Built-in data types:

- Numbers
  - integers, floats, complex numbers, Boolean
- String
- Lists
- Tuple
- Dictionary
- Set
- File Objects

Python provides conditional and iterative control flow

Conditional Control flow:

- An if-elif-else construct
- Exceptions (errors) can be raised using the raise statement and caught and handled using the try-except-else construct.

Iterative control flow:

- while loop
- for loop

Variables don't have to be declared and can have any built-in data type, user defined object, function, or module assigned to them.

Numbers: Integers

Input Statement	Value of i	Data type of i
i=42	42	int
i=-7777777	-77777777	int
i=7e6	7000000.0	float
i=7e60	7e+60	float
Inorder to check the datatype of a variable use type function **type(i)**		

Numbers: Float

Input Statement	Value of f	Data type of f
f=3.0	3.0	float
f=31e12	31000000000000.0	float
f=**-6e-4**	-0.0006	float

Numbers: Complex

Input Statement	Value of C	Data type of C
C=3 + 2j	(3+2j)	complex
C=-4-2j	(-4-2j)	complex
C=4.2 + 6.3j	(4.2+6.3j)	complex

Numbers: Boolean

Input Statement	Value of B	Data type of B
B=True	True	bool
B=False	False	bool
B=bool(1)	True	bool
B=bool(0)	False	bool

Operators on int and float

int		
Operator	Input **(data type)**	Output **(data type)**
+ (Addition)	5+2	7
- (Subtraction)	5-2	3
* (Multiplication)	5*2	10
/ (Division)	5/2	2.5 ***(float)***
// (Integer Division)	5//2	2
**exponentiation	5**2 1000000001 ** 3	25 1000000003000000003000000001
% modulus	5%2	1

float	
Input **(data type)**	Output **(data type)**
3.5e30 * 2.77e45	9.695000000000002e+75
4.3/3.4	1.2647058823529411
4.3//3.4	1.0 (float)
4.3 ** 2.4 1000000001.0 ** 3	33.137847377716483 1.000000003e+27
5.2%2.5	0.20000000000000018

Operators on complex and bool

	complex		bool	
Operator	Input **(data type)**	Output **(data type)**	Input **(data type)**	Output **(data type)**
+ (Addition)	(3+2j) +(4+9j)	(7+11j)	True +True	2
- (Subtraction)	(3+2j)-(4+9j)	(-1-7j)	True-False	1
* (Multiplication)	(3+2j) * (4+9j)	(-6+35j)	True * True	1
/ (Division)	(3+2j) / (4+9j)	*(0.3092783505154639-0.1958762886597938j)*	True / False	**ZeroDivisionError**
// (Integer Division)		**TypeError: can't take floor of complex number.**	True // True	1 (int)
**exponentiation	(3+2j) ** (2+3j)	(0.6817665190890336 3-2.1207457766159625j)	True ** True False ** False	1 1
% modulus		**TypeError: can't mod complex numbers.**	True%False False%True	**ZeroDivisionError** False

Complex methods real and imag:

```
>>> x = (3+2j) * (4+9j)
>>> x                    (complex)
(-6+35j)

>>> x.real               (float)
-6.0

>>> x.imag               (float)
35.0
```

```
>>> round(3.49)          #built-in function, directly used - no import
3

>>> import math
>>> math.ceil(3.49)      #library module function
4
```

cmath library module functions (opeates on complex numbers)	
Input statements	**output**
import cmath cmath.sqrt(3+4j)	(2+1j)
cmath.sin(3+4j)	(3.853738037919377-27.016813258003936j)
Some more math module library functions (operates on int, float and bool numbers)	
import math math.pow(5.2,2.5)	61.66068698936139
math.tan(45)	1.6197751905438615

List

[] **#empty list**
[1]
[1, 2, 3, 4, 5, 6, 7, 8, 12]
[1, "two", 3, 4.0, ["a", "b"], (5,6)]

- A list can contain a mixture of other types as its elements, including strings, tuples, lists, dictionaries, functions, file objects, and any type of number.
- A list can be indexed from its front or back.

