SWINBURNE UNIVERSITY OF TECHNOLOGY SARAWAK CAMPUS

Lecture 2

Python Types, Variable and simple I/O

KNOWING

Variable

Using Variables to store information

- Controls and variables temporarily store data.
- Variable: Temporary storage location in main memory
- Reasons to use variables:
  - Hold information that is not stored in control on form.
  - Enable code to run more efficiently.

KNOW ING

Selecting a Name for a variable

- Selecting a name for a variable
  - Variables are referred by name
  - Identifier: another term for variable name
- Guidelines for naming variables
  - Name should be descriptive: StudName, Num1
- Declaring a variable
  - Declaration statement: Used to declare (create) a variable and reserve space in memory of it

KNOW ING

Variable and Value

- A variable is a memory location where a programmer can store a value. Example: roll_no, amount, name etc.
- Value is either string, numeric etc. Example: “Sara”, 120, 25.36
- Variables are created when first assigned.
- Variables must be assigned before being referenced.

KNOW ING

Variable and Value

- The value stored in a variable can be accessed or updated later.
- No declaration required.
- The type (string, int, float etc.) of the variable is determined by Python.
- The interpreter allocates memory on the basis of the data type of a variable.

KNOW ING

Python Variable Name Rules

- Must begin with a letter (a-z, A-B) or underscore (_).
- Other characters can be netters, numbers or _ .
- Case Sensitive.
- Can be any (reasonable) length.
- There are some reserved words which you cannot use as a variable name because Python uses them for instructions.

Good Variable Name

- Choose meaningful name instead of short name. Example: roll_no is better than rn.
- Maintain the length of a variable name. Roll_no_of_a_student is too long.
- Be consistent: roll_no or RollNo.
- Begin a variable name with an underscore (_) character for a special case.

input() Function

Using the input() function

If_Else_statement.py - C:/Users/achoo/AppData/Local/Programs/Python/Python35-32/If_Else_statem...

File Edit Format Run Options Window Help

```
#Using the input() Function
#Create a string variable with input() Function

name = input('Hi, please enter your name: ')
print('Hi ', name, ' Welcome to Pyhton Class')
```

Ln: 6 Col: 0

```
 RESTART: C:/Users/achoo/AppData/Local/Programs/Python/Python35-32/If_Else_state
ment.py
Hi, please enter your name: Choo Ai Ling
Hi  Choo Ai Ling  Welcome to Pyhton Class
>>>
```

Ln: 75 Col: 4

KNOW ING

Demonstrates numbers and math

Chap2.py - C:/Users/achoo/AppData/Local/Programs/Python/Python35-32/Chap2.py (3.5.3)

File Edit Format Run Options Window Help

```
#Demonstrates numbers and math

print('\nIf an adventurer returns from a successful quest and buys each of')
print('6 companions 3 bottles of ale, how many bottles are purchased?')
input('Press the enter key to find out.')
print("6 * 3 = ", 6*3)
```

Python 3.5.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.5.3 (v3.5.3:1880cb95a742, Jan 16 2017, 15:51:26) [MSC v.1900 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/achoo/AppData/Local/Programs/Python/Python35-32/Chap2.py =

If an adventurer returns from a successful quest and buys each of
6 companions 3 bottles of ale, how many bottles are purchased?
Press the enter key to find out.
6 * 3 =  18
>>>
```

Ln: 8 Col: 32

Useful Mathematical Operators

Operator	Description	Example	Evaluates to
+	Addition	7 + 3	10
-	Subtraction	7 – 3	4
*	Multiplication	7 * 3	21
/	Division (True)	7 / 3	2.333333333333
//	Division (Integer)	7 // 3	2
%	Modulus	7 % 3	1

KNOW ING

Python Assignment Statements

- The assignment statement creates new variables and give them values.
- Basic assignment statement in Python's syntax as below:

<variable> = <expr>

Python Assignment Statements

- Where the equal sign (=) is used to assign value (right side) to a variable name (left side).

