KIT100 PROGRAMMING PREPARATION

Lecture Eight:

- *Reading and writing with files*
- *The dictionary type*
- *Early portfolio task hints*

Lecture Objectives

- Reading and writing files
- The dictionary type
- Portfolio task help (up to 5.2PP)

Files

- Sometimes you may want to **store data** longer term
  - we do this all the time (for example Word and Excel documents).
  - imagine how inefficient it would be to have to type in your data (e.g. an essay) while the program is running but lose then data when the program stops.

- Data can be stored in **secondary storage** in the form of *files*.
  - A file is stored in a location on a storage device (called the **path**) and has a name that is used to differentiate it from other files in the same location.

Files

- Python allows us to *read* from and *write* to files easily. We simply specify:
  - what file we are interested in (the name),
  - how we are *accessing* it (whether reading or writing, or both), called the **mode**
  - then read and/or write to/from the file
  - close the file

- Closing the file is important
  - as sometimes when writing to a file some of the data may be buffered in memory, and it is only written (flushed) to the file when tidying up (closing).
  - If you open a file for writing then you have *exclusive* write-access to it, failing to close it may prevent other writers from having access.

Opening a file

Syntax

- `f = open(filename, mode)`
  - e.g. `f = open('myfile','r')`

  - `f` is a variable; it refers to a file *object* (called a *descriptor*)
  - `mode` *can be:*
    - *'r' – open the file for reading*
    - *'w' – open the file for writing (existing file is overwritten)*
    - *'a' – open the file for appending (add sth.)*
    - *'r+' – open the file for reading* ***and*** *writing*

Closing a file

- If you forget to close a file,
  - python will *eventually* close the file for you (which frees up resources, flushes 'buffers' if writing etc.
  - good programming will include closing the file properly after you have finished reading and writing

E.g.

```
f = open(“myfile", 'r’)

…

f.close()
```

Reading from a file

- **Read a file** - You can read the entire contents of a file at once with `read`
  - e.g.

```
allLines = f.read()
```

(*if the end of the file is reached, f.read() returns an empty string*)

We are only considering text files, but files can also store binary (raw) data.

Open, close, and read from a file

myfile.txt

```
This is the first line of an example file
This is the second line of an example file
This is the third line of an example file
This is the fourth line of an example file
This is the fifth line of an example file
This is the sixth line of an example file
```

```python
'''This program demonstrates reading all data
from a file '''

f = open("myfile.txt", 'r')

allLines = f.read()

f.close()

print("Here is the data from the file:")

print(allLines)
```

Outputs Ln: 13 Col: 0

```
>>>
= RESTART: /Users/czh513/Desktop/KIT001/Teaching in 2020/1 Lecture/week8/w8 lect
ure examples/readFile_p8.py
Here is the data from the file:
This is the first line of an example file
This is the second line of an example file
This is the third line of an example file
This is the fourth line of an example file
This is the fifth line of an example file
This is the sixth line of an example file
```

an empty string

Reading from a file

Read a line - To read individual lines **one-at-a-time** (which is the most common way to do read them),

– use e.g.:

```
line = f.readline()
```

(*line* and *f* are variable names)

- Each call to `readline()` will return a different (the next) line from the file.
- Eventually `readline()` will return an empty string when there are no more lines to read
  – 'blank lines' will return a string that contains only the newline character ('\n')

Reading from a file


```python
'''This program demonstrates reading data
a line at a time from a file
'''

# example 1

f = open("myfile.txt",'r')

print("here is the data from the file, a line at a time:")
print()

# note: each call to readline() will return a different (the next) line from the file.
line = f.readline()
print(line)

line = f.readline()
print(line)

line = f.readline()
print(line)

line = f.readline()
print(line)

line = f.readline()
print(line)

line = f.readline()
print(line)

f.close()
```

myfile.txt

```
This is the first line of an example file
This is the second line of an example file
This is the third line of an example file
This is the fourth line of an example file
This is the fifth line of an example file
This is the sixth line of an example file
```

