CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

http://www.cit.ie

Lists

Lists – Todays Topics

- Using a method to total the numbers in a list
- Writing a method to calculate an average
- Returning a new list from a method
- Processing elements in lists

)

Functions and Lists

- In order to make our code more readable and reusable we will need to use functions to carry out specific tasks.
- A list can be passed in as an argument to a function. The list can be used in the function and changed by the function
- A list can be created in a function and returned from the function using the return mechanism

Passing a list as a function argument

01_list_show_names.py

- Write a function that prints out the contents of a list of strings. Pass the list as an argument to the function

```
def print_list(names):
    for item in names:
        print(item)

def main():
    counties = ["Kerry","Cork","Waterford", "Limerick", "Tipp"]
    print_list(counties)

main()
```

```
Kerry
Cork
Waterford
Limerick
Tipp
```

Passing a list as a function argument

01_list_show_names.py

- Two ways of printing a list of information while also keeping a counter at the same time

```
def print_list(names):
    for i in range(len(names)):
        print(str(i) + ".  "+ names[i])
    print()
    for i, item in enumerate(names):
        print(str(i) + ".  " + item)

def main():
    counties = ["Kerry","Cork","Waterford", "Limerick", "Tipp"]
    print_list(counties)


main()
```

This is a very concise way to iterate over a list and keep track of its position in the list

```
0.  Kerry
1.  Cork
2.  Waterford
3.  Limerick
4.  Tipp
```

```
0.  Kerry
1.  Cork
2.  Waterford
3.  Limerick
4.  Tipp
```

Calculating a total of a list of numbers

02_list_calculate_total.py

- We can process the information in a list and return some new information
- This method takes a list of numbers and then calculates and returns the total

```
def get_total( numbers_list):
    total = 0
    for i in numbers_list:
        total = total + i
    return total

ages  = [ 10, 22, 13, 14, 19]
total = get_total(ages)
print("The total is " +str(total))
```

Note: the built –in *sum* method does the same as our method getTotal

```
total = sum(ages)
```

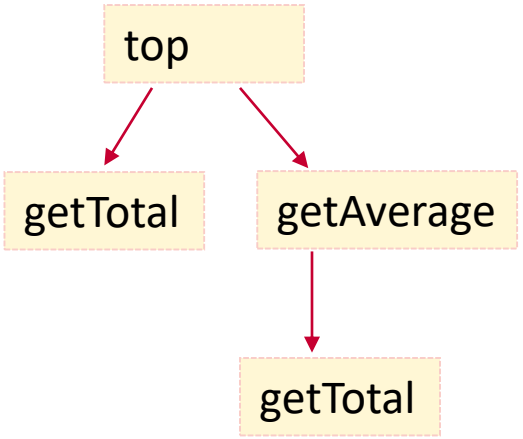
```
The total is 78
```

Calculating an average of a list of numbers

02_list_calculate_total.py

- Write a function to calculate the average of a list of numbers and return the average

```
def get_average(numbers_list ):
    total = get_total(numbers_list)
    amount = len(numbers_list)
    average = 0;
    if (amount != 0 ):
        average = total/amount
    return average

ages  = [ 10, 22, 13, 14, 19]
total = get_total (ages)
avg = get_average (ages)
print("The total is " +str(total))
print("The average is " +str(avg))
```




```
The total is 78
The average is 15.6
```

A function that returns a list of numbers

03_list_get_numbers.py

- Write a function to get a list of numbers from the user. The amount of numbers required is specified as an input parameter

```
def get_numbers(amount):
    numbers = [0] * amount
    i = 0
    while ( i < amount):
        number  = int(input("Please enter number " + str(i+1)))
        numbers[i]  = number
        i = i + 1
    return numbers

def main():
        amount_required = int(input("How many numbers do you want?"
        numbers_list = get_numbers(amount_required)
        print(numbers_list)

main()
```

```
How many numbers do you want to add to
the list?3
Please enter number 15
Please enter number 29
Please enter number 312
[5, 9, 12]
```