```
>>>
>>> Item_name = "Computer" #A String
>>> Item_qty = 10 #An Integer
>>> Item_value = 1000.23 #A floating point
>>> print(Item_name)
Computer
>>> print(Item_qty)
10
>>> print(item_value)
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    print(item_value)
NameError: name 'item_value' is not defined
>>>
```

Case sensitive

Note: assignment statement read from right to left only.

Python Assignment Statements

- Syntax error:

Python 3.5.3 Shell

File Edit Shell Debug Options Window Help

```
>>>

>>>

>>>
>>> Item_name = "Computer" #A String
>>> Item_qty = 10 #An Integer
>>> Item_value = 1000.23 #A floating point
>>> print(Item_name)
Computer
>>> print(Item_qty)
10
>>> print(item_value)
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    print(item_value)
NameError: name 'item_value' is not defined
>>>
>>>
>>>
>>>
>>>
>>> a = 12
>>> 12 = a
SyntaxError: can't assign to literal
>>>
```

Ln: 88 Col: 4

12 = a does not make sense to Python

Multiple Assignment

- The basic assignment statement works for a single variable and a single expression.
- Syntax:

var1 = var2 = var3 … = <expr>

```
>>> x = y = z = 12
>>> print(x)
12
>>> print(y)
12
>>> print(z)
12
```

Multiple Assignment

- Here is an another assignment statement where the variables assign to many values at the same time
- Syntax:

var1 = var2 = var3 … = <expr>, <expr> … <expr>

```
>>>
>>> x,y,z = 1, 2, "abc"
>>> print(x)
1
>>> print(y)
2
>>> print(z)
abc
>>>
```

Multiple Assignment

- You can reuse variable names by simply assigning a new value to them.

```
>>> x = 100
>>> print(x)
100
>>> x = "Python"
>>> print(x)
Python
>>>
```

Swap Variables

- Python swap values in a single line and this applies to all objects in Python.
- Syntax:

var1, var2 = var2, var1

```
>>> x = 10
>>> y = 100
>>> print(x)
10
>>> print(y)
100
>>> x,y = y, x
>>> print(x)
100
>>> print(y)
10
>>> |
```

Local and Global Variables in Python

- In Python, variables that are only referenced inside a function are implicitly global. If a variable is assigned a value anywhere within the function's body.
- It is assumed to be local unless explicitly declared as global.

Local Variables in Python

Samples1.py - C:/Users/achoo/Desktop/Samples1.py (3.5.3)

File Edit Format Run Options Window Help

```python
nam1 = "Python"
def func1():
    nam1 = "PHP"
    print("In func1() nam1 is ", nam1)

def func2():
    print("In func2() nam1 is ", nam1)

func1()
func2()
```

Ln: 12 Col: 0

```
In func1() nam1 is  PHP
In func2() nam1 is  Python
>>>
```

Ln: 200 Col: 4

KNOW ING

Global Variables in Python

- Declare a global variable in other functions by using the global keyword.

Samples1.py - C:/Users/achoo/Desktop/Samples1.py (3.5.3)

File Edit Format Run Options Window Help

```
nam1 = "Python"
def func1():
    global nam1
    nam1 = "PHP"
    print("In func1() nam1 is ", nam1)

def func2():
    print("In func2() nam1 is ", nam1)

func1()
func2()
```

Ln: 13 Col: 0

```
In func1() nam1 is  PHP
In func2() nam1 is  PHP
>>>
```

Ln: 204 Col: 4

KNOW ING

SWINBURNE UNIVERSITY OF TECHNOLOGY SARAWAK CAMPUS

Python Data Type

KNOWING

Numbers

- Numbers are created by numeric literals. Numeric objects are immutable, which means when an object is created its value cannot be changed.
- Three distinct numeric types: Integers, Floating point numbers, and Complex numbers.

Numbers

- Integer represent negative and positive integers without fractional parts.
- Floating point numbers represents negative and positive numbers with fractional parts.
- Booleans are a subtype of plain integers,.

Numbers

Python 3.5.3 Shell

File Edit Shell Debug Options Window Help

```
>>>
>>>
>>> a = 1234
>>> type(a)
<class 'int'>
>>> b = (-1234)
>>> type(b)
<class 'int'>
>>> c = 0
>>> type(c)
<class 'int'>
>>> aa = 12.34
>>> type(aa)
<class 'float'>
>>> bb = (-12.34)
>>> type(bb)
<class 'float'>
>>> cc = 0.1234
>>> type(cc)
<class 'float'>
>>> dd = 2.12e-10
>>> type(dd)
<class 'float'>
>>> ee = 5e220
>>> type(ee)
<class 'float'>
>>>
```

Ln: 235 Col: 4

KNOW ING

Numbers

- Mathematically, a comple number is a number of the form A+Bi where i is the imaginary number.
- Complex numbers have a real and imaginary party.
- Python supports complex numbers either by specifying the number in (real + imagJ) or (real+imagj) form or using a built-in method complex(x,y).

Numbers

