# s08

June 13, 2024

## 1 fstrings in python

f strings are power ful tools for formating strings. fromat means to put a variable inside a string. let's see some example.

[ ]:
```
name = "mohammad"
age =33
message = "hello my name is ... and I'm ... years old"
print(message)
```

```
hello my name is … and I'm … years old
```

if we want to fill the blanks with variable in blow example we can use diffrent aproach

### 1.1 first approach: string concat

[ ]:
```
# first aproach string concat
name = "mohammad"
age =33
# message = "hello my name is ... and I'm ... years old"
message = "hello my name is "+name+ " and I'm "+str(age)+" years old"
print(message)
```

```
hello my name is mohammad and I'm 33 years old
```

### 1.2 second approach: old style formating (c style)

[ ]:
```
# first aproach string concat
name = "mohammad"
age =33
# message = "hello my name is ... and I'm ... years old"
message = "hello my name is %s and I'm %d years old" % (name,age)
print(message)
```

```
hello my name is mohammad and I'm 33 years old
```

## 1.3 third approach: format method

[ ]:
```
# first aproach string concat
name = "mohammad"
age =33
# message = "hello my name is ... and I'm ... years old"
message = "hello my name is {0} and I'm {1} years old".format(name,age)
print(message)
```

```
hello my name is 33 and I'm mohammad years old
```

## 1.4 4th approach: fstrings (best and newer)

### 1.4.1 simple example

[ ]:
```
# first aproach string concat
name = "mohammad"
age =33
# message = "hello my name is ... and I'm ... years old"
message = f"hello my name is {name} and I'm {age} years old"
print(message)
```

```
hello my name is mohammad and I'm 33 years old
```

### 1.4.2 how to manage field width in fstrings

[ ]:
```
# previous example but with an extera condition.
# name must be at least 15 char. if len of name is less than 15, fill string
# with blank space

name = "mohammad"
age =33
# message = "hello my name is ... and I'm ... years old"
message = f"hello my name is {name:15} and I'm {age} years old"
print(message)
```

```
hello my name is mohammad        and I'm 33 years old
```

[ ]:
```
name = "mohammad"
age =33
message = f"hello my name is {name:>15} and I'm {age} years old"
print(message)
```

```
hello my name is        mohammad and I'm 33 years old
```

[ ]:
```
name = "mohammad"
age =33
message = f"hello my name is {name:^15} and I'm {age} years old"
print(message)
```

```
hello my name is    mohammad     and I'm 33 years old
```

[ ]:
```
name = "mohammad"
age =33
message = f"hello my name is {name:*^15} and I'm {age} years old"
print(message)
```

```
hello my name is ***mohammad**** and I'm 33 years old
```

### 1.4.3 field width for numbers

[ ]:
```
name = "mohammad"
money =10.357951
message = f"hello my name is {name} and I have {money}$ money"
print(message)
```

```
hello my name is mohammad and I have 10.357951$ money
```

[ ]:
```
name = "mohammad"
money =10.357951
message = f"hello my name is {name} and I have {money:10.2f}$ money"
print(message)
```

```
hello my name is mohammad and I have      10.36$ money
```

[ ]:
```
name = "mohammad"
money =10.357951
message = f"hello my name is {name} and I have {money:010.2f}$ money"
print(message)
```

```
hello my name is mohammad and I have 0000010.36$ money
```

[ ]:
```
name = "mohammad"
money =10.357951
message = f"hello my name is {name} and I have {money:.2f}$ money"
print(message)
```

```
hello my name is mohammad and I have 10.36$ money
```

[ ]:
```
a = 10
b = 20
message = f"calculation: {a+b}"
print(message)
```

```
calculation: 30
```

[ ]:
```
a = 10
b = 20
message = f"calculation: {a+b=}"
print(message)
```

```
calculation: a+b=30
```

[ ]:
```
a = 1044987894564684984641
message = f"big number: {a:_}"
print(message)
```

```
big number: 1_044_987_894_564_684_984_641
```

[ ]:
```
a = 1044987894564684984641
message = f"big number: {a:,}"
print(message)
```

```
big number: 1,044,987,894,564,684,984,641
```

## 2 list comprehension

shorter syntax for creation of list in loop

[ ]:
```
a = [2,3,5]
# assume we want to create a new list like b, each element of b is coresponding
# element in a *2
b = []
for ai in a:
    b.append(ai*2)

print(a)
print(b)
# instead of previous method, we can use list comprehension
c = [ai*2 for ai in a]
print(c)
```

```
[2, 3, 5]
[4, 6, 10]
[4, 6, 10]
```

[ ]:
```
a = [2,3,5,4,7,10,9]
# assume we want to only even element in a. we can use filter clause in list
# comprehension

b = []
for ai in a:
    if ai%2==0:
        b.append(ai)

print(a)
print(b)
# instead of previous method, we can use list comprehension with filter clause
c = [ai for ai in a if ai%2==0]
print(c)
```

```
[2, 3, 5, 4, 7, 10, 9]
[2, 4, 10]
[2, 4, 10]
```