Table 3.1 List indices

x=	[	"first" ,	"second" ,	"third" ,	"fourth"	]
Positive indices		0	1	2	3	
Negative indices		−4	−3	−2	−1	

List’s subsegment or slice

slice notation

	x = ["first", "second", "third", "fourth"]	
	Input	Output
Indexing	x[4]	IndexError: list index out of range
	x[-5]	IndexError: list index out of range
	x[-1]	'fourth'
	x[-2]	'third'
Slicing (Finding sublist)	**x[m:n]**	**Extracts the elements of m^{th} index to $(n-1)^{th}$ index.**
	x[0:3]	['first', 'second', 'third']
	x[1:-1]	['second', 'third']
	x[-2:-1]	['third']
	x[:3]	['first', 'second', 'third']
	x[-2:]	['third', 'fourth']

L=[0,1,2,3,4]	Result
L[-1:]	[4]
L[-2:4]	[3]
L[-2:-4] m>n	[]
L[4:2] m>n	[]
L[-4:2]	[1]
L[-5:]	[0, 1, 2, 3, 4]
L[:]	[0, 1, 2, 3, 4]
L[2:-2]	[2]
L[-2:2] m>n	[]
L[3:3] m=n	[]

- m>n is with respect to index position in the array. m>n mean index position of m is later than n^{th} position. The result of L[m:n] when position-wise m>n is [] (empty list).

- Value-wise **2>-2,** but position-wise 2th index appear before -2th index, therefore position-wise ***2<-2***. L[2:-2] returns [2]

List updation

- >>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
- >>> x[1] = "two"
- >>> x[8:9] = []
- >>> x
- [1, 'two', 3, 4, 5, 6, 7, 8]
- >>> x[5:7] = [6.0, 6.5, 7.0]
- >>> x
- [1, 'two', 3, 4, 5, 6.0, 6.5, 7.0, 8]
- >>> x[5:]
- [6.0, 6.5, 7.0, 8]

Type	Example	Use
Built-in functions	len(x)	Returns the number of elements in list x
	max(x)	returns the maximum number in the list x
	min(x)	returns the minimum number in the list x
operators	obj in x	Returns True if object obj is in list x. Otherwise, returns False
	list1+list2	returns a new list and does not modify list1 or list2
	list1*n	Returns list1 repeated by n times. Does not modify list1
Statement	del list1[index]	removes(deletes) the element at the index position in the list

Type	Example	Use
Method	list1.append(object)	appends a single object
	list1.count(value)	Counts the number of occurrences of value in the list1
	list1.reverse()	reverses the list
	list.insert(index,object)	Inserts an object at index position
	list1.index(object)	returns index of an object
	list1.remove(value)	Removes the first occurrence of the value from list
	list1.pop()	Removes the last item from the list
	list1.sort()	Sorts the list having similar kind of objects
	list1.clear()	Makes the list empty

TUPLES

- Tuples are similar to lists but are *immutable*—that is, they can’t be modified after they have been created.

- () empty tuple
- (1,) one element tuple needs a comma. Otherwise it will become simple object
- (1, 2, 3, 4, 5, 6, 7, 8, 12)
- (1, "two", 3, 4.0, ["a", "b"], (5, 6)) may contain elements of any datatype

- x[index] – retrives the indexed element. Accessing element is same as list.

Mutable tuple members are mutable
Ex: a list in tuple is mutable

- >>> t=(1, "two", 3, 4.0, ["a", "b"], (5, 6))
- t[4] which is [‘a’,’b’] is mutable as the list is mutable.
- >>> t[4]
- ['a', 'b']
- >>> t[4][0]='c'
- >>> t
- (1, 'two', 3, 4.0, ['c', 'b'], (5, 6))
- >>> t[4].append('z')
- >>> t
- (1, 'two', 3, 4.0, ['c', 'b', 'z'], (5, 6))

TUPLES

- A list can be converted to a tuple using the built-in function tuple:

```
>>> x = [1, 2, 3, 4]
>>> tuple(x)
(1, 2, 3, 4)
```

- Conversely, a tuple can be converted to a list using the built-in function list:

```
>>> x = (1, 2, 3, 4)

>>> list(x)
[1, 2, 3, 4]
```

STRINGS

- Strings can be delimited by
  - single (' '),
  - double (" "),
  - triple single (''' '''), or
  - triple double (""" """) quotations

  - and can contain tab (\t) and newline (\n) characters.