```
>>>
>>> x = complex(1,2)
>>> type(x)
<class 'complex'>
>>> print(x)
(1+2j)
>>> y = 1+2j
>>> type(y)
<class 'complex'>
>>> z = 1+2J
>>> type(z)
<class 'complex'>
>>>
```

Selected type Conversion Functions

Function	Description	Example	Returns
float(*x*)	Returns a floating-point value by converting *x*	float("10.0")	10.0
int(*x*)	Returns an Integer value by converting *x*	int("10")	10
str(*x*)	Returns a string value by converting *x*	str(x)	'10'

KNOW ING

Boolean (bool)

- The simplest build-in type in Python is the bool type, it represents the truth values False and True.

```
>>>
>>> x = True
>>> type(x)
<class 'bool'>
>>> y = False
>>> type(y)
<class 'bool'>
>>> z = true
Traceback (most recent call last):
  File "<pyshell#156>", line 1, in <module>
    z = true
NameError: name 'true' is not defined
>>>
```

Case Sensitive

Using Augmented Assignment Operators

Operator	Example	Is Equivalent to
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5

KNOW ING

String

- In Python, a string type object is a sequence (left-to-right order) of characters.
- Strings start and end with single or double quotes Python strings are immutable.
- Single and double quote strings are same and you can use a single quote within a string when it is surrounded by double quote and vice versa.

String

```
>>>
>>> str1 = "String" #Strings start and end with double quotes
>>> print(str1)
String
>>> str2 = 'String' #Strings start and end with single quotes
>>> print(str2)
String
>>> str3 = "String' #Strings start with double quotes and end with single quotes
SyntaxError: EOL while scanning string literal
>>> str3 = 'String" #Strings start with single quotes and end with double quotes
SyntaxError: EOL while scanning string literal
>>> str4 = "Day's" #Single quote within double quotes
>>> print(str4)
Day's
>>> str5 = 'Day"s' #Double quote within single quotes
>>> print(str5)
Day"s
>>>
```

KNOW ING

Creating New strings with String methods

Chap2.py - C:/Users/achoo/AppData/Local/Programs/Python/Python35-32/Chap2.py (3.5.3)

File Edit Format Run Options Window Help

```
#Creating New strings and String methods

StudName = input("Student Name: ")
print('Upper Case: ', StudName.upper())
print('Lower Case: ', StudName.lower())
print('Title format: ', StudName.title())
print('Swapcase format: ', StudName.swapcase())
```

Ln: 8 Col: 0

```
= RESTART: C:/Users/achoo/AppData/Local/Programs/Python/Python35-32/Chap2.py =
Student Name: summary
Upper Case:  SUMMARY
Lower Case:  summary
Title format:  Summary
Swapcase format:  SUMMARY
>>>
```

Ln: 46 Col: 4

KNOW ING

Useful String Methods

Method	Description
upper()	Returns the uppercase version of the string.
lower()	Returns the lowercase version of the string.
swapcase()	Returns a new string where the case of each letter is switched. Uppercase becomes lowercase and lowercase becomes uppercase.
capitalize()	Returns a new string where the first letter is capitalized and the rest are lowercase.
title()	Returns a new string where the first letter of each word is capitalized and all others are lowercase.
strip()	Returns a string where all the white space(tabs, spaces, and newlines at the beginning and end is removed.)
replace(***old***, ***new*** [,***max***])	Returns a string where occurrences of the string ***old*** are replaced with the string ***new***. The optional ***max*** limits the number of replacements.

Special Characters in Strings

- The backslash(\) character is used to introduce a special character.

Escape Sequence	Meaning
\n	Newline
\t	Horizontal Tab
\\	Backslash
\’	Single Quote
\”	Double Quote

KNOW ING

Special Characters in Strings

```
>>> print("This is a backlash (\\) mark/")
This is a backlash (\) mark/
>>> print("This is tab \t key")
This is tab [redacted] key
>>> print("These are \' single quotes\'")
These are ' single quotes'
>>> print("These are \" double quotes\"")
These are " double quotes"
>>> print("This is newline\n New Line")
This is newline
 New Line
>>>
```

String Indices and Accessing String Elements

- Strings are arrays of characters and elements of an array can be accessed using indexing.
- Indices start with 0 from left side with -1 when starting from right side.

String Indices and Accessing String Elements

- String1 = “PYTHON TUTORIAL”

Character	P	Y	T	H	O	N		T	U	T	O	R	I	A	L
Index (from left)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Index (from right)	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

String Indices and Accessing String Elements

Samples1.py - C:/Users/achoo/Desktop/Samples1.py (3.5.3)

File Edit Format Run Options Window Help