Write a function to return a list of strings

04_list_get_names.py

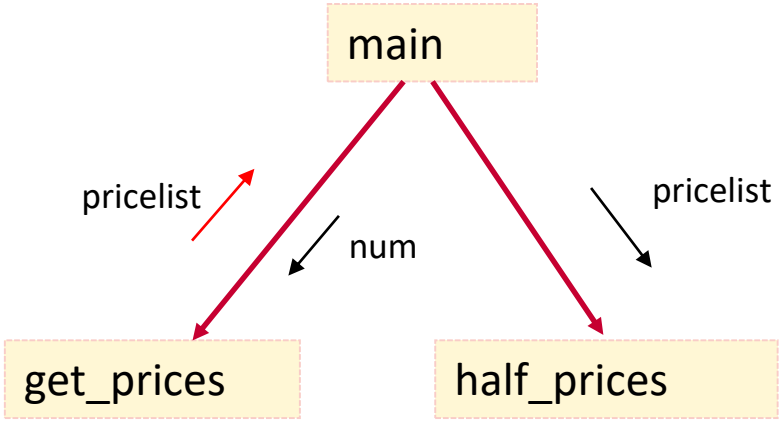
- This function gets an unspecified number of names from the user

```
def getNames():
    names = []
    add_another_name = True
    while (add_another_name):
        name  = input("Please enter a name: ")
        names.append(name)
        more = input("Would you like to enter another name y/n")
        if more != 'y' and more != 'Y':
            add_another_name = False
    return names
def main():
    names_list = getNames()
    print(names_list)


main()
```

```
Please enter a name: Mary
Would you like to enter another name y/n:y
Please enter a name: Joe
Would you like to enter another name y/n:y
Please enter a name: Pat
Would you like to enter another name y/n:n
['Mary', 'Joe', 'Pat']
```

Problem: update a list of prices

- Write a function that takes the prices for three items from the user and stores them in a list. Write a second function that reduces all the prices by half. Print out the updated prices list.



We will look at the implementation of half_prices

A function to modify a list of prices

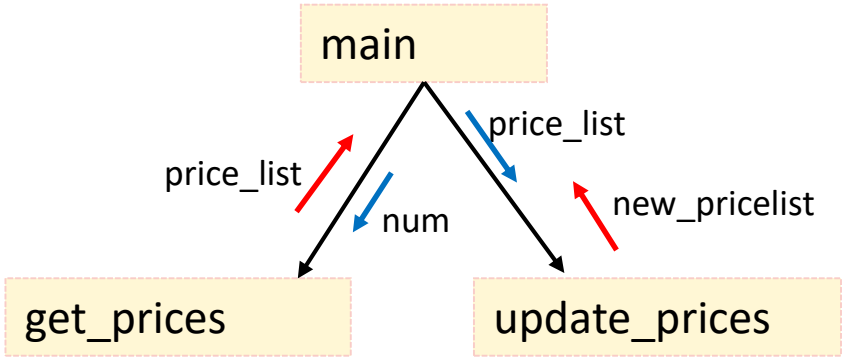
05_list_update_prices1.py

```
def  half_prices(list):
    for i in range(len(list)):
        list[i] = list[i]/2

def main():
    # TODO - get list from the user
    list = [ 40.00, 30.50, 1.10, 99.00]
    print("Initial Price list")
    print(list)
    print("Updated Price list")
    half_prices(list)
    print(list)

main()
```

```
Initial Price list
[40.0, 30.5, 1.1, 99.0]
Updated Price list
[20.0, 15.25, 0.55, 49.5]
```

A function that returns updated prices in a new list

- Write a function that takes a list of prices as a parameter and returns a new list containing the reduced prices.
- The original list passed to the method is unchanged. This means the function is a **pure function** as there are no unexpected side effects on the input parameters.



A function that returns updated prices in a new list

06_list_update_prices2.py

```
def  half_prices(list):
    new_prices = []
    for item in list:
        new_prices.append(item/2)
    return (new_prices)

def main():
    list = [ 40.00, 30.50, 1.10, 99.00]
    print(list)
    print("Create a new list to store the new prices")
    new_list = half_prices(list)
    print(list)
    print(new_list)
```

```
[40.0, 30.5, 1.1, 99.0]
create a new list to store the new prices
[40.0, 30.5, 1.1, 99.0]
[20.0, 15.25, 0.55, 49.5]
```

List Comprehension: to generate for a new list from an existing list

06_list_update_prices2.py

- In the last example we created the `new_prices` list based on information in the original `list`.
- This uses a for loop to take the items from `list`, divide each item by 2 and then add it to the `new_prices` list

```
def  half_prices(list):
    new_prices = []
    for item in list:
        new_prices.append(item/2)
```