- Strings are also immutable. The operators and functions that work with them return new strings derived from the original.

STRINGS

- The print function outputs strings. Other Python data types can be easily converted to strings and formatted:

```
>>> e = 2.718
>>> x = [1, "two", 3, 4.0, ["a", "b"], (5, 6)]

>>> print("The constant e is:", e, "and the list x is:", x)
The constant e is: 2.718 and the list x is: [1, 'two', 3, 4.0, ['a', 'b'], (5, 6)]
```

- Objects are automatically converted to string representations for printing.

DICTIONARIES

- A dictionary is mutable
- Dictionaries consist of pairs (called items) of keys and their corresponding values.
- Keys should be unique. Duplicate keys are not stored. Errors are not shown if you enter duplicate key but duplicate key wont get stored.
- Python dictionaries are also known as **associative arrays or hash tables.**

DICTIONARIES

- Keys must be of an immutable type. This includes numbers, strings, and tuples.

- Values can be any kind of object, including mutable types such as lists and dictionaries.

  - **Example:**

  - Var_name={key1:value1,key2:value2,…keyn:valuen}

  - dict={1:”one”,2:”two”,3:”three”}

Type	Example	Does
Built-in function	len(dict)	Returns the number of key-value pairs in dictionary dict
operators	obj in dict	Returns True if object obj is in dict. Otherwise returns False.
statement	del dict[key]	Deletes a key-value pair whose key is given from dictionary dict
Access	dict[key]	Extracts the value of the key given
	dict.get(key,message)	Extracts the value of the key given If key does not exists then returns message(which is second parameter to get method)
modification	dict[key]=value	Alters the value part of the key given.

DICTIONARIES

- The del statement can be used to delete a key-value pair.
- As is the case for lists, a number of dictionary methods (clear, copy, get, has key, items, keys, update, and values) are available.

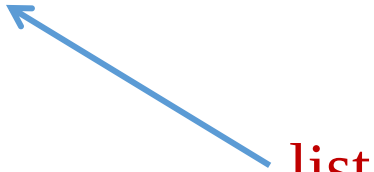
DICTIONARIES

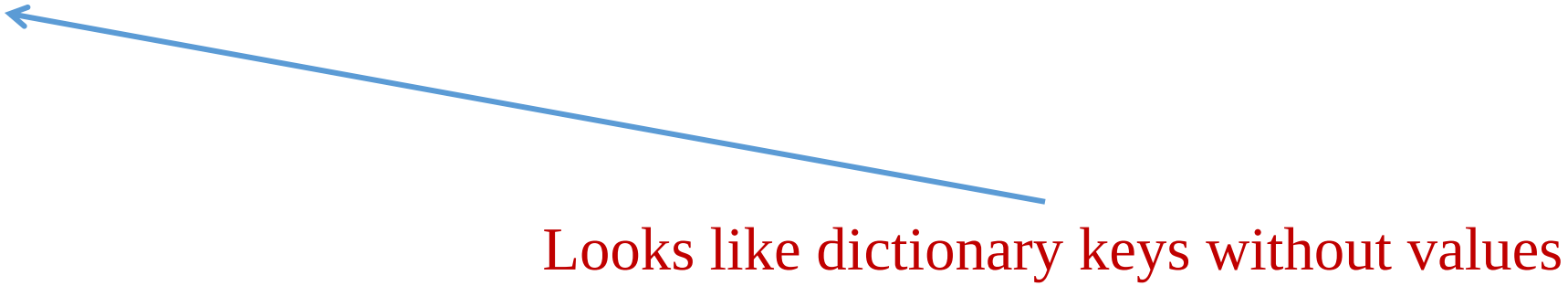
- . This includes numbers, strings, and tuples. Values can be any kind of object, including mutable types such as lists and dictionaries.
- The dictionary method get optionally returns a user-definable value when a key isn’t in a dictionary.