Outputs

```
here is the data from the file, a line at a time:

This is the first line of an example file

This is the second line of an example file

This is the third line of an example file

This is the fourth line of an example file

This is the fifth line of an example file

This is the sixth line of an example file

>>>
```

an empty string
(print the blank line)

Reading from a file

Note here we're overriding the end character to NOT insert a new line for each print.

```
# example 2 - avoid to return a new line

f = open("myfile.txt",'r')

print("here is the data from the file, a line at a time:")
print()

line = f.readline()
print(line, end='') # add end='' to aviod to return a new line

line = f.readline()
print(line, end='')

line = f.readline()
print(line,end='')

line = f.readline()
print(line,end='')

line = f.readline()
print(line,end='')

line = f.readline()
print(line,end='')

line = f.readline() #return an empty when no more lines to read
print(line,end='')

f.close()
```

Outputs

```
here is the data from the file, a line at a time:

This is the first line of an example file
This is the second line of an example file
This is the third line of an example file
This is the fourth line of an example file
This is the fifth line of an example file
This is the sixth line of an example file
>>>
```

Reading from a file

- Why `readline()` will insert a new line for each print?

Check the raw variables

Open terminal
-> find “myfile.txt” path
-> commend “od -a myfile.txt”

Space

New line

```
Desktop — -bash — 97×42
Last login: Tue Feb 18 15:56:17 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) ict-02yc0pkjgh7:~ czh513$ ls
Applications                    Movies
Applications (Parallels)        Music
Creative Cloud Files            OneDrive - University of Tasmania
Desktop                         Parallels
Documents                       Pictures
Downloads                       Public
Dropbox                         Sites
Google Drive                    University of Tasmania
Library                         ml-agents
MADRL                           opt
MAgent
(base) ict-02yc0pkjgh7:~ czh513$ cd Desktop
(base) ict-02yc0pkjgh7:Desktop czh513$ od -a myfile
od: myfile: No such file or directory
od: myfile: Bad file descriptor
(base) ict-02yc0pkjgh7:Desktop czh513$ od -a myfile.txt
0000000    T   h   i   s  sp   i   s  sp   t   h   e  sp   f   i   r   s
0000020    t  sp   l   i   n   e  sp   o   f  sp   a   n  sp   e   x   a
0000040    m   p   l   e  sp   f   i   l   e  nl   T   h   i   s  sp   i
0000060    s  sp   t   h   e  sp   s   e   c   o   n   d  sp   l   i   n
0000100    e  sp   o   f  sp   a   n  sp   e   x   a   m   p   l   e  sp
0000120    f   i   l   e  nl   T   h   i   s  sp   i   s  sp   t   h   e
0000140   sp   t   h   i   r   d  sp   l   i   n   e  sp   o   f  sp   a
0000160    n  sp   e   x   a   m   p   l   e  sp   f   i   l   e  nl   T
0000200    h   i   s  sp   i   s  sp   t   h   e  sp   f   o   u   r   t
0000220    h  sp   l   i   n   e  sp   o   f  sp   a   n  sp   e   x   a
0000240    m   p   l   e  sp   f   i   l   e  nl   T   h   i   s  sp   i
0000260    s  sp   t   h   e  sp   f   i   f   t   h  sp   l   i   n   e
0000300   sp   o   f  sp   a   n  sp   e   x   a   m   p   l   e  sp   f
0000320    i   l   e  nl   T   h   i   s  sp   i   s  sp   t   h   e  sp
0000340    s   i   x   t   h  sp   l   i   n   e  sp   o   f  sp   a   n
0000360   sp   e   x   a   m   p   l   e  sp   f   i   l   e  nl
0000376
(base) ict-02yc0pkjgh7:Desktop czh513$
```

Reading – problem?

- The previous example had prior knowledge on how many lines to read from the file (six), but in general we don't know how many lines to read!
- We need a more general approach – use **loops**!

