به نام خدا

پایتون \_

مقدمه \_مهندسی نرم افزار:

داده: مجموعه ای از مقادیر در مورد یک موضوع یا شی که به صورت کمی با یک مقدار عددی و با به صورت کیفی نشان داده می شود.

اطلاعات: برای اینکه از داده ها بتوانیم بهتر استفاده کنیم لازم است بر روی آنها عملیات را انجام دهیم. نتایج حاصل از این عملیات را اطلاعات می نامیم.

دانش: اطلاعات می تواند مورد تفسیر و بررسی قرار گیرد و نتیجه بررسی آن ها به دانش می تواند مبنای تصمیم گیری برای انجام کاری شود.

پردازش: مجموعه محاسبات و عملیات که بر روی داده ها صورت می گیرد را پردازش می نامند.

Data 🡪 Process 🡪 Information

است.CPU قلب هر کامپیوتر یک

Central Process Unit : CPU

: CPU اجزای

Arithmetic Logic Unit : ALU

Control Unit : CU

: Memory Unit

CACHE , Registers

مراحل اجرای دستور :

I – Fetch (Data & Instruction)

II – Decode (Instruction)

III – Execute (Commands)

IV – Store (Result)

زبان ماشین :

زبان قابل فهم سخت افزار کامپیوتر – زبان ماشین – نام دارد و متشکل از دنباله ای از کد های

0 و 1 است.

مترجم :

برنامه ای که با زبانی غیر از زبان ماشین نوشته شود ابتدا باید توسط یک نورم افزار مترجم تبدیل به زبان ماشین شود و سپس برنامه ترجمه شده توسط کامپیوتر اجرا گردد.

زبان های برنامه سازی :

زبان های سطح پایین : زبان هایی که به زبان پردازشگر کامپیوتر نزدیک باشد و از زبان محاوره ای ما دور هستند.

🡪مانند زبان ماشین و زبان اسمبلی

زبان های سطح میانی : زبان هایی در این دسته قرار می گیرند که دستوراتی برای دسترسی راحت تر به سخت افزار پیش بینی شده باشد. و همچنین به زبان عامیانه نزدیک باشد.

🡪مانند زبان سی

زبان های سطح بالا : زبان برنامه نویسی که به زبان محاوره ای ما نزدیک باشد.

🡪مانند پایتون و جاوا

برخی از سازنده های زبان های برنامه نویسی

آدا : زبان آدا – 1979

دنیس ریچی : زبان سی – 1972

استراس تروپ : سی پلاس پلاس – 1983

رسوم : پایتون – 1989

مقدمه\_ برنامه نویسی

\_الگوریتم

الخارزمی : دانشمند ایرانی که اصطلاح امروزی الگوریتم از نام ایشان گرفته شده است.

الگوریتم : مجموعه دستورالعمل ها که اجرای آن ها با ترتیب معین شده منجر به انجام حل یک مساله میشود.

مثال )

1. شروع
2. عدد الف و ب را دریافت کن
3. عدد الف و ب را با هم جمع کن
4. حاصل جمع را نمایش بده
5. پایان

فلوچارت

مجموعه ای از علائم ساده که الگوریتم را به صورت نماد های تصویری یا نموداری تبدیل می کند.

علائم اصلی فلوچارت

مستطیل : پردازش بیضی : شروع و پایان متوازی الاضلاع : ورودی و خروج

علائم شرطی و تکرار

شرط : لوزی لوزی متسطیلی : حلقه

**پایتون**

­\_ انواع داده ها

متغیر ها می توانند داده های مختلف را ذخیره کنند و انواع مختلف می توانند کار های متفاوتی انجام دهند

Int عدد کامل 23, 12, -456

Str رشته ‘hello, taha’

Float عدد اعشاری 12.5, 0.75, 12.0

Bool منطفی True, False

: یک عدد کامل مثبت یا منفی بدون اعشار با طول نامحدود int

متغیر ها را بگیریم – خروجی به ما با توجه به نوع متغیر به ما می گوید Typeاگر

<class, ‘int’>

: یک عدد مثبت یا منفی است که حاوی یک یا چند اعشار است. Float

\_عملگر ها

عملگر ها ریاضی :  
+ : جمع - : تفریق \* : ضرب / : تقسیم % : باقی مانده تقسیم صحیح \*\* : توان // : تقسیم صحیح

توجه : عملگر های ریاضی با توجه به علویت های انها اجرا میشود

عملگر های انتساب :

= مساوی =- از یک متغیر مقداری کم شود =+ به یک متغیر مقداری اضافه شود

=/ =%

عملگر مقایسه :

== برابر

=! نابرابر  
> < بزرگتر یا کوچکتر

<= >=بزرگتر مساوی – کوچکتر مساوی

عملگر منطقی :

: باید دو طرف صحیح باشد تا جواب صحیح باشد And

: باید دو طرف ناصحیح باشد تا جواب ناصحیح باشدOr

: اگر صحیح باشد ناصحیح میشود و عکس این دقیقاnot

عملگر عضویت :

