P01: INTRODUCTION TO PYTHON

INTRODUCTION

The programming in this course will be written in Python, an interpreted, object-oriented language that shares some features with both Java and Scheme. This tutorial will walk through the primary syntactic constructions in Python, using short examples.

Arithmetic operators

The Python interpreter can be used to evaluate expressions, for example simple arithmetic expressions.

In [1]:

```
# This is a comment
1 + 1   # addition operation
```

Out[1]:

```
2
```

In [4]:

```
g = 4
h = 3.2
j = 'string' #not char in py
print( g, h, j)
```

```
4 3.2 string
```

In [6]:

```
j = 2
print(j)
```

```
2
```

In [2]:

```
2 * 3   # multiplication operaiton
```

Out[2]:

```
6
```

In [3]:

```
11 / 3  # Normal division
```

Out[3]:

```
3.6666666666666665
```

In [7]:

```
11 // 3 # integer division
```

Out[7]:

```
3
```

In [8]:

```
11 % 3  # modulus operation
```

Out[8]:

```
2
```

In [9]:

```
2 ** 3  # exponent operation
```

Out[9]:

```
8
```

In [10]:

```
s = 12
```

Boolean operators also exist in Python to manipulate the primitive True and False values.

In [11]:

```
1 == 0
```

Out[11]:

```
False
```

In [12]:

```
not (1 == 0)
```

Out[12]:

```
True
```

In [13]:

```
(2 == 2) and (2 == 3)
```

Out[13]:

```
False
```

In [14]:

```
(2 == 2) or (2 == 3)
```

Out[14]:

```
True
```

Variables

We can also store expressions into variables.

The syntax is different from C++. Here are some important rules about declaring and using varaibles in Python:

1. You *do not have to perform variable declaration*
2. The variable *must be created before it can be used*
3. The *type will be automatically determined* by the interpreter based on the value assigned

In [15]:

```
s = 'hello world'   # defining a string variable. The type of s is assigned automatically b
print(s)
print(type(s))
```

```
hello world
<class 'str'>
```

In the above code, we use `type` to display the type of `s`. This is because different from C++, the type of a variable can be changed during execution.

In [16]:

```
fval = 7.3   # this will create a floating point variable
print(fval)
print(type(fval))
```

```
7.3
<class 'float'>
```

In [17]:

```
ival = 12    # this will create an integer variable
print(ival)
print(type(ival))
```

```
12
<class 'int'>
```

In [18]:

```
fval += 2.5  # operate on fval
print(fval)  # you can print the content of a variable  using print command
```

```
9.8
```

In [19]:

```
print(type(fval))
```

```
<class 'float'>
```

In [20]:

```
ival += 3
ival          # you can display the value of the LAST instruction in the cell by typing the
```

Out[20]:

```
15
```

Remember that you cannot use a variable that you have not created. The following command will generate an error because `var` has not been created

In [21]:

```
print(var)   # we have not defined a variable called var! This will generate an error!
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-0b65bda2f1d6> in <module>
----> 1 print(var)   # we have not defined a variable called var! This will
 generate an error!

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

The Math Library

The Math Unit enables you to use mathematical functions such as π (pi) and sine, cosine and tangent.

In [22]:

```
import math
```

In [23]:

```
print ('The sin of 90 degrees is', math.sin(math.pi/2))
```

```
The sin of 90 degrees is 1.0
```

In [24]:

```
print ('log of 100 is ', math.log(100))
```

```
log of 100 is  4.605170185988092
```

Strings

Like Java, Python has a built in string type.

- You can use either a single quote (') or double quote (") to define a string.
- The + operator is overloaded to do string concatenation on string values.

In [25]:

```
'artificial' + "intelligence"
```

Out[25]:

```
'artificialintelligence'
```

In [26]:

```
s = 'Hello'
```

In [27]:

```
s = s + 'world'
```

In [28]:

```
s             # displaying the string value
```

Out[28]:

```
'Helloworld'
```

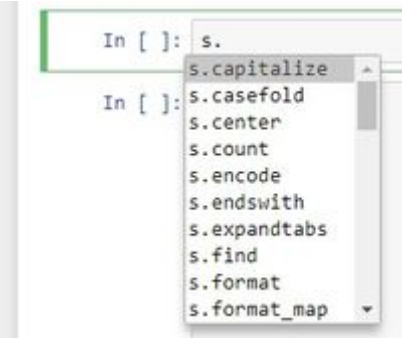
In [29]:

```
print(s)      # printing the string
```

```
Helloworld
```

There are many built-in function that comes with a string variable.

Tips: *To see the list of functions provided by a library or string, type the name and then press TAB. For example, if you type `s.<TAB>` to see the list of built-in functions:



The following coded show how to use several built-in functions to manipulate a string.

In [31]:

```
s = 'HELP'
s.lower()
```

Out[31]:

```
'help'
```

In [32]:

```
'artificial'.upper()
```

Out[32]:

```
'ARTIFICIAL'
```

To get the length of a string, you can use the `len` command

In [33]:

```
len('Help')
```

Out[33]:

```
4
```

Getting help

You can also use the command `help` to get help on certain command.

In [34]:

```
help(s.find)
```

```
Help on built-in function find:

find(...) method of builtins.str instance
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

INPUT & OUTPUT

Reading input from keyboard

To get input from user, use the `input()` function.

In [35]:

```
name = input('Enter your name: ')
print(name)
```

```
Enter your name: Kong
Kong
```

The function returns the user's input as a string. You will need to convert the string to a numeric number using the `int` and `float` functions.