```
string1 = "PYTHON TUTORIAL"
print("Print first character from the left: ", string1[0]) #Print first character from the left
print("Print last character from the left: ", string1[14]) #Print last character from the left
print("\n")
print("Print first character from the right: ", string1[-1]) #Print first character from the right
print("Print last character from the right: ", string1[-15]) #Print last character from the right
print("\n")
print("Print 4th character from the left: ", string1[4]) #Print 4th character from the left
print("Print 4th character from the right: ", string1[-11]) #Pring 4th character from the right
print("\n")
print("Index out of range: ", string1[16])
```

Ln: 13 Col: 0

```
================ RESTART: C:/Users/achoo/Desktop/Samples1.py ================
Print first character from the left:  P
Print last character from the left:  L


Print first character from the right:  L
Print last character from the right:  P


Print 4th character from the left:  O
Print 4th character from the right:  O


Traceback (most recent call last):
  File "C:/Users/achoo/Desktop/Samples1.py", line 11, in <module>
    print("Index out of range: ", string1[16])
IndexError: string index out of range
>>>
```

Ln: 361 Col: 39

Strings are immutable

- Strings are immutable character sets. Once string is generated, you can not change any character within the string.

```
>>>
>>> string1="PYTHON TUTORIAL"
>>> string1[0]
'P'
>>> string1[0] = 'A' #Try to change the first character of the string
Traceback (most recent call last):
  File "<pyshell#197>", line 1, in <module>
    string1[0] = 'A' #Try to change the first character of the string
TypeError: 'str' object does not support item assignment
>>> string1[0]
'P'
>>> |
```

‘in’ operator in Strings

- The ‘in’ operator is used to check whether a character or a substring is present in a string or not.
- The expression returns a Boolean value.

```
>>> string1="PYTHON TUTORIAL"
>>> 'Z' in string1
False
>>> 'P' in string1
True
>>> 'TUT' in string1
True
>>> |
```

String Slicing

- To cut a substring from a string is called string slicing.
- Two (2) indices are used separated by a colon (:).
- A slice 3:7 means indices characters of 3rd, 4th, 5th and 6th positions.
- The second integer index i.e. 7th is not included.

String Slicing

Samples1.py - C:/Users/achoo/Desktop/Samples1.py (3.5.3)

File Edit Format Run Options Window Help

```
string1 = "PYTHON TUTORIAL"
print(string1[2:6])
```

Ln: 3 Col: 0

P	Y	T	H	O	N		T	U	T	O	R	I	A	L
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
================ RESTART: C:/Users/achoo/Desktop/Samples1.py ================
THON
>>>
```

Ln: 401 Col: 4

KNOW ING

Tuples

- A tuple is a container which holds a series of comma-separated values (items or elements) between parentheses.
- Tuples are immutable and can hold mix data types.

Creating Tuples

- Items (with same data type) are added with parentheses

```
>>>
>>> tup1=(0, -1, 12, 202.23, 100) #5 items are added with parentheses
>>> type(tup1)#type() function is used to get the type of tup1
<class 'tuple'>
>>> print(tup1)
(0, -1, 12, 202.23, 100)
>>> |
```

Creating Tuples

- Items (with mix data type) are added with parentheses

```
>>>
>>> tup2=('Red', 'Black', 1234, 'white'); #4 items with mixed types are added
>>> print(tup2)
('Red', 'Black', 1234, 'white')
>>>
>>>
>>> |
```

Creating Tuples

- Items are added without parentheses

```
>>>
>>> tup3='a1', 'b1', 'c1', 'd1'; #Items are added without parentheses
>>> type(tup3)
<class 'tuple'>
>>> print(tup3)
('a1', 'b1', 'c1', 'd1')
>>>
>>>
```

Creating Tuples

- To create an empty tuple or create a tuple with single element.

```
>>>
>>> empty_tup1 = () #This is command to create an empty tuple
>>> print(empty_tup1)
()
>>> single_tup1 = (100,) #Creates a tuple of single item
>>> print(single_tup1)
(100,)
>>>
>>>
```

