4,k="v")` `{}`
    (key/value associations) `{1:"one",3:"three",2:"two",3.14:"π"}`
  - **collection** `set` `{"key1","key2"}` `{1,9,3,0}` `set()`
    keys=hashable values (base types, immutables…) `frozenset` *immutable set* *empty*

Data type conversions

Conversions — `type(`*expression*`)`

```
int("15") → 15
int("3f",16) → 63          can specify integer number base in 2nd parameter
int(15.56) → 15            truncate decimal part
float("-11.24e8") → -1124000000.0
round(15.56,1)→15.6        rounding to 1 decimal (0 decimal → integer number)
bool(x)  False for null x, empty container x , None or False x ; True for other x
str(x)→"…"   representation string of x for display (cf. formatting on the back)
chr(64)→'@'  ord('@')→64        code ↔ char
repr(x)→"…"  literal representation string of x
bytes([72,9,64]) → b'H\t@'
list("abc") → ['a','b','c']
dict([(3,"three"),(1,"one")]) → {1:'one',3:'three'}
set(["one","two"]) → {'one','two'}
separator str and sequence of str → assembled str
   ':'.join(['toto','12','pswd']) → 'toto:12:pswd'
str splitted on whitespaces → list of str
   "words with  spaces".split() → ['words','with','spaces']
str splitted on separator str → list of str
   "1,4,8,2".split(",") → ['1','4','8','2']
sequence of one type → list of another type (via list comprehension)
   [int(x) for x in ('1','29','-3')] → [1,29,-3]
```

Lists

> Getting started with lists

A list is an ordered and changeable sequence of elements. It can hold integers, characters, floats, strings, and even objects.

Creating lists

```
# Create lists with [], elements separated by commas
x = [1, 3, 2]
```

List functions and methods

```
x.sorted(x) # Return a sorted copy of the list e.g., [1,2,3]
x.sort() # Sorts the list in-place (replaces x)
reversed(x) # Reverse the order of elements in x e.g., [2,3,1]
x.reversed() # Reverse the list in-place
x.count(2) # Count the number of element 2 in the list
```

Selecting list elements

Python lists are zero-indexed (the first element has index 0). For ranges, the first element is included but the last is not.

```
# Define the list
x = ['a', 'b', 'c', 'd', 'e']
x[0] # Select the 0th element in the list
x[-1] # Select the last element in the list
x[1:3] # Select 1st (inclusive) to 3rd (exclusive)
x[2:] # Select the 2nd to the end
x[:3] # Select 0th to 3rd (exclusive)
```

Concatenating lists

```
# Define the x and y lists
x = [1, 3, 6]
y = [10, 15, 21]
x + y # Returns [1, 3, 6, 10, 15, 21]
3 * x # Returns [1, 3, 6, 1, 3, 6, 1, 3, 6]
```

NumPy

> NumPy arrays

NumPy is a python package for scientific computing. It provides multidimensional array objects and efficient operations on them. To import NumPy, you can run this Python code `import numpy as np`

Creating arrays

```
# Convert a python list to a NumPy array
np.array([1, 2, 3]) # Returns array([1, 2, 3])
# Return a sequence from start (inclusive) to end (exclusive)
np.arange(1,5) # Returns array([1, 2, 3, 4])
# Return a stepped sequence from start (inclusive) to end (exclusive)
np.arange(1,5,2) # Returns array([1, 3])
# Repeat values n times
np.repeat([1, 3, 6], 3) # Returns array([1, 1, 1, 3, 3, 3, 6, 6, 6])
# Repeat values n times
np.tile([1, 3, 6], 3) # Returns array([1, 3, 6, 1, 3, 6, 1, 3, 6])
```

> Math functions and methods

All functions take an array as the input.

```
np.log(x) # Calculate logarithm
np.exp(x) # Calculate exponential
np.max(x) # Get maximum value
np.min(x) # Get minimum value
np.sum(x) # Calculate sum
np.mean(x) # Calculate mean
```

```
np.quantile(x, q) # Calculate q-th quantile
np.round(x, n) # Round to n decimal places
np.var(x) # Calculate variance
np.std(x) # Calculate standard deviation
```

Dictionary

> Getting started with dictionaries

A dictionary stores data values in key-value pairs. That is, unlike lists which are indexed by position, dictionaries are indexed by their keys, the names of which must be unique.

