ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Introduction to Information Systems and Applications

Course Unit 2: Data processing with python

M. Tzagarakis, V. Daskalou

School of Business Administration

Department of Economics

Flow control

Flow control with `if`

```
if <condition>:
        # block of commands
else:
        # block of commands
```

Comparison operators:	`==`
	`!=`
	`>, >=`
	`<, <=`

Logical operators:	`not`, π.χ. not(a) `and`, π.χ. (a and b) `or`, π.χ. (a or b)
Member operators	`in, not in`

```
grade= int(input(“Give your grade: "))
if grade>= 5:
    print("Pass :-)")
else:
    print("Fail :-(")
```

Attention: The indent in Python is important! It specifies the block of commands

if-elif-else

```
if <condition>:
   <block of commands>
elif <condition>:
   <block of commands>
else:
   <block of commands>
```

```
weather = input(“How is the weather today? ")
if weather == “rainy":
    print(“Take your umbrella")
elif weather == “cold":
    print(“Take your jacket")
elif weather == “sunny":
    print(“Wear your sun cream")
else:
    print(“Have a nice day!")
```

for loops

```
for <variable> in <range>:
        <block of commands>

for counter in range(5):
        print("hello world")
print("outside for")
```

Tip: We use *for* loop when the number of iterations is known

`range(from, to, step)`

- `from, step`: optional
- `to`: obligatory
- `from, to, step`: **integers**

Examples of range():

- `range(10)`: [0,1,2,3,4,5,6,7,8,9]
- `range(1, 7)`: [1,2,3,4,5,6]
- `range(0, 30, 5)`: [0,5,10,15,20,25]
- `range(5, -1, -1)`: [5,4,3,2,1,0]

Example use of `for`

```
# calculate average of integers from start to stop-1
start=5
stop=11
#we dont use variable name sum, since it is a reserved word
sumi=0
n=0
for x in range(start,stop):
  n=n+1
  print("x:{0} sum:{1} n{2}:".format(x,sumi,n))
  sumi=sumi+x
print("avg:", sumi/n) #outside for
```

Example use of `for` and `if`

```
# calculate average of even integers from start to stop-1
start=5
stop=100
sumi=0
n=0 #number of even numbers
for x in range(start,stop):
    if x%2==0 : # a number is even when its mod with 2 is
0
        sumi=sumi+x
        n=n+1
        print("x:{0} sum:{1} n{2}:".format(x,sumi,n))
print("avg :",sumi/n)
```

`while` loop

```
while <condition>:
      <command1>
      <command2>
[else:
      <command1 when loop ends>]
```

Tip: We use *while* when we don’t know the number of iterations and we want to control the condition

```
#stops only if the user gives no
msg=input("Give a string: ")
while msg != 'no':
    print(msg*3+'!!!')
    msg=input("Give a string: ")
else:
    print("Thank you!")
```

Example use of while and if

```
# a guess game: use tries to guess the number
number = 23
running = True
while running:
    guess = int(input('Give a number: '))
        if guess == number:
                print('Congrads! You guess it!.')
                running = False # while stops here
        elif guess < number:
                print('No, is greater.')
        else:
                print(‘No, is smaller.')
else:
    print('Loop terminated.')
```

while loop with break, continue

```
# this runs till you give quit and check input length
while True:
  s = input('Give something : ')
  if s == 'quit':
      break # terminates the loop
  if len(s) < 3:
      print('to small!')
      continue #skips the rest of command and go to while
  print('Ok’) #will not be printed if user gives a small string
print('loop terminated’) #outside the loop
```

Lists

Lists

- A collection of data ordered and changeable. Allows duplicate members.
- Allows members of different types
- List members inside square brackets [], separated by commas
- Lists represent the concept of array
- Examples:

```
fruits= ["fig", "apple", "pear", 1, 2, 3]
2Dtable= [ [ 2, 3, 5] , [ 1, 4, 7 ]]
```

