

PDSP 2022, Lecture 02, 04 August 2022 -- In class

Set up the table as a list

- Assign a value to a variable: `variable = value`
- Each entry is a row
- Each row is a tuple of columns
- (graduation year, programme, domain, pay package)
- Pay package is an integer, all other columns are text (String)


```
In [11]: row = 72
```

```
In [12]: row
```

```
Out[12]: 72
```

```
In [13]: count = 0
for row in plist:
    count = count+1
```

```
In [14]: count
```

```
Out[14]: 73
```

```
In [15]: row
```

```
Out[15]: ('2020-21', 'M.Sc Data Science', 'Banking-Finance', 1500000)
```

```
In [16]: x = x + 1
```

```
-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_185787/381015178.py in <module>
----> 1 x = x + 1

NameError: name 'x' is not defined
```

Find the maximum pay package

- Initialize `maxpay` to 0
- Update `maxpay` whenever pay package in current row exceeds current maximum
 - Extract pay package from tuple of values of current row using positional index
 - In Python, indices always start with 0, so four entries in the tuple have indices 0,1,2,3
- `if` specifies conditional execution
 - `if` conditional-expression -- expression evaluates to `True` or `False`
 - `:` and indentation indicate scope of conditional execution
 - Indented commands (update of `maxpay`) happens only when `if` condition evaluates to `True`

```
In [17]: # maxpay = 0
maxpay = plist[0][3] # Set to pay in first row
for row in plist:
    thispay = row[3]
    if thispay > maxpay:
        maxpay = thispay
```

```
In [18]: maxpay
```

```
Out[18]: 3300000
```

```
In [19]: newlist = [5,6,7]
for row in newlist:
    print(row)
```

```
5
6
7
```

Average pay package

- Iterate to compute sum of all paypackages
- Average is total divided by count

```
In [20]: sumpay = 0
for row in plist:
    sumpay = sumpay + row[3]
```

```
In [21]: sumpay
```

```
Out[21]: 99847392
```

```
In [22]: count = 0
sumpay = 0
for row in plist:
    count = count + 1
    sumpay = sumpay + row[3]
```

```
In [23]: average = sumpay/count
```

```
In [24]: average
```

```
Out[24]: 1367772.493150685
```

Number of placements with pay package above the average

- *Filtered iteration*
- Update the count only if the current pay package is \geq average

```
In [25]: aboveaverage = 0
for row in plist:
    if row[3] >= average:
        aboveaverage = aboveaverage + 1
```

```
In [26]: aboveaverage
```

```
Out[26]: 25
```

Check if there is a placement in Logistics

- Use a boolean variable
- Set to `True` if the value we are searching for is found
 - Note `==` for equality check, to distinguish from `=` for assignment
- Scans the entire list even if the value is found early

```
In [27]: logistics = False
for row in plist:
    domain = row[2]
    if domain == "Logistics":
        logistics = True
```

```
In [28]: logistics
```

```
Out[28]: True
```

```
In [29]: logistics = False
for row in plist:
    domain = row[2]
    if domain == "Logistics":
        logistics = True
        break
```

```
In [30]: logistics
```

```
Out[30]: True
```

```
In [31]: row
```

```
Out[31]: ('2017-18', 'M.Sc Appl. Math', 'Logistics', 1250000)
```

Conditional iteration -- while

- `while` conditional-expression
 - Like `if`, but at the end of the block, the condition is checked again
 - If the condition is `False`, the loop terminates
- Does not automatically proceed from one list item to the next
 - Maintain an index and extract list item by position
 - Recall that positions in Python start from `0`
 - Need to increment the index manually
- Two checks for loop termination
 - Required value has been found
 - No more items to process (index has reached the value of `count`)

```
In [32]: logistics = False
rowindex = 0
# while rowindex < count and logistics == False:
while rowindex < count and not(logistics):
    row = plist[rowindex]
    if row[2] == "Logistics":
        logistics = True
    else:
        rowindex = rowindex + 1
    # rowindex = rowindex + 1
```

```
In [33]: logistics, rowindex
```

```
Out[33]: (True, 5)
```

How many placements in Banking-Finance?

- Filtered iteration again
- Increment count only if domain is "Banking-Finance"

```
In [34]: bfcoun = 0
for row in plist:
    if row[2] == "Banking-Finance":
        bfcoun = bfcoun + 1
```

```
In [35]: bfcoun
```

```
Out[35]: 41
```

Domain with maximum placements

First count the number of placements in each domain

- Use a *dictionary* to record domain-wise counts
 - A collection of key:value pairs
 - Initialize to empty dictionary, {}
 - For each row, check if current domain is already a key in the dictionary
 - If domain is present, increment the count
 - If domain is not present, create a new key, set count to 1

```
In [36]: domaincounts = {}
for row in plist:
    domain = row[2]
    if domain in domaincounts:
        # if domain in domaincounts.keys():
        domaincounts[domain] = domaincounts[domain] + 1
    else:
        domaincounts[domain] = 1
```

```
In [37]: domaincounts
```

```
Out[37]: {'CS': 10,
'Manufacturing': 2,
'Banking-Finance': 41,
'Logistics': 3,
'Analytics': 17}
```

Now find the domain with the maximum count

- Iterate through the keys of the dictionary
- Keep track of both the key (domain name) and the value (domain count)

```
In [38]: maxdname = ""
maxdcoun = 0
# for d in domaincounts.keys():
for d in domaincounts:
    if domaincounts[d] > maxdcoun:
        maxdname = d
        maxdcoun = domaincounts[d]
```

```
In [39]: maxdname,maxdcount
```

```
Out[39]: ('Banking-Finance', 41)
```

Domain with best average pay package

Compute domain wise pay package total

- We already have domain wise counts
- Now compute total pay, domain wise

```
In [40]: domainsums = {}
for row in plist:
    domain = row[2]
    thispay = row[3]
    if domain in domainsums.keys():
        domainsums[domain] = domainsums[domain] + thispay
    else:
        domainsums[domain] = thispay
```

```
In [41]: domain_avg_pay = {}
for d in domainsums.keys():
    domain_avg_pay[d] = domainsums[d]/domaincounts[d]
```

Compute domain wise average pay

- Iterate through keys of domain counts/domain sums
- Create a new dictionary that records domain average pay

```
In [42]: domain_avg_pay
```

```
Out[42]: {'CS': 1482393.6,
'Manufacturing': 730000.0,
'Banking-Finance': 1474816.0,
'Logistics': 1166666.6666666667,
'Analytics': 1152705.8823529412}
```

Compute domain with maximum average pay

- Iterate through keys of domain average pay dictionary
- Keep track of domain name and value

```
In [43]: maxavgdname = ""
maxavgdpay = 0
for d in domain_avg_pay.keys():
    if domain_avg_pay[d] > maxavgdpay:
        maxavgdname = d
        maxavgdpay = domain_avg_pay[d]
```

```
In [44]: maxavgdname,maxavgdpay
```

```
Out[44]: ('CS', 1482393.6)
```