Creating dictionaries

```
# Create a dictionary with {}
{'a': 1, 'b': 4, 'c': 9}
```

Dictionary functions and methods

```
x = {'a': 1, 'b': 2, 'c': 3} # Define the x ditionary
x.keys() # Get the keys of a dictionary, returns dict_keys(['a', 'b', 'c'])
x.values() # Get the values of a dictionary, returns dict_values([1, 2, 3])
```

Selecting dictionary elements

```
x['a'] # 1 # Get a value from a dictionary by specifying the key
```

Di

```
>>> empty_dict = {}
>>> a = {"one": 1, "two": 2, "three": 3}
>>> a["one"]
1
>>> a.keys()
dict_keys(['one', 'two', 'three'])
>>> a.values()
dict_values([1, 2, 3])
>>> a.update({"four": 4})
>>> a.keys()
dict_keys(['one', 'two', 'three', 'four'])
>>> a['four']
4
```

Python Functions

Basic

```
def hello_world():
    print('Hello, World!')
```

Return

```
def add(x, y):
    print("x is %s, y is %s" %(x, y))
    return x + y

add(5, 6)    # => 11
```

Positional arguments

```
def varargs(*args):
    return args

varargs(1, 2, 3)  # => (1, 2, 3)
```

Keyword arguments

```
def keyword_args(**kwargs):
    return kwargs

# => {"big": "foot", "loch": "ness"}
keyword_args(big="foot", loch="ness")
```

Returning multiple

```
def swap(x, y):
    return y, x

x = 1
y = 2
x, y = swap(x, y)  # => x = 2, y = 1
```

Default Value

```
def add(x, y=10):
    return x + y

add(5)      # => 15
add(5, 20)  # => 25
```

Anonymous functions

```
# => True
(lambda x: x > 2)(3)

# => 5
(lambda x, y: x ** 2 + y ** 2)(2, 1)
```

Python Loops

Basic

```
primes = [2, 3, 5, 7]
for prime in primes:
    print(prime)
```

Prints: 2 3 5 7

With index

```
animals = ["dog", "cat", "mouse"]
# enumerate() adds counter to an iterable
for i, value in enumerate(animals):
    print(i, value)
```

Prints: 0 dog 1 cat 2 mouse

While

```
x = 0
while x < 4:
    print(x)
    x += 1  # Shorthand for x = x + 1
```

Prints: 0 1 2 3

Break

```
x = 0
for index in range(10):
    x = index * 10
    if index == 5:
        break
    print(x)
```

Prints: 0 10 20 30 40

Continue

```
for index in range(3, 8):
    x = index * 10
    if index == 5:
        continue
    print(x)
```

Prints: 30 40 60 70

Range

```
for i in range(4):
    print(i) # Prints: 0 1 2 3

for i in range(4, 8):
    print(i) # Prints: 4 5 6 7

for i in range(4, 10, 2):
    print(i) # Prints: 4 6 8
```

for/else

```
nums = [60, 70, 30, 110, 90]
for n in nums:
    if n > 100:
        print("%d is bigger than 100" %n)
        break
else:
    print("Not found!")
```

Python File Handling

Read file

Line by line

```
with open("myfile.txt") as file:
    for line in file:
        print(line)
```

With line number

```
file = open('myfile.txt', 'r')
for i, line in enumerate(file, start=1):
    print("Number %s: %s" % (i, line))
```