Creating Tuples

- Elements of a tuple are indexed like other sequences. The tuple indices start at 0.

```
>>> tup2 = ('Red', 'Black', 1234, 12.34)
>>> print(tup2)
('Red', 'Black', 1234, 12.34)
>>> print(tup2[0]) #Return first element
Red
>>> print(tup2[3]) #Return last element
12.34
>>> print(tup2[4])
Traceback (most recent call last):
  File "<pyshell#240>", line 1, in <module>
    print(tup2[4])
IndexError: tuple index out of range
>>>
>>>
```

Creating Tuples

- Tuples are immutable which means it's items values are unchangeable.

```
>>>
>>> tup2 = ('Red', 'Black', 1234, 12.34)
>>> print(tup2[0])
Red
>>> tup2[0]="White" #Try to change the value of the first item
Traceback (most recent call last):
  File "<pyshell#245>", line 1, in <module>
    tup2[0]="White" #Try to change the value of the first item
TypeError: 'tuple' object does not support item assignment
>>>
>>> |
```

Slicing a tuple

- Like other sequences like strings, tuples can be sliced.
- Slicing a tuple creates a new tuple but it does not change the original tuple.

```
>>>
>>> tup2 = ('Red', 'Black', 1234, 12.34) #There are 4 elements in the tuples,
indices start at 0 and end at 3
>>> print(tup2[0:2])#cut first two items
('Red', 'Black')
>>> print(tup2[1:2])#cut second item
('Black',)
>>> print(tup2[1:-2])#cut second item from right
('Black',)
>>> print(tup2[:3])#cut first three items
('Red', 'Black', 1234)
>>>
>>> |
```

Using + and * operators in Tuples

- Use + operator to create a new tuple that is a concatenation of tuples and use * operator to repeat a tuple.

```
>>>
>>> tup1=(1,2,3)
>>> tup2=(4,5,6)
>>> tup3=(7,8,9)
>>> tup_123 = tup1+tup2+tup3 #concatenation of 3 tuples
>>> print(tup_123)
(1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> print(tup1*4) #repetition with * operator
(1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3)
>>>
>>>
```

Lists

- A list is a container which holds comma-separated values (items or elements) between square brackets where items or elements need not all have the same type.

Creating Lists

Samples1.py - C:/Users/achoo/Desktop/Samples1.py (3.5.3)

File Edit Format Run Options Window Help

```
my_list1 = [5,15,20,25] #the list contains all Integer values
my_list2 = ['red', 'white', 'black', 'yellow'] #the list contains all string values
my_list3 = ['Red', 1234, 11,2345] #the list contains a string, an integer and a float value

print(my_list1)
print(my_list2)
print(my_list3)
```

Ln: 8 Col: 0

```
================ RESTART: C:/Users/achoo/Desktop/Samples1.py ================
[5, 15, 20, 25]
['red', 'white', 'black', 'yellow']
['Red', 1234, 11, 2345]
>>>
```

Ln: 540 Col: 4

KNOW ING

Creating Lists

- A list without any element is called an empty list.

```
>>>
>>> my_list=[]
>>> print(my_list)
[]
>>>
```

List indices

- List indices work the same way as string indices, list indices start at 0.
- If an index has a positive value it counts from the beginning and similarly it counts backward if the index has a negative value.
- As positive integers are used to index from the left end and negative integers are used to index from the right end, so every item of a list gives two alternatives indices.

List indices

- Let create a list called Color_List with four(4) items.
- Color_List=[‘Red’, ‘White’, ‘Blue’, ‘Black’]

Item	Red	White	Blue	Black
Index(from left)	0	1	2	3
Index(from right)	-4	-3	-2	-1

- If give any index value which is out of range then interpreter creates an error message.

KNOW ING

List indices

```
>>>
>>> Color_List=['Red', 'White', 'Blue', 'Black'] #The list have four elements
indices start at 0 and end at 3
>>> Color_List[0] #Return the first Element
'Red'
>>> print(Color_List[0], Color_List[3]) #Print first and last elements
Red Black
>>> Color_List[-1] #Return last Element
'Black'
>>> print(Color_List[4]) #Indices is out of range
Traceback (most recent call last):
  File "<pyshell#312>", line 1, in <module>
    print(Color_List[4]) #Indices is out of range
IndexError: list index out of range
>>>
>>>
```

List Slices

- Lists can be sliced like strings and other sequences. The syntax of list slices is easy:

```
Sliced_list = List_Name(startIndex:endIndex)
```

- This refers to the items of a list starting at index startIndex and stopping just before index endIndex.
- The default values for list are 0 (startIndex) and the end (endIndex) of the list.
- If you omit both indices, the slice makes a copy of the original list.

