Python Workshop 1

Programming is all about data manipulation. You need to tell computers what to do. There are two main things that you need to specify:

1. Variables
2. Control Flows

1. Variable

Before you can modify data, first you need to store them in the variables. Here is how you create a variable:

In [48]:

```
x = 10
y = 5.2
z = 'abc'
r = True
s = False
```

1.1 Data Types

Computers need to know the data types (what kind of values are stored in variables) so they can manipulate the data correctly (More details https://docs.python.org/3/library/types.html (https://docs.python.org/3/library/types.html)).

In [2]:

```
type(x)
```

Out[2]:

```
int
```

In [3]:

```
type(y)
```

Out[3]:

```
float
```

In [4]:

```
type(z)
```

Out[4]:

```
str
```

In [5]:

```
type(r)
```

Out[5]:

```
bool
```

In [6]:

```
type(s)
```

Out[6]:

```
bool
```

1.2 Operations

You use operations on variables to modify them.

`int` Basic Operations

In [7]:

```
x + 10
```

Out[7]:

```
20
```

In [8]:

```
x - 4
```

Out[8]:

```
6
```

In [9]:

```
x * 2
```

Out[9]:

```
20
```

In [10]:

```
x / 3
```

Out[10]:

```
3.3333333333333335
```

In [11]:

```
x // 3
```

Out[11]:

```
3
```

In [12]:

```
x % 3
```

Out[12]:

```
1
```

In [49]:

```
hex(x)
```

Out[49]:

```
'0xa'
```

In [50]:

```
bin(x)
```

Out[50]:

```
'0b1010'
```

The last operation is called Modulo and it returns the remainder from division

`float` Basic Operations

In [13]:

```
y**2
```

Out[13]:

```
27.040000000000003
```

In [14]:

```
(y*10) + 2
```

Out[14]:

```
54.0
```

`str` Basic Operations

`print` displays string on the command line

In [15]:

```
print(z)
```

```
abc
```

In [16]:

```
print('สวัสดีครับ')
```

```
สวัสดีครับ
```

In [17]:

```
print('Hello World'.upper())
```

```
HELLO WORLD
```

In [15]:

```
print('Hello World'.lower())
```

```
hello world
```

Concatenation using + between strings

In [18]:

```
text = 'Hello World '
print(text+z)
```

```
Hello World abc
```

`input` gets a string input from the user through command line

In [19]:

```
text = input()
print(text)
```

```
Haha
```

In [20]:

```
text = "Hi! "
name = input("What is your name?  ")
print(text+name)
```

```
Hi! Yoyo
```

`split` divides string using delimiter

In [18]:

```
arr = "Hello world Hi My name is SCiUs"
arr.split(' ')
```

Out[18]:

```
['Hello', 'world', 'Hi', 'My', 'name', 'is', 'SCiUs']
```

String formatting

Why use `print` ? When you can show the value of a variable by just typing its name.

In [22]:

```
msg = '''Hello {}!
My name is {} {}
This is the tutorial version {}'''.format('John', 'Sarun', 'Gulyanon', 4)
msg
```

Out[22]:

```
'Hello John!\nMy name is Sarun Gulyanon\nThis is the tutorial versio
n 4'
```

In [23]:

```
print(msg)
```

```
Hello John!
My name is Sarun Gulyanon
This is the tutorial version 4
```

In [54]:

```
msg2 = "Hello world my name is %s id=%04d wigth=%.2f"%("sothana",10,72.2345678)
print(msg2)
```

```
Hello world my name is sothana id=0010 wigth=72.23
```

Casting

Change the data types of variables

In [24]:

```
str(x)
```

Out[24]:

```
'10'
```

In [25]:

```
int(y)
```

Out[25]:

```
5
```

In [16]:

```
float(x)
```

Out[16]:

```
10.0
```

1.3 Indexing and Slicing

From a string, you can select just a character or substring.

In [19]:

```
print(arr)
```

```
Hello world Hi My name is SCiUs
```

Indexing allows you to pick an element from a string. In Python, the first element has index 0.