As the `readline()` gives us an empty string when the end of the file is reached, we can use that as a condition for a while loop!

while loop reading from file


```
f = open("myfile.txt",'r')

line = f.readline()
count = 1

while line != '':
    print(count,": ",line, end='')
    line = f.readline()
    count += 1

f.close()
```

Q: *Why are we including end here?*

A: *each line read from the file also includes a newline character (\n) at the end of the line, so we don't want to print two of them!*

```
'''This program demonstrates reading data
a line at a time from a file using a while loop
'''

f = open("myfile.txt",'r')

line = f.readline()
count = 1

while line != '':
    print(count,": ",line,end='')
    # print(count,": ",line)
    line = f.readline()
    count += 1

f.close()
```

Output

```
>>>
= RESTART: /Users/czh513/Desktop/KIT001_S2 2020/1 Lectu
ileRead_p14.py
1 :  This is the first line of an example file
2 :  This is the second line of an example file
3 :  This is the third line of an example file
4 :  This is the fourth line of an example file
5 :  This is the fifth line of an example file
6 :  This is the sixth line of an example file
>>>
```

for loop reading from file


```
f = open("myfile.txt",'r')

count = 1

for line in f:

    print(count,": ",line, end='')
    count += 1

f.close()
```

*f here evaluates to a sequence of **lines** from the file that can then be iterated over by the for loop*

```
'''This program demonstrates reading data
a line at a time from a file using a for loop
'''

f = open("myfile.txt",'r')

count = 1

for line in f:

    print(count,": ",line, end='')
    count += 1

f.close()
```

Output

```
1 :  This is the first line of an example file
2 :  This is the second line of an example file
3 :  This is the third line of an example file
4 :  This is the fourth line of an example file
5 :  This is the fifth line of an example file
6 :  This is the sixth line of an example file
>>>
```

Resetting the current read position

- **Reset** - Occasionally you may need to start again where you a reading from
  - e.g. when you reach the end of the file with a readline(), maybe you want to start at the first line again.
  - Syntax `f.seek(0)`

```
f = open('myfile.txt','r')

line = f.readline()
print(line, end='')

f.seek(0)

line = f.readline()
print(line, end='')

f.close()
```

This sets the 'read' position of the file to the first character (offset of zero from the start of the file).

```
'''This program demonstrates seeking to a position in a file'''

# example 1

f = open('myfile.txt','r')

line = f.readline()
print(line,end='')

f.seek(0) # resit to the first line

line = f.readline()
print(line,end='')

f.close()
```

Ln: 2

```
>>>
= RESTART: /Users/czh513/Desktop/KIT001/Teaching in 2020/1 Lecture/we
ure examples/readSeek_p16.py
This is the first line of an example file
This is the first line of an example file
```

Output:

```
This is the first line of an example file
This is the first line of an example file
```

Resetting the current read position

Find the outputs for f.seek(0) and f.seek(128)

Addr	Offset 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	h	i	s	sp	i	s	sp	t	h	e	sp	f	i	r	s
16	t	sp	l	i	n	e	sp	o	f	sp	a	n	sp	e	x	a
32	m	p	l	e	sp	f	i	l	e	nl	T	h	i	s	sp	i
48	s	sp	t	h	e	sp	s	e	c	o	n	d	sp	l	i	n
64	e	sp	o	f	sp	a	n	sp	e	x	a	m	p	l	e	sp
80	f	i	l	e	nl	T	h	i	s	sp	i	s	sp	t	h	e
96	sp	t	h	i	r	d	sp	l	i	n	e	sp	o	f	sp	a
112	n	sp	e	x	a	m	p	l	e	sp	f	i	l	e	nl	T
128	h	i	s	sp	i	s	sp	t	h	e	sp	f	o	u	r	t
144	h	sp	l	i	n	e	sp	o	f	sp	a	n	sp	e	x	a
160	m	p	l	e	sp	f	i	l	e	nl	T	h	i	s	sp	i
176	s	sp	t	h	e	sp	f	i	f	t	h	sp	l	i	n	e
192	sp	o	f	sp	a	n	sp	e	x	a	m	p	l	e	sp	f
208	i	l	e	nl	T	h	i	s	sp	i	s	sp	t	h	e	sp
224	s	i	x	t	h	sp	l	i	n	e	sp	o	f	sp	a	n
240	sp	e	x	a	m	p	l	e	sp	f	i	l	e	nl		

