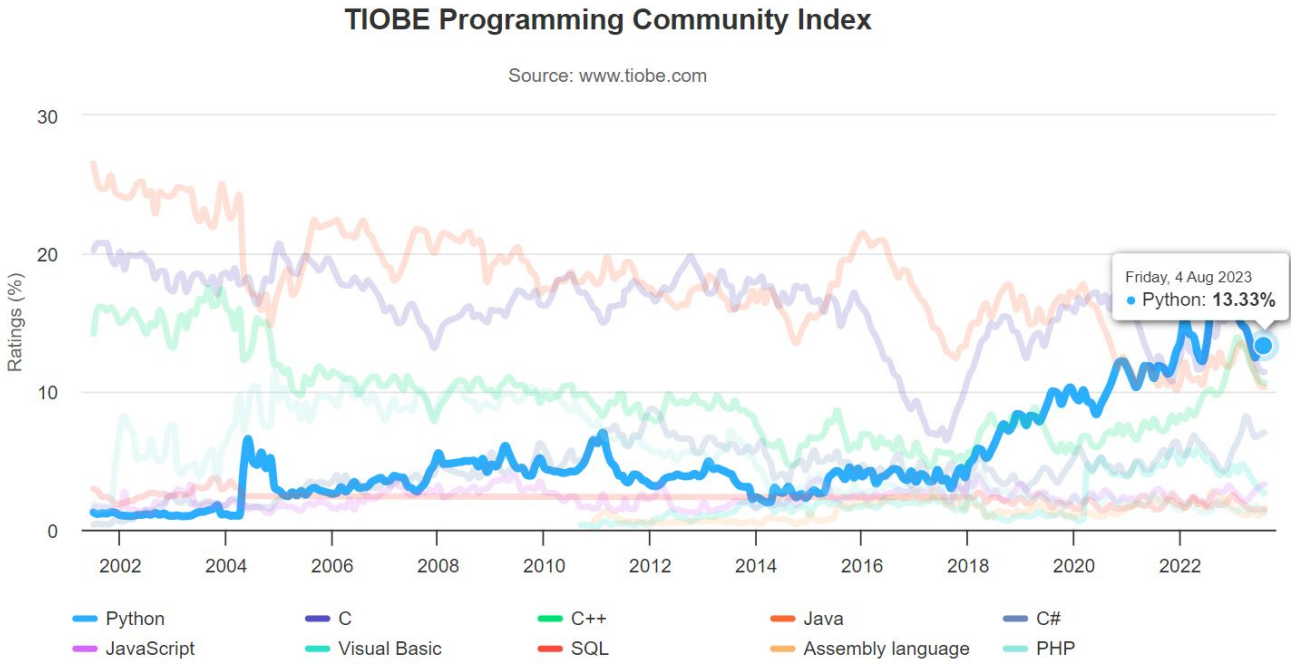
Recitation 0

Internet Technology (Section 01)

About Me

- Name: Alborz Jelvani
- Email: alborz.jelvani@rutgers.edu
- Office Hours: TBA

Python3


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
20
10
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
2022
Friday, 4 Aug 2023
Python: 13.33%
Python
C
C++
Java
C#
JavaScript
Visual Basic
SQL
Assembly language
PHP

Python3 Features

- Object oriented
- **Strongly** typed and **dynamically** typed
- Interpreted

```
jelvani@PC:~$ python3
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello!")
hello!
```

```
someVar = 100
print(type(someVar))
someVar = "a string"
print(type(someVar))
>> <class 'int'>
>> <class 'str'>
```

Control Flow

Indentation: 4 spaces

```
While True:
    if cond1:
        doFunc()
    elif cond2:
        doThis()
    else:
        break
```

Execution begins at top (*main* not required)

```
#This is a comment
items = ["a", "b", "c"]
abc = ""
for letter in items:
    print(letter)
    abc+=letter
print(abc)
```