[ ]:
```python
a = [2,3,5,4,7,10,9]
# assume we want to map a function and use a filter clause in list
# comprehension

def calc(x):
    return x**2+1

b = []
for ai in a:
    if ai%2==0:
        b.append(calc(ai))

print(a)
print(b)
# instead of previous method, we can use list comprehension with filter clause
c = [calc(ai) for ai in a if ai%2==0]
print(c)
```

```
[2, 3, 5, 4, 7, 10, 9]
[5, 17, 101]
[5, 17, 101]
```

### 2.0.1 ternary operator (? in c language)

in c language we can use ternary operator instead of if. for example code`e = a>b?10:20` is equal to `if (a>b){ e=10 }else{ e=20 }`

[ ]:
```python
# but in python we use if for ternary operator
e = 10 if a>b else 20
# is equal to
if a>b:
    e=10
else:
    e=20
```

## 2.1 how to use ternary operator in list comprehension

[ ]:
```python
a = [2,3,5,4,7,10,9]
# assume we want to create a new list based on a. if ai is even we must return
# true in new list else we must return false

b = []
for ai in a:
    if ai%2==0:
```

```
        x=True
    else:
        x=False

    b.append(x)

print(a)
print(b)
# instead of previous method, we can use list comprehension with filter clause
c = [True if ai%2==0 else False  for ai in a ]
print(c)
```

```
[2, 3, 5, 4, 7, 10, 9]
[True, False, False, True, False, True, False]
[True, False, False, True, False, True, False]
```

# 3 functional programming with map filter reduce

## 3.1 map

assume we have a iterable and want to call a function on all of it's elements. in this case we can use a loop or we can use list comprehension of finaly we can use map builtin function

[ ]:
```
def calc(x):
    return x**2+1

a = [2,3,5,7,14,11,1,4]

#first method:use a loop
b=[]
for ai in a:
    b.append(calc(ai))
print(f"first method, only use a loop.\n{b=}")

# second method: use list comprehension
c = [calc(ai) for ai in a]
print(f"second method, list comprehension.\n{c=}")

# third method: use map function
d = map(calc,a)
print(f"third method,map function.\n{d=}")
# result of map function is a map object that can be iterate in for loop or
# can be casted to a list
print(f"third method,map function.\n{list(d)=}")
```

```
first method, only use a loop.
b=[5, 10, 26, 50, 197, 122, 2, 17]
second method, list comprehension.
```

```
c=[5, 10, 26, 50, 197, 122, 2, 17]
third method,map function.
d=<map object at 0x000001E9C1667220>
third method,map function.
list(d)=[5, 10, 26, 50, 197, 122, 2, 17]
```

[ ]:
```
# we usualy use map with lambda expression
# for example we can use this code
a = [2,3,5,7,14,11,1,4]
print(f"map function with lambda.\n{list(map(lambda x:x**2+1,a))=}")
```

```
map function with lambda.
list(map(lambda x:x**2+1,a))=[5, 10, 26, 50, 197, 122, 2, 17]
```

### 3.1.1 input using map

we can read inputs and cast them to seperate variable using map and split

[ ]:
```
# assume we have 3 float input in one line
# 2 3.8 4.5

a,b,c = map(float,input().split())

# also we can use list comprehension for that purpose
a,b,c = [float(x) for x in input().split()]
```

## 3.2 filter

assume we have a iterable and want to filter it's element based on a filter function result. in this case we use filter builtin function.

[ ]:
```
a = [2,3,5,7,14,11,1,4]
# assume we want only even elements in list a

#first method:use a loop
b=[]
for ai in a:
    if ai%2==0:
        b.append(ai)
print(f"first method, only use a loop.\n{b=}")

# second method: use list comprehension
c = [ai for ai in a if ai%2==0]
print(f"second method, list comprehension.\n{c=}")

# third method: use filter function
def f(x):
    if x%2==0:
        return True
```

```
    return False

d = filter(f,a)
print(f"third method,filter function.\n{d=}")
# result of filter function is a filter object that can be iterate in for loop␣
 ↪or
# can be casted to a list
print(f"third method,filter function.\n{list(d)=}")
```

```
first method, only use a loop.
b=[2, 14, 4]
second method, list comprehension.
c=[2, 14, 4]
third method,filter function.
d=<filter object at 0x000001E9C29F3850>
third method,map function.
list(d)=[2, 14, 4]
```

[ ]:
```
# we usualy use filter with lambda expression
# for example we can use this code
a = [2,3,5,7,14,11,1,4]
print(f"filter function with lambda.\n{list(filter(lambda x:x%2==0,a))=}")
```

```
filter function with lambda.
list(filter(lambda x:x%2==0,a))=[2, 14, 4]
```

### 3.3 reduce

we used to use reduce for finding min,max,sum,any,all ,... but in newer versin of python we can use builtin function for that purpose.