SETS

- An **unordered** collection of objects
- **membership** and **uniqueness** in the set are the main things you need to know about that object.

```
>>> x = set([1, 2, 3, 1, 3, 5])
>>> x
{1, 2, 3, 5}
>>> 1 in x
True
>>> 4 in x
False
```





Control flow structures

Control flow structures

- **Boolean values**

Boolean Values	
False Values	1. False 2. 0 3. the Python nil value **None**, 4. empty values (for example, the empty list [ ] or empty string "")
True Values	1. True 2. 1 3. Other than empty values

Boolean Expressions

comparison operators (<, <=, ==,>, >=, !=, is, is not, in, not in)

and

the logical operators (and, not, or)

return True or False.

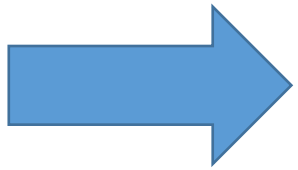
Empty Values are False

- **Empty String** s=‘’ os s=“” or s=‘’’’’’ or s=“”””””””
- **Empty List** l=[]
- **Empty tuple** t=()
- **Empty dictionary** d={}
- **Empty set** s=set()
- **Number**

The if-elif-else statement (Contd..)

```
score=int(input("Enter the score: "))
if score>=90:
    letter='A'
else:   # grade must be B, C, D or F
    if score>=80:
        letter = 'B'
    else:  # grade must be C, D or F
        if score >= 70:
            letter = 'C'
        else:    # grade must D or F
            if score >= 60:
                letter = 'D'
            else:
                letter = 'F'

print("Grade is "+letter)
```



```
score=int(input("Enter the score: "))

if score>=90:
    letter='A'
elif score>=80:
    letter = 'B'
elif score >= 70:
    letter = 'C'
elif score >= 60:
    letter = 'D'
else:
    letter = 'F'

print("Grade is "+letter)
```

While loop

```
count = 0
while count < 9:
    print('The count is:', count)
    count = count + 1

print ("Good bye!”)
```

Output:

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

The for loop

```
x=[0,1,2,3,4,5] #list
for i in x:
    print(i)
```

OR

```
for i in [0,1,2,3,4,5]:
    print(i)
```

Output:

0
1
2
3
4
5

```
for i in 1,5.5,5+8j,True,"str":
    print(i)
```

Output:

1
5.5
(5+8j)
True
str

```
for i in "abc","def","xyz":
    print(i)
```

Output:

abc
def
xyz

```
#for each letter in string
for letter in 'Python':     # First Example
   print('Current Letter :', letter)

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # Second Example
   print('Current fruit :', fruit)
```

Output

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
```

for loop	Output (Numbers are not iterables)
for i in 10: print(i,end=' ')	TypeError: 'int' object is not iterable
for i in 10.75: print(i,end=' ')	TypeError: 'float' object is not iterable
for i in True: print(i,end=' ')	TypeError: ‘bool' object is not iterable
for i in 5+9j: print(i,end=' ')	TypeError: ‘complex' object is not iterable
for i in ‘testing’: print(i,end=' ')	t e s t i n g
for i in [1,2,3,4,5]: print(i,end=' ')	1 2 3 4 5
for i in (11,22,33,44): print(i,end=' ')	11 22 33 44
for i in set([10,20,30]): print(i,end=' ')	10 20 30
for i in {1:'one',2:'two',3:'three'}: print(i,end=' ')	1 2 3

range(stop) # returns range object
range(start,stop[,step]) # returns range object