Types

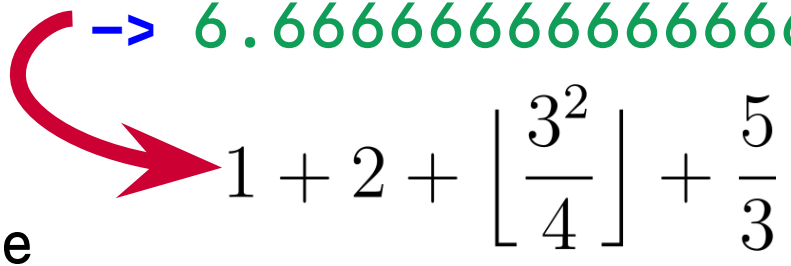
- Strings, integers, floats, bool
  - *str(),int(),float(), bool()*
- Lists
  - Collection of **mutable ordered** elements.
- Set
  - Collection of **mutable unordered** elements. 1 of each elements!
- Tuple
  - Collection of **immutable ordered** elements.
- Dictionary
  - Key-value pairs
- None

```
>>> "string"
>>> 27 #int
>>> 3.14 #float
>>> True #bool
>>> aList = [1,2,3,4,5]
>>> aSet = {1,2,3,4,5}
>>> aTuple = (1,2,3,4,5)
>>> aDict = {1: "one", 2:
    "two", 3: "three"}
>>> a = [None, None]
```

Expressions

```
>>> 1==2 -> False
>>> "hi" == "hi" -> True
>>> True and False or True
   -> True
>>> False or  False and True
   -> False
```

```
>>> 1+2+3**2//4+5/3
-> 6.666666666666667
```



$$1 + 2 + \left\lfloor \frac{3^2}{4} \right\rfloor + \frac{5}{3}$$

```
>>> 10 % 2 == 1
-> False
```

For Loops

- Iterate over *iterables*
  - Some objects implement iteration
    - *range(*start, stop (required), step*)*
    - *list(), str(), tuple(), dict(), set()…*
- Want index?
  - Use *enumerate()*

```
>>> list(enumerate([23, 56, 65, 36]))
    -> [(0, 23), (1, 56), (2, 65), (3, 36)]
```

```
for i in range(10):
    print(i)
```

```
animals = ["moose","marmot","osprey"]
for animal in animals:
    print(animal)
```

```
for idx, val in enumerate(animals):
    print(idx, val)
```

Lists

```
>>> items = [] #or list()
>>> type(items) -> <class 'list'>
>>> items.append(1)
>>> items.append("1")
>>> items.append([3.14, 2.71])
>>> items -> [1, '1', [3.14, 2.71]]
>>> [type(item) for item in items]
   -> [<class 'int'>, <class 'str'>, <class 'list'>]
>>> 3 in items -> False
```

List Manipulation

- Couple ways
  - Indexing
  - Slicing
    - List[first include : last exclude : step]
  - Add elements
    - List.append(item)
    - List.insert(index,item)
  - Remove elements
    - List.remove(target)
    - List.pop(index)

```
>>> a = [2,1,5,3,1]
>>> a[2] -> 5
>>> a[-1] -> 1
>>> a[:3] -> [2,1,5]
>>> a[::2] -> [2,5,1]
>>> a[1:3:2] -> [1]
>>> a+[30,23]
    -> [2,1,5,3,1,30,23]
>>> a.pop(1) -> 1
>>> a.remove(3); a -> [2,5,1]
```

List Comprehensions

- A more concise way to create lists

```
>>> a = [0]*10; a -> [0,0,0,0,0,0,0,0,0,0]
>>> [odd for odd in range(10) if odd%2==1] -> [1,3,5,7,9]
```

- Syntax
  - `list2 = [expr(element) for element in list1 filter]`
  - ‘for’ clause to iterate over iterable
  - Expression to manipulate elements. The output will populate the new list
  - An optional filter composed of ‘if’ clauses

Lists (useful methods)

- Get length of list
- Reverse list
- Min/max
- *count(target)*
  - Number of occurrences for *target*
- Sort
- Clear

```
>>> a = [2,1,5,3,1]
>>> len(a) -> 5
>>> a.reverse(); a
    -> [1, 3, 5, 1, 2]
>>> max(a) -> 5
>>> a.count(1) -> 2
>>> a.sort(); a ->
    [1, 1, 2, 3, 5]
>>> a.clear(); a -> []
```

Binary Data

- *bytes* type
  - **immutable** sequence of bytes

```
>>> bite = b"alborz" #or bytes()
>>> bite[1] -> 108
>>> bite.hex() -> '616c626f727a'
>>> bite[3] = 23
```

Nope!