Resetting the current read position

- Example – `f.seek(128)`

readSeek_p16.py - /Users/czh513/Desktop/KIT001/Tea

```
# example 2
print()

f = open('myfile.txt','r')

line = f.readline()
print(line,end='')

f.seek(128) # resit depending on the table

line = f.readline()
print(line,end='')

f.close()
```

Output

```
This is the first line of an example file
his is the fourth line of an example file
>>>
```

128	h	i	s	sp	i	s	sp	t	h	e	sp	f	o	u	r	t

Writing to a file

- **Writing to a file** - Two modes:
  1. Creating a new file (or overwriting an existing file)
  2. Appending data to the end of an existing file

Create/overwrite:

```
f = open('myfile.txt','w')
```

Appending:

```
f = open('myfile.txt','a')
```

Writing (overwriting) to a file

- **All** writes to a file are interpreted as **strings**, so non-string values need to be converted to strings first (using the str function)
- File write syntax:

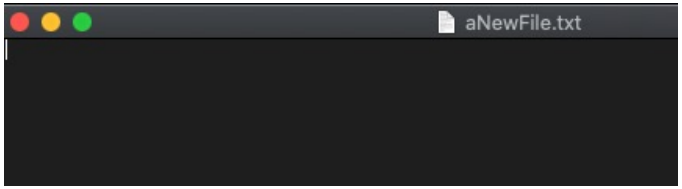
```
f.write(string)
```

e.g.

```
f = open('aNewFile.txt','w')

age = 30
name = “Jimmy Cao”
f.write("Age:" + str(age) + "\n")
f.write("Name:" + name)

f.close()
```




```
'''This program demonstrates writing data
a line at a time to a file '''

f = open('aNewFile.txt','a')

age = 30
name = "Zehong Jimmy Cao"
f.write("Age:" + str(age) + "\n")
f.write("Name:" + name + "\n")


f.close()
```

Output aNewFile.txt

```
Age:30
Name:Zehong Jimmy Cao
```

Output: *aNewFile* contents:

```
Age:30
Name:Jimmy Cao
```

Writing (appending) to a file

Assuming *aNewfile* already exists from the previous slide:

```
age = 2021
name = “Bill J”
f = open('aNewFile.txt','a')
f.write("Age:" + str(age) + "\n")
f.write("Name:" + name)
f.close()
```

Output: *aNewFile* contents*:*

```
Age:30
Name:Jimmy Cao
Age:2021
Name:Bill J
```

appendFile_p21.py - /Users/czh513/Deskt

```
'''This program demonstrates writing data
a line at a time to a file'''

age = 2021
name = "Bill J"

f = open('aNewFile.txt','a')

f.write("Age:" + str(age)+ "\n")
f.write("Name:" + name)

f.close()
```

Output aNew

```
Age:30
Name:Jimmy Cao
Age:2021
Name:Bill J
```

For fun – ask users for input e.g. their name, age, gender, phone number, and then store the details in a file. You can create a simple database, asking for data and then separately displaying data for a given person or phone number etc..

Python Dictionaries

Now that you are happy with lists and read/writing files, the last major useful datatype in Python you may come across is the ***dictionary***.

