Demo

June 5, 2023

1 Python Recap

[2]:
```
# We use quotes or double quotes to define strings
x = "Hello 'World'  " # 'Hello World'
x
```

[2]:
```
"Hello 'World'  "
```

[3]:
```
# Strings (in python) are a list of characters

# every character USED to be a 8bit = 1byte integer but that
# created a lot of problems with non-english languages.
# The used encoding was called ASCII.

# Nowadays UNICODE is used, which can handle non-english alphabets
# but we no longer have the guarantee that a single character is
# a 8 bit number.
x[0]
```

[3]:
```
'H'
```

[4]:
```
x[0:4] # x[0],x[1]
```

[4]:
```
'Hell'
```

[5]:
```
x[-5:]
```

[5]:
```
"ld'  "
```

[6]:
```
["H","e","l"]
```

[6]:
```
['H', 'e', 'l']
```

[ ]:
```

```

[7]:
```
# we can define integers (whole numbers)
i = -1      # both negative and positive is possible
u = 0xF3AB # we can also specify it in other formats (e.g. Hexadecimal)
```

```
[ ]:
```

```
[8]: 10 / 3  # = 3.333 -> this creates a result of type float
     10 // 3 # = 3     -> this creates a result of type int
```

```
[8]: 3
```

```
[9]: a, b = True, False # we can use booleans as well
```

```
[10]: a == b # checks if equal
      a != b # checks if NOT equal
```

```
[10]: True
```

```
[11]: # a == b # this checks for equality
      a = b # this is an assignment, we assign the value of b to the variable a
```

1.1 Collections

- tuples
- lists
- dictionaries

```
[12]: # define a tuple - the parantheses are optional in this case
      y = (1.0, 2)
      y
```

```
[12]: (1.0, 2)
```

```
[13]: y_1 = y[0]
      y_1
```

```
[13]: 1.0
```

```
[14]: y[0] = 2.0 # tuples are immutable!
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[14], line 1
----> 1 y[0] = 2.0 # tuples are immutable!

TypeError: 'tuple' object does not support item assignment
```

```
[15]: # Similiar to tuples, we have lists
      # These are also often called 'arrays'
      L = [1,2,3,"hello"]
      L
```

[15]:

```
[1, 2, 3, 'hello']
```

[16]:

```
L[0] = 10
L
```

[16]:

```
[10, 2, 3, 'hello']
```

[17]:

```
# The third important container is a dictionary.
# Its a key value store
D = {
    "some-key": 2,
    "hello": 3
}

# Allowed values: Anything
# Allowed keys: Anything that is "hashable"
#  input -> hash-function -> some value
#  HASH-Maps (or just maps)
D
```

[17]:

```
{'some-key': 2, 'hello': 3}
```

[ ]:

```

```

1.2 Arithmetic

[18]:

```
1 + 2  # = 3
1 - 2  # = -1
2 * 2  # = 4
2 / 2  # = 1.0
2 ** 3 # 2^3 = 8
```

[18]:

```
8
```

[19]:

```
# 10 / 3 = 3 rest 1


10 // 3 # = 3 (returns the quotient of the division)

# To get the rest, we use the Modulo operator '%'
10 % 3  # = 1 (returns the rest of the division)
```

[19]:

```
1
```

1.3 Conditionals

[20]:

```python
c = False

if a == b:
    print("a is equal to b")
elif b == c:
    print("a is not equal to b but b is equal to c")
else:
    # start of ELSE block
    print("a is not equal to b")
    print("this belongs to the ELSE")
    # end of ELSE block

print("but not this")
```

```
a is equal to b
but not this
```

1.4 Loops

[21]:

```python
# Loops: Iterate or execute a statement multiple times

# FOR LOOP: For a known number of iterations
# for _ in _:
# the second argument can be anything that is iteratable
for i in range(0,10):
    print("hello: i = ", i)
print("world")
```

```
hello: i =  0
hello: i =  1
hello: i =  2
hello: i =  3
hello: i =  4
hello: i =  5
hello: i =  6
hello: i =  7
hello: i =  8
hello: i =  9
world
```