- *bytearray* type
  - **mutable** sequence of bytes

```
>>> bite = bytearray('alborz',
'utf-8')
>>> bite[1] -> 108
>>> bite.hex() -> '616c626f727a'
>>> bite[3] = 75
>>> bite.decode() -> 'albKrz'
```

bytearray Manipulation

- Similar to strings and lists

```
>>> packet = bytearray('payload', 'utf-8')
>>> len(packet) -> 7
>>> packet.append(5)
>>> packet.decode() -> 'payload\x05'
>>> packet.reverse()
>>> packet.decode() -> '\x05daolyap'
```

Functions

- Polymorphism

```
def add(a,b):
    return a + b
print(add(1,2))
print(add("a ", "string"))
```

- Default args and keyword args

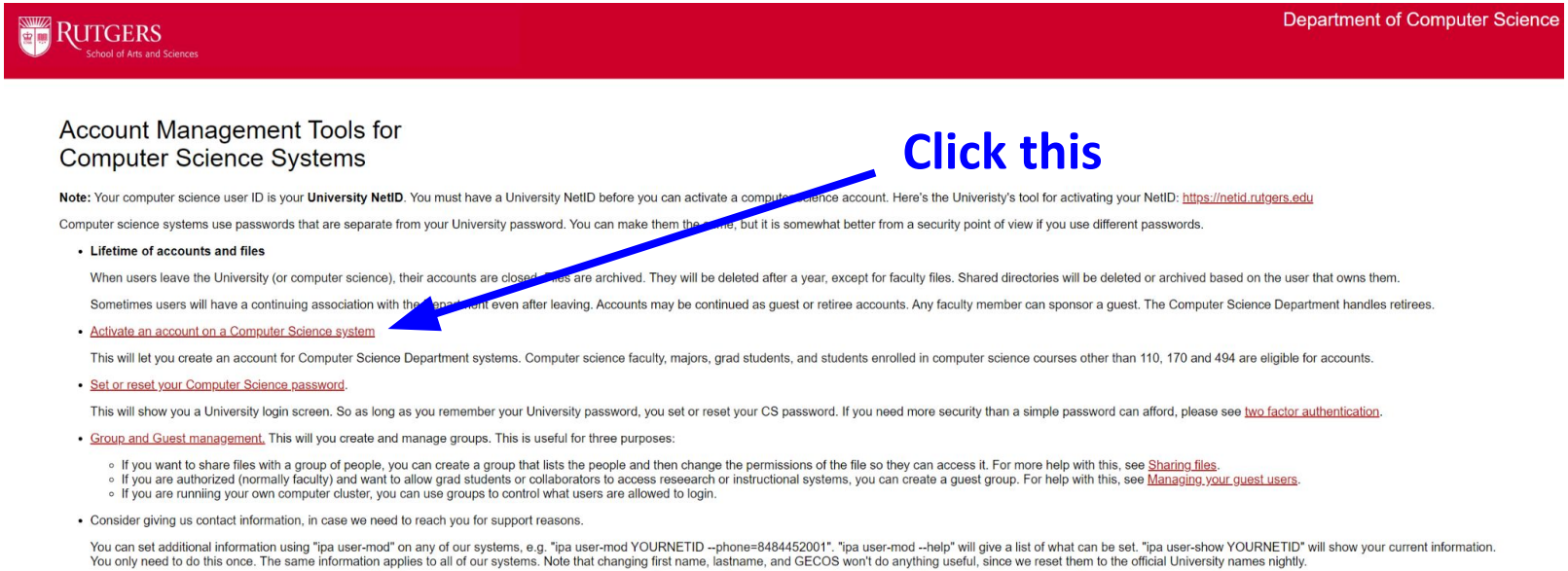
```
def add(a=4,b=5):
    return a + b
print(add(b=1))
```

- Arbitrary args

```
def merge(*lists):
    a=[]
    for l in lists:
        a+=l
    return a
print(merge([1],[2],[3]))
```

How to make a Rutgers CS account

- https://services.cs.rutgers.edu/accounts/



Weblogin for Rutgers CS

- https://weblogin.cs.rutgers.edu/
  - Login to Rutgers CS account
  - Pick a machine

- https://report.cs.rutgers.edu/nagiosnotes/iLab-machines.html
  - View machine status