- A **dictionary** is similar to a list, **except** each item in the dictionary is associated with a **key**, not an index
  - Recall a **list** *orders* each item by an index value, starting at index 0.
  - In **dictionary**, there is no ordering, instead to recall an item, you must know its key

```
>>> list=["a", "b","c"]
>>> list[0]
'a'
>>> Dict = {1:'a', 2:'b', 3:'c'}
>>> Dict[1]
'a'
```

- Syntax:

  `dictionaryVariable = {key1:value1, key2:value2,… keyn:valuen}`

  E.g.

  myDict = {'apples':1, 'oranges':5, 'bananas':12}

  - Python defines the dictionary as **a specific type - 'dict'**

```
type(myDict)
<class 'dict'>
```

Python Dictionaries

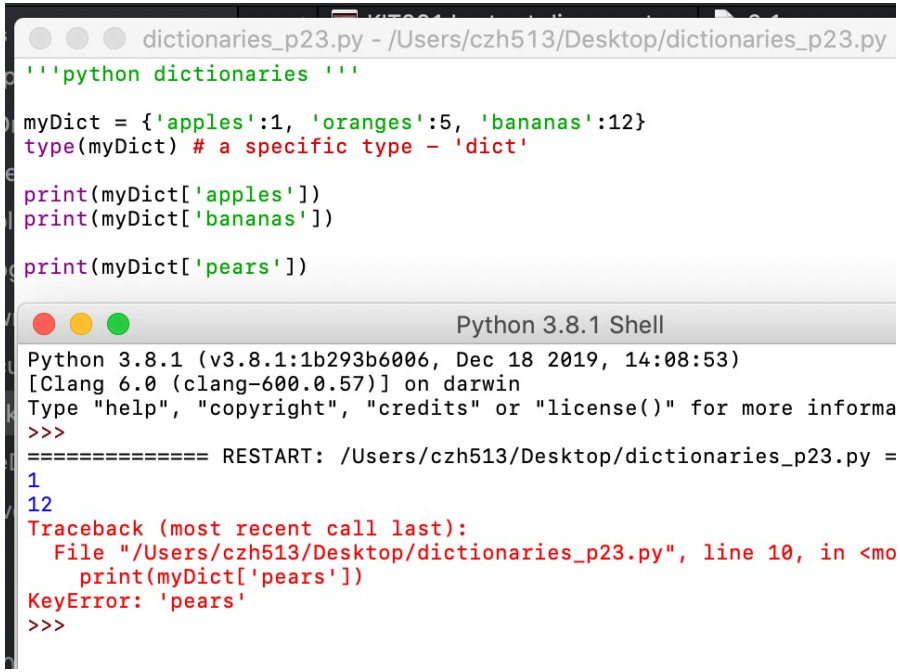
- To retrieve an item from a dictionary,
  - use the **[]** notation (like lists), but you must specify a **key** value.

E.g.

```
print(myDict['apples'])
print(myDict['bananas'])
```

Output:

```
1
12
```



- If you specify a key value that **doesn't exist**, you will get an error:

```
print(myDict['pears'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'pears'
```

Python Dictionaries

Creating an empty dictionary – using curly brackets `{}`:

```
myDict = {}
```

Adding items after a dictionary is created:

```
myDict['pears'] = "who eats pears?"
```

Removing all items from a dictionary:

```
myDict.clear()
```

Removing an item from a dictionary:

```
value = myDict.pop('apples')
```

Python Dictionaries

Iterating (looping) through values in a dictionary :

- Get a list of keys using the **keys** method: `myDict.keys()`
- You can also get a list of **values** with `myDict.values()`

Then you can iterate with each key to lookup a value, eg in a for loop:

```
myDict = {'apples': 1, 'oranges': 5, 'bananas': 12, 'pears': 'yuk, who\
eats  pears?'}

for aKey in myDict.keys():
   print(myDict[aKey])
```

Output:

```
1
5
12
yuk, who eats pears?
```

```
# Adding items:
myDict['pears'] = "who eats pears?"
print(myDict)

#get a list of keys
print(myDict.keys())
#get a list of values
print(myDict.values())

# iterate with each key to lookup a value
for aKey in myDict.keys():
   print(myDict[aKey])
```