[ ]:
```
from functools import reduce

a = [2,3,5,7,14,11,1,4]
# assume we want to find minimum of a
print(f"min of a: {reduce(lambda x,y:x if x<y else y,a)=}")

# assume we want to find maximum of a
print(f"max of a: {reduce(lambda x,y:x if x>y else y,a)=}")

# assume we want to find sum of a
print(f"sum of a: {reduce(lambda x,y:x+y,a)=}")

# assume we want to find product of a
print(f"sum of a: {reduce(lambda x,y:x*y,a)=}")

# assume we want to find any of a
print(f"any of a: {reduce(lambda x,y:bool(x) or bool(y),a)=}")
```

```python
# assume we want to find all of a
print(f"any of a: {reduce(lambda x,y:bool(x) and bool(y),a)=}")

# also we can use min, max, sum, any,all builtin function
print(min(a))
print(max(a))
print(sum(a))
print(any(a))
print(all(a))
```

```
min of a: reduce(lambda x,y:x if x<y else y,a)=1
max of a: reduce(lambda x,y:x if x>y else y,a)=14
sum of a: reduce(lambda x,y:x+y,a)=47
sum of a: reduce(lambda x,y:x*y,a)=129360
any of a: reduce(lambda x,y:bool(x) or bool(y),a)=True
any of a: reduce(lambda x,y:bool(x) and bool(y),a)=True
1
14
47
True
True
```

# 4 Exceptions

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called exceptions and are not unconditionally fatal

[ ]:
```python
# example of exception: zero division exception

a = int(input("enter a: "))
b = int(input("enter b: "))
print(a/b)
# if user enter b as zero this exception will raise

# Traceback (most recent call last):
#   File "d:\code\maktab116\s08\s08a.py", line 3, in <module>
#     print(a/b)
# ZeroDivisionError: division by zero
```

exceptions in python are builtin

## 4.1 how can we handle exceptions?

we can use try block in pyhton to handle exceptions.

[ ]:
```
a = int(input("enter a: "))
b = int(input("enter b: "))
try:
    print(a/b)
except Exception as e:
    print("something wen't wrong")
```

[ ]:
```
a = int(input("enter a: "))
b = int(input("enter b: "))
try:
    print(a/b)
except ZeroDivisionError:
    print("b can not be zero!")
    b = int(input("please enter b again: "))
    print(a/b)
except Exception as e:
    print("something wen't wrong")
finally:
    pass
    #do somthing
```

# 5 file handling in python

we can use open() built in function to manage files in python.

### 5.0.1 we can use relative of absolute address for files

[ ]:
```
# relative address example
file_address = "myData\students.txt"
fp = open(file_address,'w')
fp.write("salam")
fp.close()
```

[ ]:
```
# relative address example
file_address = "D:\code\maktab116\s08\myData\students.txt"
fp = open(file_address,'w')
fp.write("absolute")
fp.close()
```

### 5.0.2 write mode

in this mode, all previous data will clean and new data will be write to file

[ ]:
```
file_address = "data.txt"
fp = open(file_address,'w')
fp.write("write mode")
fp.close()
```

### 5.0.3 read mode

in this mode, we can't change data in file only can read data. if we want to write data in read mode, an exception will raise

[ ]:
```
file_address = "data.txt"
fp = open(file_address,'r')
fp.write("write mode")
fp.close()
```

```
---------------------------------------------------------------------------
UnsupportedOperation                                 Traceback (most recent call last)
Input In [10], in <cell line: 3>()
      1 file_address = "data.txt"
      2 fp = open(file_address,'r')
----> 3 fp.write("write mode")
      4 fp.close()

UnsupportedOperation: not writable
```

to read all content of text file we can use read() method.

[ ]:
```
file_address = "data.txt"
fp = open(file_address,'r')
data = fp.read()
print(data)
fp.close()
```

```
write mode
```

readline only read a single line and put the cursor at the begining of next line

[ ]:
```
file_address = "data.txt"
fp = open(file_address,'r')
data = fp.readline()
print(data)
fp.close()
```

```
line 1
```

[ ]:
```
file_address = "data.txt"
fp = open(file_address,'r')
while True:
    data = fp.readline()
    if not data:
        break
    print(data)
fp.close()
```

```
line 1

line 2

line 3
```

[ ]:
```
# we can use readlines method to get all lines in a iterable:

file_address = "data.txt"
fp = open(file_address,'r')
for l in fp.readlines():
    print(l)
fp.close()
```

```
line 1

line 2

line 3
```

#### 5.0.4 append mode

in this mode, we can save previous data and append new data to file

[ ]:
```
file_address = "data.txt"
fp = open(file_address,'a')
fp.write("salam")
fp.write("bye")
fp.close()
```

### 5.1 how to use contex manager for files

[ ]:
```
file_address = "data.txt"
with open(file_address,'a') as fp:
    fp.write("salam")
    fp.write("bye")
```

### 5.2 encoding in file

if we want to write persian char in files we must set encoding parameter for open function

[ ]:
```
file_address = "data.txt"
with open(file_address,'a',encoding='utf8') as fp:
    fp.write("  ")
    fp.write("    ")
```

### 5.3 open file in byte mode

[ ]:
```
file_address = "myData\profile.png"
with open(file_address,'rb',) as fp:
    data =fp.read()
    print(bin(data[170]))
    print(len(data))
    print(type(data))
```

```
0b10100000
5040
<class 'bytes'>
```