In [27]:

```
print(arr[0])
print(arr[4])
print(arr[-1])
```

```
H
o
s
```

Slicing allows you to pick a substring. When you slice a string, it doesn't include the last element.

In [20]:

```
print(arr[1:4])
print(arr[:4])
print(arr[6:])
print(arr[-5:])
print(arr[:])
print(arr[::2])
```

```
ell
Hell
world Hi My name is SCiUs
SCiUs
Hello world Hi My name is SCiUs
HlowrdH ynm sSis
```

1.4 Collections

So far we handle only one variable at a time. What if we have multiple variables that we want to modify? Python have 4 basic types of arrays (a data type of handling a group of variables). But we will look at three of them.

(More details https://docs.python.org/3/tutorial/datastructures.html (https://docs.python.org/3/tutorial/datastructures.html))

1.4.1. List

List is the most versatile one. It is ordered and changeable and it can store variables with any type

In [29]:

```
a1 = []
type(a1)
```

Out[29]:

```
list
```

In [30]:

```
a2 = [2,5,3,8,6]
print(a2)
```

```
[2, 5, 3, 8, 6]
```

Use `len` to count the number of elements in list

In [31]:

```
len(a2)
```

Out[31]:

```
5
```

In [32]:

```
w = ['bird','ant','dog','cat']
print('size : ', len(w))
```

```
size :  4
```

Use `append` to add an item at the end of the list

In [33]:

```
a2.append(7)
print(a2)
```

```
[2, 5, 3, 8, 6, 7]
```

Use `sort` to order the items in the list.

In [34]:

```
a2.sort()
print(a2)
```

```
[2, 3, 5, 6, 7, 8]
```

In [35]:

```
print(w)
w.sort()
print(w)
```

```
['bird', 'ant', 'dog', 'cat']
['ant', 'bird', 'cat', 'dog']
```

You can index and slice list as well.

In [36]:

```
print(w[1])
print(a2[1:3])
```

```
bird
[3, 5]
```

In [37]:

```
w[1] = 'bee'
print(w)
a2[1:3] = [9, 11]
print(a2)
```

```
['ant', 'bee', 'cat', 'dog']
[2, 9, 11, 6, 7, 8]
```

In [38]:

```
e = [1,2,3,4,5,6]
e[::-1]
```

Out[38]:

```
[6, 5, 4, 3, 2, 1]
```

1.4.2. Tuple

Tuple is ordered and unchangeable. It supports two operations: pack and unpack. **Packing** variables bundles them into a tuple:

In [39]:

```
e = (1, 'a', 3.2)
print(e[0])
print(e[2])
```

```
1
3.2
```

Unpacking unbundles variables.

In [40]:

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

```
1
3.2
```

In [25]:

```
g = []
for i in range(10):
    x = i;
    y = i*i;
    g.append((x,y))
g
```

Out[25]:

```
[(0, 0),
 (1, 1),
 (2, 4),
 (3, 9),
 (4, 16),
 (5, 25),
 (6, 36),
 (7, 49),
 (8, 64),
 (9, 81)]
```

In [26]:

```
for x,y in g:
    print("x:{} , y:{}".format(x,y))
```

```
x:0 , y:0
x:1 , y:1
x:2 , y:4
x:3 , y:9
x:4 , y:16
x:5 , y:25
x:6 , y:36
x:7 , y:49
x:8 , y:64
x:9 , y:81
```

1.4.3. Dictionaries

Dictionary is unordered, changeable, and indexed. No duplicate members are allowed. It store data in pairs of key and value.

In [41]:

```
level = {'m6':615,'m5':515,'m4':415}
level['m5']
```

Out[41]:

```
515
```

In [42]:

```
wrds = {1:'ant', 2:'bird', 3:'cat', 4:'dog'}
wrds[3]
```

Out[42]:

```
'cat'
```

In [43]:

```
wrds.keys()
```

Out[43]:

```
dict_keys([1, 2, 3, 4])
```

In [44]:

```
wrds.values()
```

Out[44]:

```
dict_values(['ant', 'bird', 'cat', 'dog'])
```

In [45]:

```
wrds.items()
```

Out[45]:

```
dict_items([(1, 'ant'), (2, 'bird'), (3, 'cat'), (4, 'dog')])
```

Membership operator test if the collections contains a specific variable.