Ln: 24 Co

```
{'apples': 1, 'oranges': 5, 'bananas': 12, 'pears': 'who eats pears?'}
dict_keys(['apples', 'oranges', 'bananas', 'pears'])
dict_values([1, 5, 12, 'who eats pears?'])
1
5
12
who eats pears?
```

Python Dictionaries

Dictionaries - A shopping cart example:

```
'''Dictionaries  - A shopping cart example:
'''
item = input("Enter your cart item, 'quit' to exit: ")
cart = {}
while item != "quit":
    count = int(input("How many of this item? "))
    cart[item] = count
    item = input("Enter your cart item, 'quit' to exit: ")

print("You ordered the following things:")
for anItem in cart.keys():
    print("Item: %-10s"%anItem, "number: %-4d"%cart[anItem])
```

Creating an empty dictionary

keys

values

```
dir examples/shoppingcart_p26.py
Enter your cart item, 'quit' to exit: apples
How many of this item? 5
Enter your cart item, 'quit' to exit: pears
How many of this item? 0
Enter your cart item, 'quit' to exit: banana
How many of this item? 134
Enter your cart item, 'quit' to exit: quit
You ordered the following things:
Item: apples     number: 5
Item: pears      number: 0
Item: banana     number: 134
>>>
```

Portfolio task help

- **Portfolio tasks** - By now you should have tried to complete all tasks up to and including 5.2PP, and started on 6.1PP and 7.1PP
- The rest of this lecture will give hints on tasks up to and including 5.2PP to help you catch up

Show only tasks required to reach: PP CR DN HD

Task colours: PP task CR task DN task HD task

	Task	1	2	3	4	5	6	7	8	9	10	11	12	13
1.1PP	Getting Prepared	1.1PP												
1.2PP	Hello World	1.2PP												
3.1PP	Simple Calculation Expression			3.1PP										
3.2PP	User Input			3.2PP										
4.1PP	Decision Making				4.1PP									
4.2PP	Decision Making with Ranges				4.2PP									
5.1PP	Looping (for)					5.1PP								
5.2PP	Looping (while)					5.2PP								
6.1PP	Loops (for and while)						6.1PP							
T1	Test 1							T1						
7.1PP	Functions and parameters							7.1PP						
8.1PP	More Complex Functions								8.1PP					
9.1CR	Lists									9.1CR				
9.2CR	List Operations									9.2CR				
9.3CR	Error Identification									9.3CR				
10.1CR	Code Completion										10.1CR			
10.2CR	Class Definition										10.2CR			
10.3DN	Class Definition and Usage										10.3DN			
T2	Test 2											T2		
11.1DN	GUI statements											11.1DN		
11.2DN	GUI											11.2DN		
11.3HD	Complex Class Definition and Usage											11.3HD		
12.1PP	Learning Reflection Report												12.1PP	
13.1HD	HD+ Task (test 3)													13.1HD

3.1PP Task:

A car is traveling at a constant speed of **50 kilometres per hour**. Write a program that displays the following:

It will take the car 1.0 hour(s) to travel 50 kilometres

It will take the car 2.0 hour(s) to travel 100 kilometres

It will take the car 3.0 hour(s) to travel 150 kilometres

Hints:

- Speed is constant, so variable should be (e.g.) **SPEED**
- Time = distance / SPEED
- Use three **print** statements – the only things that differ are **distance**, and **time**, so set a new distance (**distance = …**) each time and work out the time based on the SPEED and before printing

```
# 3.1PP only for hint

SPEED = 50

distance = 50
time = distance / SPEED

print()
```

3.2PP

3.2PP Task:

Write a program that converts Fahrenheit temperatures (F) to Celsius temperatures (C) as well as Kelvin temperatures (K).

Ask the user to enter a temperature in Fahrenheit.