In [36]:

```
ival = int(input('Enter an integer: '))
print(ival+13)
```

```
Enter an integer: 9
22
```

In [37]:

```
fval = float(input('Enter a float: '))
print (fval + 0.5)
```

```
Enter a float: 2.1
2.6
```

Printing to the screen

The print function by default prints a space character between items and a newline at the end of the sentence

In [38]:

```
a = 3
print('There are', a, 'people in the meeting')
```

```
There are 3 people in the meeting
```

You can *format* the output using the `str.format()` method where we use the curly braces `{}` as the placeholders.

In the following example, the first bracket `{}` would be replaced by `x` and the second by `y`.

In [39]:

```
x = 5; y = 10
print('The value of x is {} and y is {}'.format(x,y))
```

```
The value of x is 5 and y is 10
```

We can also specify *the order* in which it is printed by using numbers (tuple index)

In the following example, the placeholder `{0}` would be replaced by the first parameter `bread` regardless of its position in the `print` statement. Similarly for the placeholder `{1}`.

In [40]:

```
print('I love {0} and {1}'.format('bread', 'butter'))
print('I love {1} and {0}'.format('bread', 'butter'))  # the order
```

```
I love bread and butter
I love butter and bread
```

Formatting string

Formatting integer values:

The placeholder `{:6d}` has the following meaning:

- `d` specifies that the placedholder is to be replaced by an *integer*.
- `6` specifies that a total of 6 characters would be allocated to print out the integer.

In [41]:

```
print('Integer value is {:d}.'.format(23)) #d mean integer
print('Integer value is {:6d}.'.format(23))#6 places
```

```
Integer value is 23.
Integer value is     23.
```

Formatting float values:

The placedholder `{:06f}` has the following meaning:

- `f` specifies that the placedholder is to be replaced by an *real number*.
- `6` specifies that a total of 6 characters would be allocated to print out the number.
- `.2` displays the real number up to two decimal places
- `0` specifies that if the number of characters is less than number of allocated characters (6 in our case), the empty spaces should be replaced with 0.

In [42]:

```
print('The float value is {:f}'.format(3.141592653589793))
print('The float value is {:06.2f}'.format(3.141592653589793))
print('The float value is {:6.2f}'.format(3.141592653589793))
```

```
The float value is 3.141593
The float value is 003.14
The float value is   3.14
```

Exercise 1

Q1. You want to compute the future value `F` of our saving account. Assuming that you want to deposit `P` amount of money into your saving account at present then the formula to compute `P` in `y` years time for a fixed interest rate of `r` is given by the formula:

$$P = \frac{F}{(1+r)^y}$$

How much do you need to deposit today to have RM10,000 in the bank in 10 years time if the interest rate r = 0.05.

You must

- Use variables to capture the value of future value (F), interest rate (r) and years (y).
- Display the answer up to two decimal places.

[Ans = 6139.13]

In [48]:

```
F = 100000
r = 0.05
y = 10
P = F/(1+r)**y
print('P = {:.2f}'.format(P))
```

```
P = 61391.33
```

Q2. Prompt the user for a temperature in celcius. Then, convert celcius temperatures to fahrenheit temperatures using the following formula:

$$F = \frac{9}{5}C + 32$$

and then display the result to the user. [12 C is equivalent to 53.6 F]

In [52]:

```
C = int(input('Please input temperature in celcius: '))
F = 9/5*C + 32
print('{:d} C is equivalent to {:.1f} F'.format(C,F))
```

```
Please input temperature in celcius: 12
12 C is equivalent to 53.6 F
```

Q3. Write a program that will ask the user for two numbers a then divide one by the other. The number of times one goes into another and the remainder should be displayed.

Example answer:

```
Please enter first number: 5
Pleae enter second number: 3
5/3 = 1 remainder 2
```

In [53]:

```
val1 = int(input('Please enter first number: '))
val2 = int(input('Please enter second number: '))
divide = val1//val2
mod = val1%val2
print("{}/{} = {} remainder {} ".format(val1,val2,divide,mod))
```

```
Please enter first number: 5
Please enter second number: 3
5/3 = 1 remainder 2
```

Q4. Define a string and initialize it with '*The lyrics is not bad at all, not bad at all.*'. Write a Python program to find the first appearance of the substring '*not bad at all*' from a given string. Replace that term with '*good*'. Hints: Use the built-in function of string.

Answer:

```
The lyrics is good, not bad at all.
```

In [54]:

```
sentence = "The lyrics is not bad at all, not bad at all."
sentence.replace('not bad at all','good',1)# your code here
```

Out[54]:

```
'The lyrics is good, not bad at all.'
```

FUNCTIONS

Creating basic functions

The following code shows how to define a function. Note that the parameters, documentation string and the return statements are optional depending on the application.

USE IDENTATION TO DEFINE A FUNCTION. Unlike many other languages, Python defines its course structure using indentation (`TAB`) rather than curly brackets (`{ }`). To define a function, all codes belonging to the function must be at least **one tab** away from the function header. For logical or loop statements, they can be more than than one tab.