```
for x in range(0, 3):
    print("We're on time",x)
```

Output

```
We're on time 0
We're on time 1
We're on time 2
```

```
for x in range(1,10,2):
    print(x)
```

Output

```
1
3
5
7
9
```

The range function

Input	Output
list(range(5,10))	[5, 6, 7, 8, 9]
list(range(10))	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
list(range(-10,-5))	[-10, -9, -8, -7, -6]
list(range(5,10,2))	[5, 7, 9]
list(range(5,10,-2))	[]
list(range(5,-5,-1))	[5, 4, 3, 2, 1, 0, -1, -2, -3, -4]
list(range(-10))	[]

Nested for loop

```
for x in range(1, 5):
    for y in range(1, 5):
        print("x=",x," y=",y," x*y=",x*y)
```

Output

```
x= 1  y= 1  x*y= 1
x= 1  y= 2  x*y= 2
x= 1  y= 3  x*y= 3
x= 1  y= 4  x*y= 4
x= 2  y= 1  x*y= 2
x= 2  y= 2  x*y= 4
x= 2  y= 3  x*y= 6
x= 2  y= 4  x*y= 8
x= 3  y= 1  x*y= 3
x= 3  y= 2  x*y= 6
x= 3  y= 3  x*y= 9
x= 3  y= 4  x*y= 12
x= 4  y= 1  x*y= 4
x= 4  y= 2  x*y= 8
x= 4  y= 3  x*y= 12
x= 4  y= 4  x*y= 16
```

Break

```
for x in range(1, 5):
    for y in range(1, 5):
        if x%2==0:
            break;
        print("x=",x," y=",y," x*y=",x*y)
```

Output

```
x= 1  y= 1  x*y= 1
x= 1  y= 2  x*y= 2
x= 1  y= 3  x*y= 3
x= 1  y= 4  x*y= 4
x= 3  y= 1  x*y= 3
x= 3  y= 2  x*y= 6
x= 3  y= 3  x*y= 9
x= 3  y= 4  x*y= 12
```

Continue

```
for num in range(2, 10):
    if num % 2 == 0:
        continue
    print("Found a number", num)
```

Output

```
Found a number 3
Found a number 5
Found a number 7
Found a number 9
```

Generate first ‘n’ terms of Fibonacci series.

- **Solution:**

```
n=int(input("Enter value of n: "))
x=0;y=1;i=1
print(x)     # First Fibonacci Term
while i<n:
    print(y)
    x,y=y,x+y
    i=i+1
```

Find the factorial of ‘n’

- **Solution:**

```
n=int(input("Enter value of n: "))
fact=1
i=1
while i<=n:
    fact=fact*i
    i=i+1
print("Factorial of ",n,"is ",fact)
```

Armstrong number

- The property of Armstrong number is, if the sum of the cubes of the digits of number is same as the given original number, then that number is treated as Armstrong Number.
- The number 153 is regarded as Armstrong number because 1^3 + 5^3 + 3^3 = 153(1+125+27).
- There are six Armstrong numbers in the range of 0 and 999.
  - Armstrong number 1: 0
  - Armstrong number 2: 1
  - Armstrong number 3: 153
  - Armstrong number 4: 370
  - Armstrong number 5: 371
  - Armstrong number 6: 407

Write a Python program to find if the user entered number is Armstrong or not

```
num=int(input("Enter a number: "))
sum=0
temp=num
while temp > 0:
    digit=temp%10
    sum+=digit**3
    temp//=10

if num==sum:
     print(num,"is an Armstrong number")
else:
     print(num,"is not an Armstrong number")
```

Generate the following pattern using nested for loops

1

12

123

1234

12345

123456

1234567

12345678

Generate pattern

****

****

****

******

*****

****

***

**

*

Pattern generation

```
for i in range(1,10):
    for j in range(1,i):
            print(j,end='')     #does not print newline at the end
    print()    # prints a newline
```

```
for i in range(10,1,-1):
    for j in range(1,i):
        print('*',end='')
    print()
```