String

Write a string

```
contents = {"aa": 12, "bb": 21}
with open("myfile1.txt", "w+") as file:
    file.write(str(contents))
```

Read a string

```
with open('myfile1.txt', "r+") as file:
    contents = file.read()
print(contents)
```

Object

Write an object

```
contents = {"aa": 12, "bb": 21}
with open("myfile2.txt", "w+") as file:
    file.write(json.dumps(contents))
```

Read an object

```
with open('myfile2.txt', "r+") as file:
    contents = json.load(file)
print(contents)
```

Delete a File

```
import os
os.remove("myfile.txt")
```

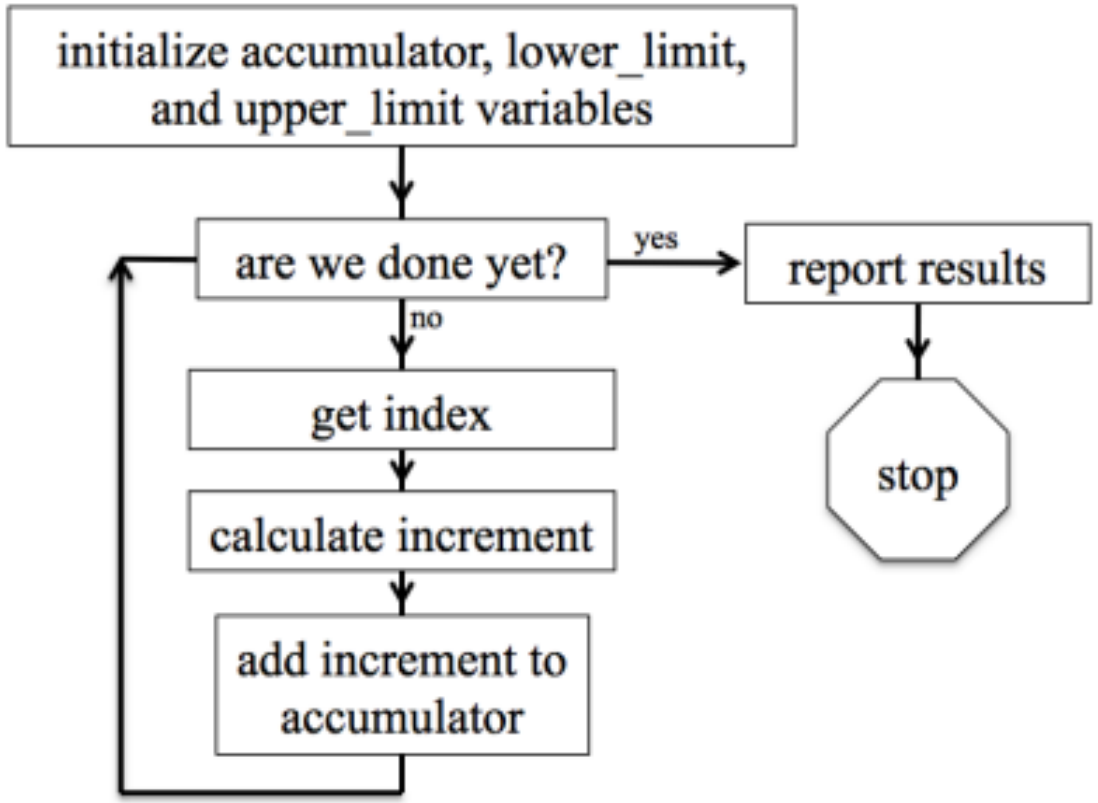
Check and Delete

```
import os
if os.path.exists("myfile.txt"):
    os.remove("myfile.txt")
else:
    print("The file does not exist")
```

Delete Folder

```
import os
os.rmdir("myfolder")
```

A typical python script



A loop to calculate the sum of a few squares

Homework

- Due next Tuesday noon.
- Please email your python scripts to Garrett Williams, **grw5@illinois.edu.**