KNOW ING

List Slices

Samples1.py - C:/Users/achoo/Desktop/Samples1.py (3.5.3)

File Edit Format Run Options Window Help

```
#The list have four elements indices start at 0 and end at 3
Color_List=['Red', 'White', 'Blue', 'Black']
print(Color_List[0:2]) #cut first 2 items
print(Color_List[1:2]) #cut second item from left
print(Color_List[1:-2]) #Cut second item from right
print(Color_List[:3]) #cut first 3 items
print(Color_List[:]) #creates copy of original list
```

Ln: 4 Col: 0

```
================ RESTART: C:/Users/achoo/Desktop/Samples1.py ================
['Red', 'White']
['White']
['White']
['Red', 'White', 'Blue']
['Red', 'White', 'Blue', 'Black']
>>>
```

Ln: 554 Col: 0

Lists are Mutable

- Items in the list are mutable i.e. after creating a list you can change any item in the list.

```
>>> Color_List=['Red', 'White', 'Blue', 'Black']
>>> print(Color_List[0])
Red
>>> Color_List[0] = "Orange" #Change the value of first item 'Red' to "Orange"
>>> print (Color_List)
['Orange', 'White', 'Blue', 'Black']
>>> print(Color_List[0])
Orange
>>>
```

Using + and * operators in List

- Use + operator to create a new list that is a concatenation of two lists and use * operator to repeat a list.

```
>>> color_list1 = ['White', 'Yellow']
>>> color_list2 = ['Red', 'Blue']
>>> color_list3 = ['Green', 'Black']
>>> color_list = color_list1 + color_list2 + color_list3
>>> print(color_list)
['White', 'Yellow', 'Red', 'Blue', 'Green', 'Black']
>>> number = [1,2,3,4]
>>> print(number[1]*4) #number[1] is value 2, thus is 2 * 4
8
>>> print(number*4) #number list repeated 4 times
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
>>>
```

Sets

- A set is an unordered collection of unique elements.
- Basic uses include dealing with set theory (which support mathematical operations like union, intersection, difference, and symmetric difference) or eliminating duplicate entries

Sets

```
>>>
>>> a = [1,2,1,3,0,0,4,7,8,6]
>>> b = [5,5,7,8,7,9,6,1,1,2]
>>> s1 = set(a) #Unique numbers in s1
>>> s2 = set(b) #Unique numbers in s2
>>> s1
{0, 1, 2, 3, 4, 6, 7, 8}
>>> s2
{1, 2, 5, 6, 7, 8, 9}
>>> s1 - s2 #numbers in s1 but not in s2
{0, 3, 4}
>>> s1 | s2 #numbers in either s1 or s2
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> s1 & s2 #numbers in both s1 and s2
{8, 1, 2, 6, 7}
>>> s1 ^ s2 #numbers in s1 or s2 but not both
{0, 3, 4, 5, 9}
>>>
```

Dictionaries

- Python dictionary is a container of the unordered set of objects like lists.
- The objects are surrounded by curly braces {}.
- The items in a dictionary are a comma-separated list of key:value pairs where keys and values are Python data type.
- Each object or value accessed by key and keys are unique in the dictionary.

Dictionaries

- As keys are used of indexing, they must be the immutable type (string, number, or tuple).
- You are create an empty dictionary using empty curly braces.

```
>>>
>>> pd={"Class" : 'V', "Section" : 'A', "roll_no" :12}
>>> print(pd["Class"])
V
>>> print(pd["Section"])
A
>>> print(pd["roll_no"])
12
>>> print(pd)
{'Section': 'A', 'roll_no': 12, 'Class': 'V'}
>>>
```

Making Decisions in a Program

- Three basic control structures
  - Sequence
  - Selection
  - Repetition
- All procedures in an application are written using one of more of these structures.

KNOW ING

Testing and Debugging

- Test an application using some sample data
  - Use both valid and invalid data
- **Valid data**: Data that application is expecting
- **Invalid data**: Data that application is not expecting
- **Debugging**: Process of locating and correcting errors
- Errors can be related to either syntax or logic.

KNOW ING

Testing and Debugging

- **Syntax error**: Occurs when a rule of programming language is broken.
- **Logic error**: Occurs when syntax is correct but outcome is not what was desired.
  - Causes may include missing instructions, instructions out of order, or wrong type of instruction.

KNOW ING