Lists: create, slices, copy, concat

```
>>> shoplist = ['apple', 'mango', 'carrot', 'banana', 'pear', 'fig']
>>> shoplist[0] #[0] is the first list item
'apple’
>>> shoplist[-1] # [-1] is the last list item
'fig'
>>> shoplist[0:2]
['apple', 'mango']
>>> shoplist[0:6:2] #from item 0 to 5, step 2
['apple', 'carrot', 'pear’]
>>> shoplist[6:2:-1] #from item 6 to 3, step -1
['fig', 'pear', 'banana']
>>> shoplist[-1] # 1 before end
‘fig'
>>> shoplist[1:-2] #from 1 to 2nd before end
['mango', 'carrot', 'banana’]
>>> new_list=shoplist[:] #make a copy of shoplist
>>> huge_list=new_list + shoplist #add two lists together
```

Loop through a list with `for` & `in`

```
>>> for each in shoplist:
   print(each)

apple
mango
carrot
banana
pear
```

```
>>> for each in shoplist:
        if each[1]=='a':
                print(each)


mango
carrot
banana
```

- Membership operator `in` can also be used to check whether specific item is in a list
- `each` is just a variable name that make code easy to read

Loop through 2D lists

```
# create a 2D list
a = [ [ 2, 3, 5] , [ 1, 4, 7 ] ]
print ("List before: a ={}".format(a))
# find list dimensions
rows = len(a)
cols = len(a[0])
# run the list and add 1 in every item
for row in range(rows):
    for col in range(cols):
        a[row][col] += 1
# print list items
print ("List after: a ={}".format(a))
```

List operations

```
>>> shoplist
['apple', 'mango', 'carrot', 'banana', 'pear', 'fig']
>>> len(shoplist)
6
>>> shoplist.reverse()
>>> shoplist
['fig', 'pear', 'banana', 'carrot', 'mango', 'apple']
>>> shoplist.append('orange') # add at the end
>>> shoplist
['fig', 'pear', 'banana', 'carrot', 'mango', 'apple', 'orange']
>>> shoplist.insert(3,'grapes') # insert before position 3
>>> shoplist
['fig', 'pear', 'banana', 'grapes', 'carrot', 'mango', 'apple', 'orange']
>>> shoplist.pop(5) # delete item from index 5
'mango'
>>> shoplist
['fig', 'pear', 'banana', 'grapes', 'carrot', 'apple', 'orange']
```

List methods:

- `len(list)` : list length
- `max(list) & min(list)`: max and min values in list
- `list.reverse()` : reverse list (changes original)
- `list.append(x)`: add item *x* at the end of list
- `list.insert(i,x)`: insert item *x* at given position *i* in list
- `list.pop([i])`: remove item from the given position *i* or from the end of the list
- `list.index(x)`: return zero-based index of first item whose value is equal to *x*.
- `list.remove(x)`: remove first item whose value is equal to *x*
- `list.sort()`: sorts original list
- `sorted(list)`: sorts without changing original
- `list.count(x)`: return number of times *x* appears in list.

statistics module

```
from statistics import *
```

mean(list)->Arithmetic mean or average
median(list)->Median
mode(list)->Mode

pstdev(list)->Standard deviation (population)

pvariance(list)->Variance (population)

stdev(list)->Standard deviation (sample)

variance(list)->Variance (sample)

- Basic functions
  - max(list)
  - min(list)
  - sum(list)

Tuples

- Sequence of immutable data, used as constants
- Tuples are like lists but **can't** be modified
- Tuple members inside parentheses (), separated by commas, different types
- Same operations like lists
- Examples:

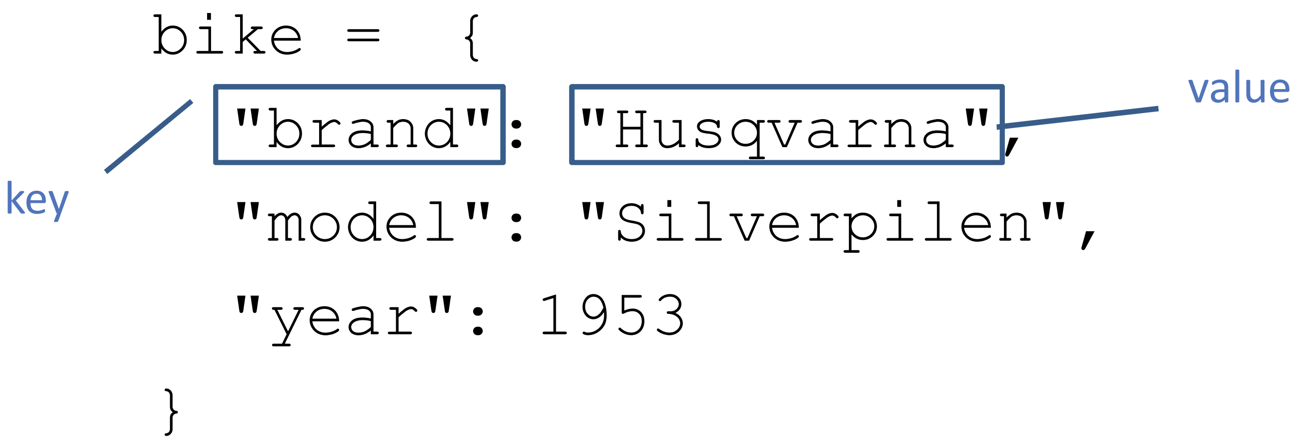
```
tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = () #empty tuple

tup3 = (50,) #single value tuple, should use one comma
```

Dictionaries

- Collection of data, unordered, changeable and indexed by *keys*.
- Keys can be integers, string or other objects and are unique
- Written with curly brackets {}, have keys and values.



```
bike =  {
   "brand": "Husqvarna",
   "model": "Silverpilen",
   "year": 1953
}
```

Dictionary operations

```
>>> bike =          {
  "brand": "Husqvarna",
  "model": "Silverpilen",
  "year": 1953
} # create a dictionary
>>> bike
{'brand': 'Husqvarna', 'model': 'Silverpilen', 'year': 1953}
>>> bike['model'] # Get the value of the model key
'Silverpilen’
>>> x = bike.get("brand") # get the value of brand key and assign it to x
>>> x
'Husqvarna'
>>> bike['year']=1955 #change value of year key
>>> bike
{'brand': 'Husqvarna', 'model': 'Silverpilen', 'year': 1955}
```

Dictionary: keys, values, items & copy

```
# use keys(), values() and items() in dictionary
# to print keys, values and key-value pairs
bike =  {
  "brand": "Husqvarna",
  "model": "Silverpilen",
  "year": 1953
}
print("Out-dict1: {}".format(bike.keys()))
print("Out-dict2: {}".format(bike.values()))
print("Out-dict3: {}".format(bike.items()))
new_bike=bike.copy() # copy an existing dictionary to a new variable
print("Out-dict4: {}".format(new_bike))
```

Loop through a Dictionary

```
bike =    {
  "brand": "Husqvarna",
  "model": "Silverpilen",
  "year": 1953
}
#printing keys
for key in bike:
    print("Out-dict5: {}".format(key))
#printing values
for key in bike:
    print("Out-dict6: {}".format(bike[key]))
#printing both keys and values
for key, value in bike.items():
        print("Out-dict7: {} {}".format(key,value))
```

Modify a dictionary

```
>>> bike =       {
  "brand": "Husqvarna",
  "model": "Silverpilen",
  "year": 1953
}
>>> bike["color"]='red’ # add a new key, value pair
>>> bike
{'brand': 'Husqvarna', 'model': 'Silverpilen', 'year': 1953, 'color':
'red'}
>>> bike.pop("color") #remove a key, if present, and return its value
'red'
>>> bike
{'brand': 'Husqvarna', 'model': 'Silverpilen', 'year': 1953}
>>> new_stuff={"year": [1955, 1960], "colors": ["black","white"]}
>>> new_stuff
{'year': [1955, 1960], 'colors': ['black', 'white']}
>>> bike.update(new_stuff) #merge a dictionary with another
>>> bike
{'brand': 'Husqvarna', 'model': 'Silverpilen', 'year': [1955, 1960],
'colors': ['black', 'white']}
```