The formulas are as follows:

$F = (9 \div 5) \times C + 32$

$K = C + 273.15$

Hints:

- Define three variables `fahrenheit`, `celcius` and `kelvin`
- Consider how to ask user for input – most likely they may enter a decimal number so you need to convert the string `input` gives you to a `float`
- Use the **formulas** to calculate the input fahrenheit temperature to celcius and kelvin and then use the variables in print output
  - C = (F - 32) x 5/9

```
# 3.2PP - only for hit

#variables input
degF = float(input("Please enter temperature in Fahrenheit: "))

degC = (degF - 32) * 5/9
degK = (degC + 273.15)

#output to user
print()
```

4.1PP

4.1PP Task: Scientists measure an object's mass in kilograms and its corresponding weight on Earth in Newtons.

If you know the of weight an object has in Newtons, you can calculate its mass in kilograms by using the following formula:

mass = weight / 9.8 (where 9.8 is the acceleration a mass feels due to the force of gravity)

GRAVITY

input("…")

weight

Write a Python program that **asks the user** to enter an object's weight in Newtons, and then calculate and display its mass in kilograms.

mass

If the object's calculated mass is more than 500 kilograms, display a message indicating that it is **too heavy**. If the object's mass is less than 100 kilograms, display a message indicating that it is **too light**.

Hints:

- Use an `if` statement to compare the `mass` to **500** and another `if` statement to compare `mass` to **100**
- `mass = weight / GRAVITY`

```
# 4.1pp - only for hint


GRAVITY = 9.8

weight = float(input("Enter weight in Newtons: "))

mass = weight / GRAVITY
print(mass)

#decision
if mass > 500:
    print(1)
elif mass < 100:
    print(2)
```

4.2PP

4.2PP Task:

A restaurant has controversially decided to charge customers a surcharge multiplier on their meal based on their age:

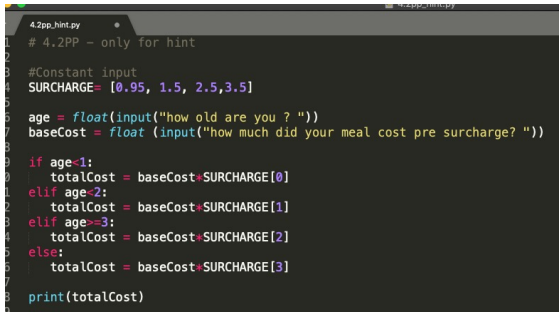
Your program should ask the user their age and then their pre-surcharge meal cost. It should then display the correct total meal cost for the user after the surcharge multiplier has been applied.

Age	Meal Surcharge
10 or less	0.95
More than 10 but less than or equal to 20	1.5
More than 20 but less than or equal to 40	2.5
More than 40	3.5

Example: a user is 27, meal cost $10.50 x meal surcharge $2.5 = total meal cost $26.25

Hints:

- The table gives you the values to use in the if statement comparisons; the logic is much simpler if you use `if .. elif .. elif .. else` statements
- You need to work out which comparison operators to use – less-than (<), less-than-or-equal (<=), greater-than (>) or greater-than-or-equal (>=)
- The surcharge amounts should be defined as constants e.g. SURCHARGE1 = 0.95
- Initially you should ask the user for the base meal cost, and then based on the value apply the appropriate surcharge cost.



```
4.2pp_hint.py
# 4.2PP - only for hint

#Constant input
SURCHARGE= [0.95, 1.5, 2.5,3.5]

age = float(input("how old are you ? "))
baseCost = float (input("how much did your meal cost pre surcharge? "))

if age<1:
    totalCost = baseCost*SURCHARGE[0]
elif age<2:
    totalCost = baseCost*SURCHARGE[1]
elif age<=3:
    totalCost = baseCost*SURCHARGE[2]
else:
    totalCost = baseCost*SURCHARGE[3]

print(totalCost)
```

5.1PP

5.1PP Task:

input("…")