[22]:

```python
L = [1,2,3,4,5]
for l in L:
    print(2 * l)
```

```
2
4
6
```

```
8
10
```

[23]:

```
for c in x:
    print(c)
```

```
H
e
l
l
o
 
'
W
o
r
l
d
'
```

[24]:

```
# LOOP 2: WHILE:
# Use it, when unclear how often something should be repeated

x = 0
# while _ :
while x < 10:
    print("waiting")
    x = x + 1
print("done")
```

```
waiting
waiting
waiting
waiting
waiting
waiting
waiting
waiting
waiting
waiting
done
```

[25]:

```
# Loop 3: Variant of for loop



# We can write the code below in a shorter notation
```

```
some_list = [1,2,3]
result = []
for x in some_list:
    result.append(2 * x)
result
```

[25]:
```
[2, 4, 6]
```

[26]:
```
# This is the short form of the loop above
# this is called list comprehension
[2 * x for x in some_list]
```

[26]:
```
[2, 4, 6]
```

[27]:
```
D["hello"]
```

[27]:
```
3
```

1.5 Code readability

[ ]:
```
# Example: Hard to read code
#  1) Unecessary parantheses
#  2) too much in one expression
#  3) unnecessary dictionary
netCharge = (
    +(sum({x: ((seqCount[x]*(10**pKR[x]))/((10**pH)+(10**pKR[x]))) \
    for x in ['k','h','r']}.values()))
    -(sum({x: ((seqCount[x]*(10**pH))/((10**pH)+(10**pKR[x]))) \
    for x in ['y','c','d','e']}.values())))





# Alternative 1
# split it and dont use dictionaries
posCharges = [(seqCount[x] * (10**pKR[x])) / ((10**pH) + (10**pKR[x])) for x in␣
 ↪['k','h','r']]
negCharges = [(seqCount[x] * (10**pKR[x])) / ((10**pH) + (10**pKR[x])) for x in␣
 ↪['y','c','d','e']]
netCharge = sum(posCharges) - sum(negCharges)


# Alternative 2
positiveChargedProteins = ['k','h','r']
netCharge = 0
for x in ['k','h','r','y','c','d','e']:
    # calcule the charge for x
    x_charge = seqCount[x] * 10**pKR[x] / (10**pH + 10**pKR[x])

    if x in positiveChargedProteins:
        netCharge = netCharge + x_charge
    else:
        netCharge = netCharge - x_charge
```

1.6 Functions

[197]:

```
# Functions are a way to reuse a block of code
print("hello: ", 21) # we dont have to implement 'print' ourselfes!
```

```
hello:  21
```

[29]:

```
#      function name
#       |    input(s)
#       |    |
#       v    v
def is_prime(n):
    # Check if a number is prime (only divisible by 1 and itself)
    # we do that by testing every number between 1 and itself

    # Example: 6
    # Test: 6 % 2 == 0 -> not prime

    # Example 7:
    # Test:
    #  7 % 2 != 0
    #  7 % 3 != 0
    #  7 % 4 != 0
    #  7 % 5 != 0
    #  7 % 6 != 0
    #
    # 7 is prime!
    for i in range(2,n):
        if n % i == 0: # % = modulo operator, gives the rest of a division
            return False # <-- return indicates that the function should stop␣
 ↪here

    return True # <--- return indicates that the function should stop and which␣
 ↪value will be delivered to outer code
```

[32]:

```
# When python sees this line, it first will evelaute the left side. For this␣
 ↪the function is "called" (=executed) with the input 13
# Then the left side is replaced with the "output" (the return value) of the␣
 ↪function
```

```
result = is_prime(13)
print(f"The result is: {result}")
```

```
The result is: True
```