This is how to define a function. Note that all the codes belonging to the function is at least one TAB away from the `def` statement

In [55]:

```
# Correct way of defining a function
def square(num):
    "Returns the square of a number"
    result = num**2
    return result
```

Now that we have defined our function `square` , we can make use of the function to square some numbers

In [56]:

```
x = square(2)
print(x)
```

```
4
```

The following shows the wrong way to define a function. Python does not use curly bracket to define the code block for a function

In [57]:

```
# this will not work.
def square(num):
{ result = num**2
return result
}
```

```
  File "<ipython-input-57-a77f327d3812>", line 3
    { result = num**2
    ^
IndentationError: expected an indented block
```

This code below will NOT work too. The code `return result` does not belong to the function square because not one tab away from `def`

In [58]:

```
# This will NOT work too. The code `return result` does not belong to the function square
# because not one tab away from `def`

def square(num):
    result = num**2
return result
```

```
  File "<ipython-input-58-86bddabf8776>", line 6
    return result
                ^
SyntaxError: 'return' outside function
```

Returning more than one variable

Different from most language, Python can return more than one variables

In [59]:

```
def square2and3(num):
    "Returns the square of a number"
    result1 = num**2
    result2 = num**3

    return result1, result2
```

In [60]:

```
sq2, sq3 = square2and3 (2)
print(sq2, sq3)
```

```
4 8
```

Passing arguments to a function

By default, **primitive variables** such as integer, double and float as well as **strings** are **passed by value**.

For example, for the following code, any changes to parameters `intval` and `strval` in function `func` that are passed from main will not affect the original variables in main.

In [61]:

```
def func (intval, strval):
    intval = 100
    strval = "Modified string value"
    print (">>> func: numval = {}, strval = {}".format(intval, strval))
```

In [62]:

```
intval  = 0
strval = "Original string value"

print ("main (before calling func): numval = {}, strval = {}".format(intval, strval))
func (intval, strval)
print ("main (after calling func): numval = {}, strval = {}".format(intval, strval))
```

```
main (before calling func): numval = 0, strval = Original string value
>>> func: numval = 100, strval = Modified string value
main (after calling func): numval = 0, strval = Original string value
```

Setting default arguments

In [63]:

```
def printinfo(name, age = 35):
    "This prints a passed info into this function"
    print ("Name: ", name)
    print ("Age: ", age)
```

In [64]:

```
printinfo (name = "Suzuki")
printinfo (name = "Michiko", age = 12)
```

```
Name:  Suzuki
Age:  35
Name:  Michiko
Age:  12
```

Exercise 2

Q1. Write a program that prints out the area and circumference of a circle. Create two functions `area` and `circumference` that returns the area and circumference of a circle, respectively.

Ans:

```
r = 5: area = 78.54, circumference = 31.42
```

In [82]:

```
def area(r):
    return math.pi * r ** 2
def circumference(r):
    return 2 * math.pi * r
def aNc(r):
    return area(r),circumference(r)
r = int(input("r = "))
a,c = aNc(r)
print('r = {}: area = {:.2f} circumference = {:.2f}'.format(r,area(r),circumference(r)))
print( a , c)
```

```
r = 5
r = 5: area = 78.54 circumference = 31.42
78.53981633974483 31.41592653589793
```

Q2. Write a function that returns both the quotient and remainder of a division. Do not use two functions and do not print the results in the function.

Ans:

```
    5/2 = 2 remainder 1
```

In [66]:

```
def division(a,b):
    return(a//b,a%b)
a = 5
b = 2
c,d =division(a,b)
print("{}/{} = {} remainder {}".format(a,b,c,d))
```

```
5/2 = 2 remainder 1
```

CONDITIONAL STATEMENT AND LOOP

Conditional Statements

The conditional operators for Python is `if` and `elif` . There is NO `switch-case` operator in Python.

PYTHON USES INDENTATION TO DEFINE THE SCOPE: In C++, curly brackets `{}` is used to define the scope of conditional statements. Differently, Python uses the indentation to define scopes. All statements belonging to a conditional statements must be at least one indent away from a conditional operator such as `if` or `elif` statement belongs that condition.

The following code shows the syntax for implementing conditional statements in Python.

In [67]:

```
# This is what a comment looks like
price = float(input('how much is one apple? '))
print(price)
if price < 2.00:
    print (price, 'is cheap, I should buy more')
elif price < 4.0:
    print (price, 'is not cheap, I should buy less')
else:
    print (price, 'is too expensive, I cannot afford it')
```

```
how much is one apple? 3
3.0
3.0 is not cheap, I should buy less
```

BE CAREFUL OF INDENTATION. Wrong indentation changes the behaviour of your program.

```
    if 0 == 1:
        print ('We are in a world of arithmetic pain')
    print ('Thank you for playing')
```

will output:

```
    Thank you for playing
```

But if we had written the script as

```
    if 0 == 1:
        print ('We are in a world of arithmetic pain')
        print ('Thank you for playing')
```

there would be no output.

The moral of the story: be careful how you indent!

The following shows how to put compound conditions in a conditional statement

In [69]:

```
price = float(input('how much is one apple? '))
if price >= 2 and price <= 5:
    print('price is reasonable')
else:
    print('price is not reasonable')
```

```
how much is one apple? 1
price is not reasonable
```

In [70]:

```
price = float(input('how much is one apple? '))
if price < 2 or price > 5:
    print('price is not reasonable')
else:
    print('price is reasonable')
```

```
how much is one apple? 3
price is reasonable
```

Loops

The operator for performing loops in Python is `for` and `while`. Similarly, indentation is used to define the scope for loops. The following example show how to use the `for` loop to iterate a list of integers and strings

In [71]:

```
for num in [1, 2, 3, 4, 5]:
    print (num)
```

```
1
2
3
4
5
```

In [72]:

```
for name in ['lessie', 'joe', 'bob']:
    print (name)
```

```
lessie
joe
bob
```

In [73]:

```
num = 1
while num <= 5:
    print (num)
    num += 1
```

```
1
2
3
4
5
```

The `range` command

You can use the function `range(end)` to generate an iterable sequence of integers starting from 0 up to not including `end`.

In [74]:

```
for num in range(5):
    print (num)
```

```
0
1
2
3
4
```

You can use the function `range(start, end)` to generate an iterable sequence of integers starting from `start` up to `end-1` .