- Create a program that tourists can use to determine how many steps they can climb in a certain amount of time

stepRate

- Write a Python program that **asks the user** to enter their stair-climbing rate (in steps per minute), followed by a time duration (in minutes). time
- The program will then display a heading, followed by several rows of data (using a **for** loop), with each row (suitably aligned to the heading) displaying the elapsed minute (starting at 1), followed by the cumulative number of steps climbed. The last row *time* should match the duration entered by the user.
- Your program should finally indicate which minute (if at all) they would reach the **lookout situated 1000 steps** above that starting point. If the lookout was not reached, a suitable message should be displayed.

Example output:

```
Please enter your stair step rate in steps per minute: 250
Please enter the length of time in minutes to display: 6

Time Step total
----------------
  1    250
  2    500
  3    750
  4   1000
  5   1250
  6   1500
You reached the lookout at 1000 steps in 4.0 minutes
```

A second example:

```
Please enter your stair step rate in steps per minute: 100
Please enter the length of time in minutes to display: 7

Time Step total
----------------
  1    100
  2    200
  3    300
  4    400
  5    500
  6    600
  7    700
You did not reach the lookout at 1000 steps!
```

Hints:

- You need to use input to ask the user for two values (stepRate and time)
- The **height** is a CONSTANT.. so… GOALSTEPS = 1000.
- The **for** loop displays a minute value each line, up to the total time value (use the range() function). The number of steps to display in the current minute is related to the stepRate and the minute.
- You can work out the time it would take to reach the top based on the **height** and the stepRate (set: timereach = GOALSTEPS / stepRate), and then compare this to the time value entered via the input to see if they reached the top (use if …else…).

```python
# 5.1PP - only for hint

#CONSTANT and Variables
GOALSTEPS = 1000

time = int(input("Enter time in min please: "))
stepRate = int(input('Enter step rate per min please: '))

print()
print('Time  Step total')

for minute in range(1,time+1):
   print()

timereach = GOALSTEPS / stepRate

if timereach >= time:
   print()
else:
   print()
```

5.2PP Task:

Write a Python program that first tells the user what it does, and then **asks the user** to enter a series of **integers** (positive or negative). `numberInput`

The user should enter **zero** to signal the end of the series.

After all the numbers have been entered, the program should display their **product** and the count of how many numbers were entered.

Example output:

```
I multiply your numbers together.
Enter your next number, 0 to finish: 2
Enter your next number, 0 to finish: 3
Enter your next number, 0 to finish: 4
Enter your next number, 0 to finish: 0

The product of your numbers is 24
You entered 3 numbers
```

A second example:

```
I multiply your numbers together.
Enter your next number, 0 to finish: 2
Enter your next number, 0 to finish: -2
Enter your next number, 0 to finish: 2
Enter your next number, 0 to finish: 5
Enter your next number, 0 to finish: 0

The product of your numbers is -40
You entered 4 numbers
```

Hints:

- **Product** is to multiple the entered numbers (number 1 x number 2 x number 3 …).
- This uses a **while** loop – you need to work out the appropriate condition for when the loops **stops** -> `[Out of loop: numberInput == 0]`
- You will need to **store** the current `product,` and `count` of how many numbers are entered. You'll also need to store the current `number` entered via the input function.
- Before the loop, think about what the initial `count` is.. and what the initial `product` is.
- Each time through the loop you should ask for a new `number` to be entered ("remember to call the input again"), then update the `product` and `count` variable appropriately.
- When the loop finishes, print the values of the `count` and `product` variables.

```python
# 5.2PP - only for hint

print("I multiply your numbers together.")
numberInput = int(input("Enter your number, 0 to finish:"))

product=1 # right?
count=0 # right?

while numberInput!=0: # right?

    count=count+1
    product=product*numberInput
    numberInput=int(input("Enter your number,0 to finish: "))
    # remember to call the input again

print(product)
print(count)
```

Questions?