[31]:

```
is_prime(11)
```

[31]:

```
True
```

[33]:

```
is_prime(12)
```

[33]:

```
False
```

[34]:

```
def fetch_large_dataset(filename):
    import time
    # Fetch a large dataset and write to a file

    # TODO: implement fetch
    time.sleep(1)
```

[ ]:

```

```

[ ]:

```

```

[52]:

```
def my_addition(a, b):
    return a + b
```

[54]:

```
my_addition(1, 2) # <-- this "calls" the function
```

[54]:

```
3
```

[56]:

```
my_addition(b=3,a=2) # we can also specify the parameters "a" and "b"␣
 ↪explicitely. Then the order does not matter.
```

[56]:

```
5
```

[57]:

```
y = my_addition(1,2) # y = 3
```

[60]:

```
# Multiple inputs and outputs are also possible
import random
def pick_random(some_list, n_experiments=1):
    # This function will pick a random element from the provided list␣
 ↪"some_list" multiple times.
    # n_experiments is the number, how often a random element will be picked.

    n = len(some_list)
    indices = [random.randint(0,n-1) for _ in  range(n_experiments)]
    elements = [some_list[ix] for ix in indices]
    return indices, elements
```

[61]:
```
pick_random([1,-1], n_experiments=10)
```

[61]:
```
([1, 1, 0, 1, 1, 0, 1, 1, 0, 0], [-1, -1, 1, -1, -1, 1, -1, -1, 1, 1])
```

[62]:
```
pick_random([1,-1], n_experiments=2)
```

[62]:
```
([1, 0], [-1, 1])
```

[63]:
```
pick_random(n_experiments=2, some_list=[-1,1])
```

[63]:
```
([0, 1], [-1, 1])
```

[ ]:
```

```

[64]:
```
import math
```

[65]:
```
math.ceil(0.9) # rounds up to the nearest integer
```

[65]:
```
1
```

[70]:
```
math.floor(1.2) # rounds down to the nearest integer
```

[70]:
```
1
```

[71]:
```
"There are {} apples".format(10) # <-- a method (is a function bound to an␣
 ↪object)
```

[71]:
```
'There are 10 apples'
```

[ ]:
```

```

[72]:
```
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
alphabet2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

[73]:
```
key = 28 % 26 # / 26 -> rest
key
```

[73]:
```
2
```

[80]:
```
10 // 3   # integer division -> 10 / 3 = 3 Rest 1.
```

[80]:
```
3
```

[81]:
```
10 % 3   # modulo operation = the rest of an integer division
```

[81]:
```
1
```

[ ]:

```

```

2 Example: Fetch data and display it

[85]:

```
# Import functions and code from other python modules
# Libraries/packages can be installed with pip
#

import requests                   # "requests" is a popular python library to␣
 ↪work with HTTP requests
import datetime
from datetime import datetime
import matplotlib.pyplot as plt   # "matplotlib" is a popular python library␣
 ↪draw plots and diagrams
```

[86]:

```
def fetch_market_data():
    l = "https://api.awattar.at/v1/marketdata"
    response = requests.get(l)
    if response.status_code != 200:
        response.raise_for_status()
    x = response.json()
    return x['data']
```

[87]:

```
def prep_data(X):
    t1 = [datetime.utcfromtimestamp(x['start_timestamp']/ 1000)  for x in X]
    t2 = [datetime.utcfromtimestamp(x['end_timestamp'] / 1000) for x in X]
    p = [x['marketprice'] for x in X]
    u = [x['unit'] for x in X]
    return (t1,t2,p,u)
```

[88]:

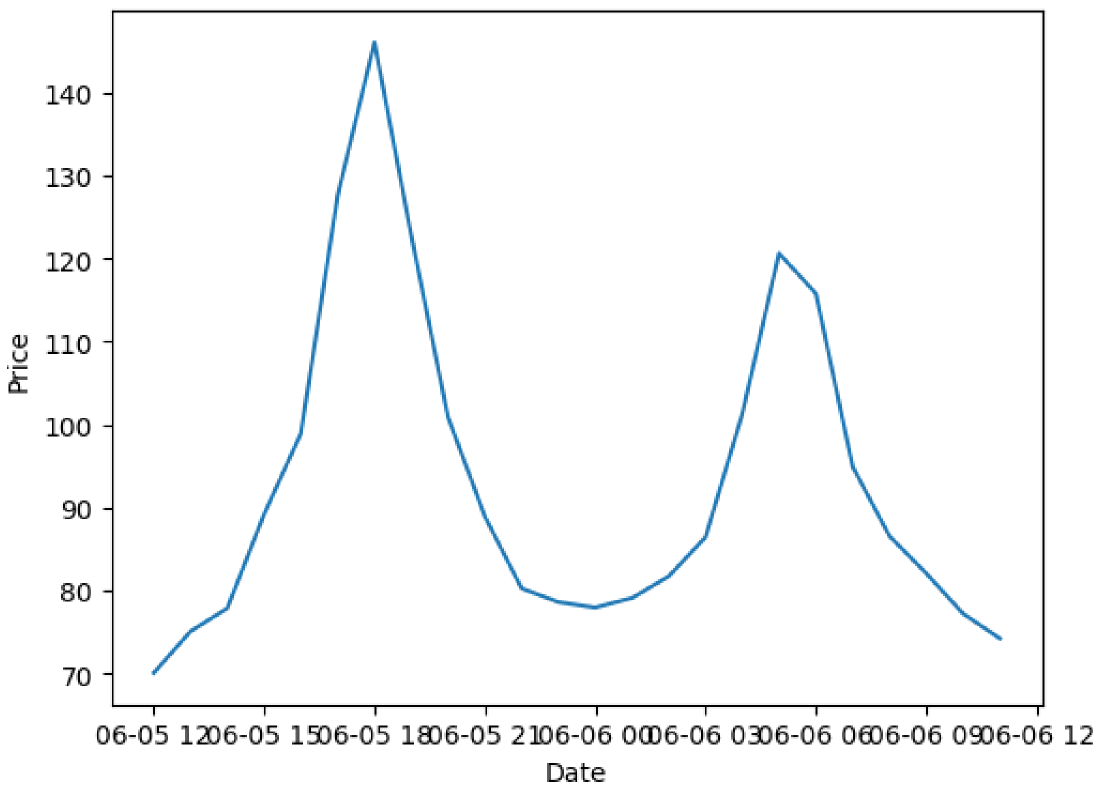
```
X = fetch_market_data()
```

[89]:

```
t_starts, t_ends, prices, units = prep_data(X)
```

[90]:

```
plt.plot(t_starts, prices)
plt.xlabel("Date")
plt.ylabel("Price")
plt.show()
```



2.1 Example 2: RSA Encryption

In this example we show how to use functions to structure the code a little bit better

2.2 Questions

What does the “f” in front a string mean, e.g.: `f"x = {x}"`

Short hand syntax to do “string interpolation” and formating. All occurences of the curly braces are replaced with the result

```
[91]: x = 10
      f"There are {x} apples" # format strings
```

```
[91]: 'There are 10 apples'
```

```
[92]: "There are {} apples".format(x)
```

```
[92]: 'There are 10 apples'
```

```
[93]: "There are " + str(x) + " apples"
```

```
[93]: 'There are 10 apples'
```

[ ]:

```python

```

[94]:

```python
# Functions and Methods
#
# function: Gets some input, does some things and returns some output
# method: Gets some input, does some things and returns some output BUT it also can modify the object it belongs to
#
#
# Object is an instance of a data type (or to be more correct, instance of a "Class")
```

[95]:

```python
cidr = "10.0.0.0/16"
```

[96]:

```python
cidr.split()
```

[96]:

```
['10.0.0.0/16']
```

[97]:

```python
x = cidr.split("/")
x
```

[97]:

```
['10.0.0.0', '16']
```

[98]:

```python
x[0].split(".")
```

[98]:

```
['10', '0', '0', '0']
```

[ ]:

```python

```