In [46]:

```
'm6' in level
```

Out[46]:

```
True
```

In [47]:

```
'ant' in wrds
```

Out[47]:

```
False
```

In [48]:

```
'ant' in wrds.values()
```

Out[48]:

```
True
```

In [49]:

```
'ant' not in wrds
```

Out[49]:

```
True
```

In [50]:

```
print(e)
'a' in e
```

```
(1, 'a', 3.2)
```

Out[50]:

```
True
```

In [51]:

```
print(a2)
3 in a2
```

```
[2, 9, 11, 6, 7, 8]
```

Out[51]:

```
False
```

1.4.4. Set

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

In [33]:

```
sets = {}
print(type(sets),sets)
```

```
<class 'dict'> {}
```

In [44]:

```
setA = {1,2,3,4}
setB = {*()}
print("A",type(setA),setA)
print("B",type(setB),setB)
```

```
A <class 'set'> {1, 2, 3, 4}
B <class 'set'> set()
```

In [45]:

```
setB.union(setA)
```

Out[45]:

```
{1, 2, 3, 4}
```

In [46]:

```
setB.intersection(setA)
```

Out[46]:

```
set()
```

[top]

2. Control Flows

When you gives orders to the computers, you can only control the order of the instructions to be executed (control flows)

2.1.If Else Statement

`if` statement adds a condition into your program. The code block will be executed only if the condition is true.

In [52]:

```
a = 1
b = 1
if a==b:
    print('a and b are equaled')
```

```
a and b are equaled
```

Code block is a piece of program that can be executed as a unit. In Python, the indentation is used for marking code blocks.

Conditions

Python supports the usual logical conditions from mathematics:

In [53]:

```
print(a == b)
print(a != b)
print(a < b)
print(a <= b)
print(a > b)
print(a >= b)
```

```
True
False
False
True
False
True
```

`else` statement will be executed if the condition is false.

In [54]:

```
a = 1
b = 2
if a==b:
    print('a and b are equaled')
else:
    print('a and b are not equaled')
```

```
a and b are not equaled
```

`elif` statement tests a condition if the previous condition(s) were not true.

In [55]:

```
a = 1
b = 2
if a==b:
    print('a and b are equaled')
elif a<b:
    print('a is smaller than b')
else:
    print('a and b are not equaled')
```

```
a is smaller than b
```

In [56]:

```
number = input("Enter some number : ")
n = int(number) or float(number)
if n%2==0:
    print("Even number")
else:
    print("Odd number")
```

```
Odd number
```

For multiple conditions, use `and` or `or` to combine them.

In [57]:

```
c = 3
if a < b and c > a:
    print("Both conditions are True")
```

```
Both conditions are True
```

In [58]:

```
if a > b or c > a:
      print("At least one of the conditions is True")
```

```
At least one of the conditions is True
```

2.2. For loop

`for` loop is used for iterating over a sequence.

In [59]:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
      print(x)
```

```
apple
banana
cherry
```

In [60]:

```
for x in 'banana':
    print(x)
```

```
b
a
n
a
n
a
```

To loop through a set of code a specified number of times, use the `range` function. It returns a sequence of numbers.

In [61]:

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

```
0
1
2
3
4
```

In [62]:

```
for i in range(30,5,-3):
    print(i)
```

```
30
27
24
21
18
15
12
9
6
```

Break

`break` statement stops the loop before it has looped through all the items.

In [63]:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
    if x == "banana":
        break
```

```
apple
banana
```

Continue

`continue` stop the current iteration of the loop, and continue with the next.

In [64]:

```
for x in fruits:
    if x == "banana":
        continue
    print(x)
```

```
apple
cherry
```

2.3. While Loop

`while` loop allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating `if` statement.

In [47]:

```
k = 0
i = int(input("input number "))
thes = 100
while k<thes:
    k=k+i
    print(k)
```

```
input number 20
20
40
60
80
100
```

[top]

In [ ]:

```

```