In [75]:

```
for num in range(3,7):
    print (num)
```

```
3
4
5
6
```

You can use the function `range(start, end, skip)` to generate an iterable sequence of integers starting from start up to end-1 with a resolution of `skip` .

In [76]:

```
for num in range(10,26,5):
    print (num)
```

```
10
15
20
25
```

The `enumerate` command

When looping a sequence, the position index and corresponding value can be retrieved using the `enumerate()` function.

In [77]:

```
for i, v in enumerate(['tic', 'tac', 'toe']):#i start from 0
    print (i, v)
```

```
0 tic
1 tac
2 toe
```

The `zip` command

To loop over two or more sequences at the same time, the entries can be paired using the `zip()` function.

In [78]:

```
questions = ['name', 'quest', 'favorite color']
answers = ['lancelot', 'the holy grail', 'blue']
for q, a in zip(questions, answers):
    print ('What is your {0}?  It is {1}.'.format(q, a))
```

```
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

In [79]:

```
names = ['Krishnan', 'Shafie', 'Kim Loong']
grades = [99, 100, 89]

for n, g in zip(names, grades):
    print ('{0}: {1}.'.format(n, g))
```

```
Krishnan: 99.
Shafie: 100.
Kim Loong: 89.
```

Exercise 3

(Q1). Write a program to ask the user for a floating point. Print out which category the number is in: 'positive', 'negative', or 'zero'.

In [109]:

```
n = float( input('Please enter a floating point number :'))
if n > 0 :
    print('It is positive')
elif n < 0 :
    print('It is negative')
else:
    print('It is zero')
```

```
Please enter a floating point number :0
It is zero
```

(Q2). Write a Python program to find those numbers which are divisible by 7 and multiple of 5, between 1500 and 2700 (both included).

In [115]:

```
for n in range(1500,2701):
    if n % 7 == 0 and n % 5 == 0 :
        #print(n)
        print(n,end =' ')
```

```
1505 1540 1575 1610 1645 1680 1715 1750 1785 1820 1855 1890 1925 1960 1995 2
030 2065 2100 2135 2170 2205 2240 2275 2310 2345 2380 2415 2450 2485 2520 25
55 2590 2625 2660 2695
```

(Q3). Create the `A = [12, 13, 5, 16, 13, 7, 11, 19]` . Compute the average values of the items in the

list.

Ans: 12

In [103]:

```
A = [12, 13, 5, 16, 13, 7, 11, 19]
sum = 0
num = 0
for n in A:
    sum = sum + n
    num = num + 1
print(sum//num)
```

```
12
```

(Q4). Create two arrays as follows:

```
A = [12, 13, 5, 16, 13]
B = [12, 2, 5, 2, 13]
```

Your task is to compare how many of the corresponding items (items at the same position) are equivalent.

Ans: There are 3 similar corresponding items in the two lists.

In [96]:

```
A = [12, 13, 5, 16, 13]
B = [12, 2, 5, 2, 13]
count = 0
for a,b in zip(A,B):
    if a==b:
        count = count + 1
print('There are {} similar corresponding items in the two lists.'.format(count))
```

```
There are 3 similar corresponding items in the two lists.
```

(Q5). Create the array `A = [12, 11, 5, 64, 21, 182, 33]` . Your task is to extract all even numbers and their position in the list. Use `enumerate` to capture the position.

Ans:

```
index 0 : 12
index 3 : 64
index 5 : 182
```

In [98]:

```
A = [12, 11, 5, 64, 21, 182, 33]
for i , num in enumerate(A):
    if num%2==0:
        print('index {} : {}'.format(i,num))
```

```
index 0 : 12
index 3 : 64
index 5 : 182
```

BUILT-IN DATA STRUCTURES

Python comes equipped with some useful built-in data structures, broadly similar to Java's collections package. This includes

- list
- tuple
- set
- dictionary

List

First, we look at lists. Lists store a sequence of *mutable* items. Mutable items are items that can be changed later. They are not constants.

Create a new list:

In [99]:

```
fruits = ['apple','orange','pear']   # a list of strings
print (fruits)

numbers = [11,12,13,14]              # a list of integers
print(numbers)

isHandsome = [True, False, True]       # a list of booleans
print(isHandsome)

rojak = [11, 'hello', True, [1,2,3], ['one', 2, False]]     # a list of different types of
print(rojak)
```

```
['apple', 'orange', 'pear']
[11, 12, 13, 14]
[True, False, True]
[11, 'hello', True, [1, 2, 3], ['one', 2, False]]
```

Get the length of a list:

In [100]:

```
len(numbers)
```

Out[100]:

4

Iterate all items in an array:

In [104]:

```
numbers = [11,12,13,14]
for x in numbers:
    print (x)
```

```
11
12
13
14
```

Unpack the values in a list into individual variables

In [105]:

```
a, b, c = fruits
print(a)
print(b)
print(c)
```

```
apple
orange
pear
```

Concatenate multiple lists

Use the + operator to do list concatenation:

In [106]:

```
fruits = ['apple','orange','pear']
print(fruits)

otherFruits = ['kiwi','strawberry']
print(otherFruits)

allFruits = fruits + otherFruits
print(allFruits)
```

```
['apple', 'orange', 'pear']
['kiwi', 'strawberry']
['apple', 'orange', 'pear', 'kiwi', 'strawberry']
```

Access a specific item in the list:

In [107]:

```
fruits[0]           # first item of fruits
```

Out[107]:

```
'apple'
```

In [111]:

```
fruits[0] = 'durian'
fruits
```

Out[111]:

```
['durian', 'orange', 'pear']
```

In [112]:

```
rojak[3]
```

Out[112]:

```
[1, 2, 3]
```

In [113]:

```
rojak[3] = 'new item'
rojak
```

Out[113]:

```
[11, 'hello', True, 'new item', ['one', 2, False]]
```

Slice indexing: accessing a range of item in the list

We can also index multiple adjacent elements using the slice operator. For instance, `fruits[1:3]` , returns a list containing the elements at position 1 and 2. In general fruits[start:stop] will get the elements in `start` , `start` +1, ..., `stop` -1. We can also do `fruits[start:]` which returns all elements starting from the start index. Also `fruits[:end]` will return all elements before the element at position end:

In [116]:

```
print(allFruits)    # print all items

print(allFruits[0:2])  # items 0 and 1 (not including 2)
print(allFruits[:3])   # first item to item 2 (not including 3)
print(allFruits[2:])   # items 2 to last item
```

```
['apple', 'orange', 'pear', 'kiwi', 'strawberry']
['apple', 'orange']
['apple', 'orange', 'pear']
['pear', 'kiwi', 'strawberry']
```

Negative indexing

Python also allows negative-indexing from the back of the list. For instance, `allfruits[-1]` will access the last element `'strawberry'` :

In [117]:

```
print(allFruits)
allFruits[-1]
```

```
['apple', 'orange', 'pear', 'kiwi', 'strawberry']
```

Out[117]:

```
'strawberry'
```

The following codes combine negative indexing and slice indexing

In [118]:

```
allFruits[-3:]       # Start from the third last item to the end of the list
```

Out[118]:

```
['pear', 'kiwi', 'strawberry']
```

In [119]:

```
allFruits[-3:-1]     # Start from the third last item to the second last item of the list
```

Out[119]:

```
['pear', 'kiwi']
```

Reference vs Shallow vs Deep Copy

Reference copy of a list

For list, the assignment operator `=` only copy the address of the source. This means that it is *pointing* or *referencing* to the same object as the source.

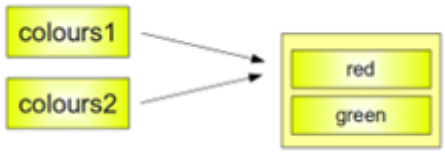
For example:

In [120]:

```
colours1 = ["red", "green"]
colours2 = colours1
print ('colours1 = ', colours1)
print ('colours2 = ', colours2)
```

```
colours1 =  ['red', 'green']
colours2 =  ['red', 'green']
```

The above code results in the following structure:



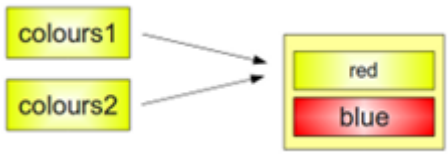
Since they point to the same list, any changes to either `colour1` or `colour2` would affect the other.

In [121]:

```
print ('Changing colours2')
colours2[1] = 'blue'                          # changing the content of the second item in co

print ('colours1 = ', colours1)               # colours1 changes too because of shallow copy
print ('colours2 = ', colours2)
```

```
Changing colours2
colours1 =  ['red', 'blue']
colours2 =  ['red', 'blue']
```



Shallow copy of a list

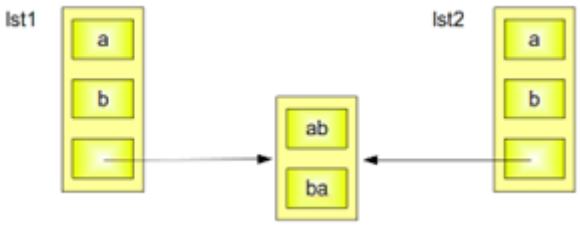
The command `.copy()` performs a shallow copy where only the content of the shallowest level are copied.

In [122]:

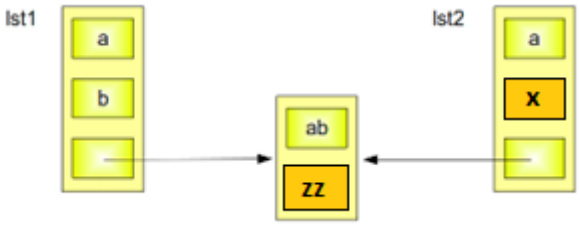
```
list1 = ['a','b',['ab', 'ba']]      # list is shallow - contains only one level
list2 = list1.copy()                # copy list
print (list1)
print (list2)
```

```
['a', 'b', ['ab', 'ba']]
['a', 'b', ['ab', 'ba']]
```

The above command results in the following structure:



From the figure, it is easy to see why changing the content of the shallow items of one array would not have affect the other array. But, any changes to non-shallow items will affect both arrays:



In [123]:

```
list2[1] = 'x'                # changing a shallow item in list 2  will not affect list1
list2[2][1] = 'zz'            # changing a non-shallow item in list2 will also affect list1
print (list1)
print (list2)
```

```
['a', 'b', ['ab', 'zz']]
['a', 'x', ['ab', 'zz']]
```

Deep copy of list

To copy the whole structure, we have to use the library `copy.deepcopy` . The function copies the list wholely.

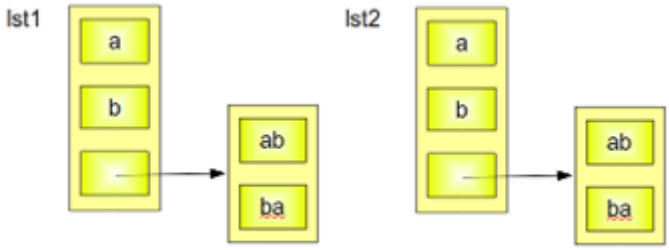
For example:

In [124]:

```
from copy import deepcopy

lst1 = ['a','b',['ab','ba']]
lst2 = deepcopy(lst1)   # slice copying
```

The above commands result in the following structure.

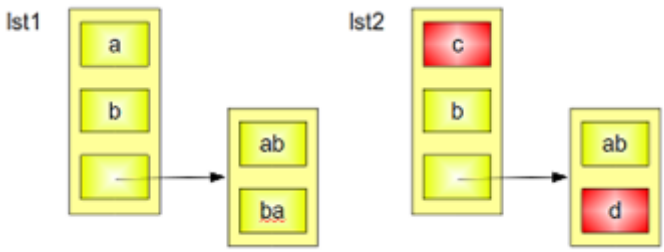


Therefore, any changes to either `lst1` or `lst2` would NOT affect the other.

In [125]:

```
lst2[0] = 'c'
lst2[2][1] = 'd'
print('lst1: ', lst1)
print('lst2: ', lst2)
```

```
lst1:  ['a', 'b', ['ab', 'ba']]
lst2:  ['c', 'b', ['ab', 'd']]
```



Difference between == and is statement

- **The operator `is`** checks if two variables are referencing the same list or object.
- **The operator `==`** checks if the lists have the same content.

For *reference copy*, both `is` and `==` return `True` because they are referencing the same list or object.

In [126]:

```
a = [1, 2, 3]
b = a
print(b is a)
print(b == a)
```

```
True
True
```

For *shallow copy* and *deep copy*:

- `is` returns `False` because they are pointing to different lists or objects.
- `==` returns `True` because their content are the same.

In [128]:

```
b = a.copy()
print(b is a)#different reference
print(b == a)
```

```
False
True
```

In [129]:

```
from copy import deepcopy
b = deepcopy(a)
print(b is a)
print(b == a)
```

```
False
True
```

Useful functions for processing list

1. **`list.append(x)`** : Add an item to the end of the list; equivalent to `a[len(a):] = [x]` .
2. **`list.extend(x)`** : Extend the list by appending all the items in the given list; equivalent to `a[len(a):] = L` .
3. **`list.insert(i, x)`** : Insert the item x at position i. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)` .
4. **`list.remove(x)`** : Remove the first item from the list whose value is `x` . It is an error if there is no such item.
5. **`list.pop(i)`** : Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list.
6. **`list.index(x)`** : Return the index in the list of the first item whose value is `x` . It is an error if there is no such item.

7. **`List.count(x)`** : Return the number of times `x` appears in the list
8. **`list.sort(key=None, reverse=False)`** : Sort the items of the list in place
9. **`list.reverse()`** : Reverse the elements of the list, in place.
10. **`len(list)`** : Returns the number of items in the sequence
11. **`min(list)`** : Returns the smallest item in the sequence
12. **`max(list)`** : Returns the biggest item in the sequence
13. **`any(list)`** : Returns `True` if there exists an item in the sequence which is `True`
14. **`all(list)`** : Returns `True` if all items in the seuqnece are True

Exercise 4

(Q1) Create a list `A` which stores the values a^i where $i = 0...9$. Prompt the users for a.

Answer:

```
Please enter base: 2
A =  [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

In [173]:

```
base = int(input('Please enter base: '))
A = []
# complete the code
for i in range(10):
    A.append(base**i)
A
```

```
Please enter base: 2
```

Out[173]:

```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

In [174]:

```
A[:4]
```

Out[174]:

```
[1, 2, 4, 8]
```

(Q2) Given a list `A = [21, 27, 34, 23, 3, 22, 33, 1]` . Extract all items of `A` is divisble by 3 and store your result into a list called `divisibleBy3` . Create another list called `indices` which store the original location of all items in `divisibleBy3` in `A`

Ans:

```
divisibleBy3 = [21, 27, 3, 33]
indices = [0, 1, 4, 6]
```

In [142]:

```
A = [21, 27, 34, 23, 3, 22, 33, 1]
divisibleBy3 = []
indices = []
for i,num in enumerate(A):
    if num%3==0 :
        divisibleBy3.append(num)
        indices.append(i)
print(divisibleBy3)
print(indices)

# complete the code
```

```
[21, 27, 3, 33]
[0, 1, 4, 6]
```

(Q3) Create the following two lists:

names = ['John', 'Mokhtar', 'Choon Kit', 'Ramasamy']
scores = [10, 20, 16, 29]

Then use a program to combine the two lists into one list as follows:

Ans: `combined = [['John', 10], ['Mokhtar', 20], ['Choon Kit', 16], ['Ramasamy', 29]]`

In [146]:

```
names = ['John', 'Mokhtar', 'Choon Kit', 'Ramasamy']
scores = [10, 20, 16, 29]

combined= []

for n,s in zip(names,scores):
    combined.append([n,s])
combined

# complete the code
```

Out[146]:

```
[['John', 10], ['Mokhtar', 20], ['Choon Kit', 16], ['Ramasamy', 29]]
```

Tuple

A tuple is a data structure that is similar to the list except that it is immutable once it is created, i.e., you cannot change its content once created. It takes up less memory for the same number of items compared to list. So, it is advisable to use tuples compared to list if possible.

To **create a tuple**, use a parentheses rather than a square brackets.

In [147]:

```
#create then cannot change
pair = (3,5)
print(pair)
print(pair[0])
```

```
(3, 5)
3
```

The attempt to modify an immutable structure will raise an exception. Exceptions indicate errors: index out of bounds errors, type errors, and so on will all report exceptions in this way.

In [148]:

```
pair[1] = 6
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-148-392fbb166e50> in <module>
----> 1 pair[1] = 6

TypeError: 'tuple' object does not support item assignment
```

Set

A set is another data structure that serves as an unordered list with **no duplicate items**. Below, we show how to create a set, add things to the set, test if an item is in the set, and perform common set operations (difference, intersection, union):

Create a new set from list:

In [149]:

```
#not duplicate
shapes = ['circle','square','triangle','circle']
setOfShapes = set(shapes)
setOfShapes
```

Out[149]:

```
{'circle', 'square', 'triangle'}
```

Insert new items into a set:

In [150]:

```
setOfShapes.add('polygon')
setOfShapes
```

Out[150]:

```
{'circle', 'polygon', 'square', 'triangle'}
```

A set would not add items that already exist in the set. Note that the items in `setOfShapes` remain unchanged after the following two `add` operations.

In [151]:

```
setOfShapes.add ('circle')
setOfShapes.add('triangle')
print(setOfShapes)
```

```
{'triangle', 'square', 'circle', 'polygon'}
```

Check if exist in set:

In [152]:

```
if 'circle' in setOfShapes:
    print ('circle exists in setOfShapes')
else:
    print ('circle does not exist in setOfShapes')

if 'rhombus' in setOfShapes:
    print ('rhombus exists in setOfShapes')
else:
    print ('rhombus does not exist in setOfShapes')
```

```
circle exists in setOfShapes
rhombus does not exist in setOfShapes
```

Operations on set:

The following shows how to perform intersection (`&`), union (`|`) and deletion (`-`) operation on two sets.

In [153]:

```
favoriteShapes = ['circle','triangle','hexagon']
setOfFavoriteShapes = set(favoriteShapes)
print ('setOfShapes:         ', setOfShapes)
print ('setOfFavoriteShapes: ', setOfFavoriteShapes)
print ()

print('setOfShapes - setOfFavoriteShapes:', setOfShapes - setOfFavoriteShapes)
print('setOfShapes & setOfFavoriteShapes:', setOfShapes & setOfFavoriteShapes)
print('setOfShapes | setOfFavoriteShapes:', setOfShapes | setOfFavoriteShapes)
```

```
setOfShapes:          {'triangle', 'square', 'circle', 'polygon'}
setOfFavoriteShapes:  {'triangle', 'circle', 'hexagon'}

setOfShapes - setOfFavoriteShapes: {'square', 'polygon'}
setOfShapes & setOfFavoriteShapes: {'triangle', 'circle'}
setOfShapes | setOfFavoriteShapes: {'triangle', 'square', 'circle', 'polygo
n', 'hexagon'}
```

Exercise 5

(Q1) Write a program that gets a total of 5 strings from users and then display a list of the unique words found in

the list.

Example output:

```
Please enter strings: hello how do you do
{'hello', 'how', 'you', 'do'}
```

Hints: use

- `<string>.split()` to split a string into a list of words.
- Create a `set` object to keep a unique set of items from a list

In [154]:

```
s = input('Please enter strings: ')
words = s.split()
setofword = set(words)
print(setofword)
```

```
Please enter strings: hello how do you do
{'do', 'hello', 'you', 'how'}
```

Dictionary

The last built-in data structure is the dictionary which stores a map from one type of object (the `key`) to another (the `value`).

- The `key` must be an immutable type (string, number, or tuple).
- The `value` can be any Python data type.

Creating a dictionary:

There are many ways to create a dictionary.

(1) The standard way is to use the curly bracket `{}`.

In [155]:

```
studentIds = {'joe': 42.0, 'mary': 56.0}
studentIds
```

Out[155]:

```
{'joe': 42.0, 'mary': 56.0}
```

(2) The second way is to pass a list of tuples to the dict constructor:

In [156]:

```
d1 = dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
d1
```

Out[156]:

```
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

Notes: In the above example, *the order of the keys returned by Python would be different than the order in the code*. Therefore, do NOT rely on key ordering when using dictionary.

(3) The third way is to pass keyword arguments to the dict constructor:

In [157]:

```
d2 = dict(sape=4139, guido=4127, jack=4098)
d2
```

Out[157]:

```
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

(4) Lastly, you can also use two different lists, one for keys and the other for values. Then, we can construct the dictionary using list comprehensions (to be covered next).

In [158]:

```
namelist = ['joe', 'mary']
Ids = [42.0, 56.0]
d3 = {name: age for name, age in zip(namelist, Ids)}
d3
```

Out[158]:

```
{'joe': 42.0, 'mary': 56.0}
```

Indexing an item in dictionary:

In [159]:

```
studentIds = {'joe': 42.0, 'mary': 56.0}
print (studentIds)

# accessing the item with key 'mary'
print(studentIds['mary'])


# Changing the item with key 'mary'
studentIds['mary'] = 100
print(studentIds)
```

```
{'joe': 42.0, 'mary': 56.0}
56.0
{'joe': 42.0, 'mary': 100}
```

Adding an item in dictionary:

In [160]:

```
studentIds['ali'] = 92.0
studentIds
```

Out[160]:

```
{'joe': 42.0, 'mary': 100, 'ali': 92.0}
```

The values can be of different types:

In [161]:

```
studentIds['joe'] = 'forty-two'          # string type
print(studentIds)

studentIds['ali'] = [42.0,'forty-two']  # list type
print(studentIds)
```

```
{'joe': 'forty-two', 'mary': 100, 'ali': 92.0}
{'joe': 'forty-two', 'mary': 100, 'ali': [42.0, 'forty-two']}
```

Getting the size of a dictionary:

In [162]:

```
len(studentIds)
```

Out[162]:

```
3
```

Accessing the keys and values in dictionary:

In [163]:

```
print(studentIds.keys())
print(studentIds.values())
```

```
dict_keys(['joe', 'mary', 'ali'])
dict_values(['forty-two', 100, [42.0, 'forty-two']])
```

Converting the dict_keys and dict_values directly to list:

In [164]:

```
print(list(studentIds.keys()))
for k in list(studentIds.keys()):
    print(k)

print(list(studentIds.values()))
for v in list(studentIds.values()):
    print(v)
```

```
['joe', 'mary', 'ali']
joe
mary
ali
['forty-two', 100, [42.0, 'forty-two']]
forty-two
100
[42.0, 'forty-two']
```

Accessing items, keys and values in a dictionary:

- `<dictionary>.items()` is an iterable object. When used with `for`, it iterates through the list of all *items* in `dictionary`, returning one item at a time. Each item is a tuple `(key, value)`.
- `<dictionary>.keys()` is an iterable object. When used with `for`, it iterates through the *keys* of all items in `dictionary`, returning one key at a time.
- `<dictionary>.values()` is an iterable object. When used with `for`, it iterates through the *values* of all items in `dictionary`, returning one value at a time.

In [165]:

```
print('All items in studentIds: ')
for item in studentIds.items():
    print (item[0], ':', item[1])   # item[0] is the key, item[1] is the value

print('\nThe keys of all items in studentIds: ')
for key in studentIds.keys():
    print (key)

print('\nThe values of all items in studentIds: ')
for value in studentIds.values():
    print (value)
```

```
All items in studentIds:
joe : forty-two
mary : 100
ali : [42.0, 'forty-two']

The keys of all items in studentIds:
joe
mary
ali

The values of all items in studentIds:
forty-two
100
[42.0, 'forty-two']
```

Removing an item from dictionary:

In [166]:

```
del studentIds['joe']
studentIds
```

Out[166]:

```
{'mary': 100, 'ali': [42.0, 'forty-two']}
```

As with nested lists, you can also create dictionaries of dictionaries.

Exercise 6

Q1. Write a Python script to create the following dictionary `{0: 10, 1: 20}` . Then, add the following new key to the dictionary `{2: 30}` . Then remove the record with key `1` .

In [178]:

```
dict = {0:10,1:20}
print(dict)
print(dict[0])
dict[2]=30
print(dict)
del dict[1]
print(dict)
```

```
{0: 10, 1: 20}
10
{0: 10, 1: 20, 2: 30}
{0: 10, 2: 30}
```

Q2. Create the dictionary `d = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}` . Then ask the user for a key. If the key is present, then return the value for the key. Else, inform the user that `Key is not present in the dictionary`

In [170]:

```
d = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
k = int(input('Please enter a present key: '))
if k in d.keys():
    print(d[k])
else:
    print('Key is not present in the dictionary')
```

```
Please enter a present key: 1
10
```

Q3. Write a program that gets a string from the user. The program should create a dictionary in which the keys are the unique words found in the string and the values are the number of times each word appears. For example, if 'the' appears 2 times, the dictionary would contain a 'the' as the key and 2 as the value

In [184]:

```
s = 'The program should create a dictionary in which the keys are the unique words found in
dict={}
word = s.lower().split()
uword = set(word)
count = 0
#for w in uword:
    #for w1 in word:
     #   if w == w1:
    #        count = count + 1
   # dict[w]=count
 #   count=0
#dict

for w in word:
    if w in dict.keys():
        dict[w] += 1
    else:
        dict[w] = 1
dict
```

Out[184]:

```
{'the': 6,
 'program': 1,
 'should': 1,
 'create': 1,
 'a': 1,
 'dictionary': 1,
 'in': 2,
 'which': 1,
 'keys': 1,
 'are': 2,
 'unique': 1,
 'words': 1,
 'found': 1,
 'string': 1,
 'and': 1,
 'values': 1,
 'number': 1,
 'of': 1,
 'times': 1,
 'each': 1,
 'word': 1,
 'appears': 1}
```

LIST COMPREHENSION

List comprehension provide a concise way to create a lists. Let's say you want to create a list of squares. The normal way to do this is as follows:

In [176]:

```
squares = []
for x in range(10):
    squares.append (x**2)
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehension can help you represent the loop more concisely as follows:

In [185]:

```
squares2 = [x**2 for x in range(10)]
print(squares2)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

You can also do filtering on the items that you want to operate on. Let's say we want to extract only odd numbers from a list and then multiply all of them by 100:

In [186]:

```
nums = [1, 2, 3, 4, 5, 6, 7]
result = [x*100 for x in nums if x % 2 == 1]
print (result)
```

```
[100, 300, 500, 700]
```

So the basic syntax for List Comprehension is: [expression **for** item **in** list **if** conditional]
This is equivalent to:
for item **in** list:
    if conditional:
        expression

Exercise 7

Q1. Generate an array of 10 numbers with random numbers between 0 and 20. To generate random number, use the command

```
import numpy as np
np.random.randint(a, b).
```

In [188]:

```
import numpy as np
random = [ np.random.randint(0, 20) for n in range(10) ]
print(random)
```

```
[9, 11, 14, 15, 3, 7, 18, 5, 11, 9]
```

Q2. Let's say I give you a list saved in a variable: `a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]` . Write one line of Python code that takes this list *a* and makes a new list that has only the even elements of this list in

it.

In [189]:

```
a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even = [ num for num in a if num%2==0 ]
even
```

Out[189]:

```
[4, 16, 36, 64, 100]
```

In [ ]:

```

```