- A short cut to do the same thing is to use a technique called **list comprehension**. This is how we would use it in this example.

```
def  half_prices(list):
    # shorter way to do the same thing as for loop above
    new_prices = [item * .5 for item in list ]
    return new_prices
```

Strings

- A string is a sequence. The slicing techniques we saw with lists apply to strings

```
word1 = "Hawaii"
startLetters = word1[0:2]
endLetter = word1[-1]
print(word1[0])
```

- Iterating over strings is similar to iterating over lists

```
for x in word1:
    print(x)
for i,x in enumerate(word1):
    print(i, x)
```

```
H
a
w
a
i
i
```

```
0 H
1 a
2 w
3 a
4 i
5 i
```

- The membership operator **in** can be used

```
if 'H' in word1 :
        print ("contains H")
```

- However strings are **immutable** so **an existing string may not be changed**. The following is illegal

```
word1 = "Hawaii"
word1[0] = 'K  #illegal
```

String methods

- Some of the methods used with lists can be used with Strings
  - count the numbers of i's in Hawaii

```
number = word1.count('i')
print("There are "+ str(number) + 'i s in ' + word1)
```

  - find the position/index of a letter in a String

```
pos = word1.index('w')
print("w is at position " + str(pos)  + " in " + word1)
```

String specific operations

- There are some methods that are specific to strings
  - The `split` method converts a sentence to a list of strings

"Hello There" -> ["Hello", "There"]

  - The `join` method converts from a list to a string/sentence

["Hello", "There"] -> "Hello There"

  - `find` method finds a substring in a string

  - `replace` method replaces a substring with another string and returns a new string. The original string is not modified

String Specific methods

08_String_specific_methods.py

```
lyrics = "Fly me to the moon"
pos = lyrics.find("me")
print("me" + " is at position/index " + str(pos))

new_lyrics = lyrics.replace("Fly", "Take")
print(lyrics)
print(new_lyrics)

print("convert the string to a list:")
list1 = lyrics.split()
print(list1)

print("Create a string from a list of words")
list2 = [ "this", "is", "a", "list"]
word = ''.join(list2)
print(word)
```

```
me is at position/index 4
Fly me to the moon
Take me to the moon
convert the string to a list:
['Fly', 'me', 'to', 'the', 'moon']
Create a string from a list of words
thisisalist
```

Problem: Megan's Barista Coffee Shop

Write a program that allows Megan enter the number of employees in a coffee shop and then allows her record the hours worked by each barista. Ask Megan for the hourly pay rate. The program should then:

- Print out the amount of pay owned to each barista.
- Print out the total amount Megan needs to pay the wages.
- Print the average no. of hours worked by each employee.
- Identify the least number of hours worked by any employee.
- Allow for a bonus payment of 20 Euros for all employees who worked more than 30 hours. Find out how many are eligible for the bonus and print out how much extra money is needed

Megan's Barista : Identify the variables

Variables/Constant	Types for inputs	User input or Calculation ?
number_of_employees	int	From user
hours_worked	List of ints	From user
hourly_pay_rate	float	From user
total_pay_bill		Calculate from hours_worked and hourly_pay_rate
average hours		Calculate from hours_worked and number_of_employees
least_hours_worked		Find min value in hours_worked list
number_eligible_bonus		Go thru hours_worked list
bonus_money_needed		Calculate from number_eligible_bonus and BONUS_PAYMENT
BONUS_PAYMENT	20	
HOURS_REQUIRED	30	

Megans Barista: Identify Functions to write

- Identify the function inputs and outputs
  - **get_hours_worked**
    - input: number_of_employees
    - output: hours_worked
  - **get_hourly_pay_rate**
    - output: hourly_pay_rate
  - **show_pay_per_employee**
    - input: hours_worked
    - output: none
  - **get_total_pay**
    - input: hours_worked, hourly_pay_rate
    - output: cost_of_pay_bill
  - **get_avg_hours**

Megans Barrista: Identify Functions

- **get_min_hours**
  - input: hours worked
  - output: min
- **get_bonus_money_needed**
  - input: hours worked
  - output: bonus_money_needed