: برای معلوم کردن وجود یک متغیر در یک مجموعه in

: جوابی مخالف بالا بازمیگرداند not in

نکته : از کلمات کلیدی خود پایتون نمی توان به عنوان اسم تابع – متغیر و هر چیز دیگری استفاده کرد

برای مشخص کردن اینکه آیا اسمی رزرو شده برای پایتون هست یا نه !؟ می توان کد زیر را استفاده کرد

Iskeyword(‘for’) 🡪 True

\_رشته

به مجموعه کاراکتر های هم جوار که در علامت نقل قول نمایش داده میشود

اندیس🡪

0 1 2 3 4 5

P y t h o n

S = ‘python’

Print(s) 🡪 python

Print(s[0]) 🡪 p

Print(s[3]) 🡪 h

برش با عملگر [:]

S = “python”

Print(s[1:3]) 🡪 yt

Print(s[0:6]) 🡪 python

Print(s[:6]) 🡪 python

Print(s[0:]) 🡪 python

Print(s[:]) 🡪 python

برش + تعداد گام

s = ‘python’

print(s[0:6:1]) 🡪 python

print(s[0:6:2]) 🡪 pto

print(s[0:6:5]) 🡪 pn

رشته با اندیس منفی

-6 -5 -4 -3 -2 -1

P y t h o n

S = ‘python’

Print(s) 🡪 python

Print(s[-1]) 🡪 n

Print(s[-5]) 🡪 y

Print(s[-6]) 🡪 p

مثال)

S = ‘python’

Print(s[-6:-1]) 🡪 pytho

Print(s[-3:-2]) 🡪 h

Print(s[-1:-7:-1]) 🡪nohtyp

Print(s[0:6:-1]) 🡪 nohtyp

Print(s[::-1]) 🡪 nohtyp

Print(s[-1:-7:-2]) 🡪 nhy

Print(s[-6:-1:2]) 🡪 pto

S = ‘python’

Print(s[::-1][0]) 🡪 n

ابتدا رشته را معکوس کن – سپس اندیس 0 را برگردان

Print(s[::-1][::-1]) 🡪 python

اتصال رشته با عملگر جمع

S1 = ‘ali’

S2 = ‘reza’

S3 = s1 + s2

Print(s3) 🡪 alireza

ضرب رشته ها

S = ‘sara’

Print(s \* 2) 🡪 sarasara

Print(s \* 3) 🡪 sarasarasara

A = ‘ali’

B = ‘reza’

C = 2 \* (a + b)

Print(c) 🡪 alirezaalireza

Print(‘zaa’ in c) 🡪 True

مقایسه رشته

S1 = ‘Hello Python’

S2 = “Hello” + ‘ Python’

Print(s1 == s2) 🡪 True

Print(s1 is s2) 🡪 False

Len()

S = ‘python’

Print(len(s)) 🡪 6

Print(s[len(s)-1]) = print(s[5]) 🡪 n

Ord, chr

Print(ord(‘a’)) 🡪 97

Print(ord(‘A’)) 🡪 65

هر کاراکتر یک کد اصلی دارد – با این تابع می توان کاراکتر را داد و کد اصلی را گرفت

Print(chr(65)) 🡪 A

مخالف تابع بالا است – کد اصلی را میگیرد و کاراکتر را می دهد

\_متد کار با رشته

Format\_

Print(‘I love {0} & {1}’.format(‘apple’, ‘banana’))

Print(‘I love {1} & {0}’.format(‘apple’, ‘banana’))

1. 1

Y, x = ‘apple’, ‘banana’

Print(f’I love {y} & {x}’)

S = ‘taha’

Print(‘name : %s’ % s)

S = ‘ali’

Print(f‘{s:\*>8}’) 🡪 \*\*\*\*\*ali

Print(f’{s:\*<8}’) 🡪 ali\*\*\*\*\*

Print(f’{s:\*^8}’) 🡪 \*\*ali\*\*\*

Ljust, rjust\_

Name = ‘taha’

Print(name.rjust(10)) 🡪 ‘ taha’

Print(name.ljust(10)) 🡪 ‘taha ‘

Print(name.rjust(10, ‘-’)) 🡪 ‘------taha’

Print(name.ljust(10, ‘-’)) 🡪 ‘taha------’

Zfill\_

Txt = ‘13’

Print(txt.zfill(7)) 🡪 0000013

Lower, upper, capitalize\_

Txt = ‘hello, Welcome to Python’

Print(txt.lower()) 🡪 hello, welcome to python

Print(txt.upper()) 🡪 HELLO, WELCOMEE TO PYTHON

Print(txt.captitalize) 🡪 Hello, welcome to python

Title\_

S = ‘dark knight’

Print(s.title()) 🡪 Dark Knight

Islower, isupper, istitle\_

lang = ‘python’

print(lang.islower()) 🡪 True

print(lang.isupper()) 🡪 False

print(lang.istitle()) 🡪 False

swapcase\_

s = ‘TaHa’

print(s.swapcase()) 🡪 tAhA

translate\_

s = ‘sara’

print(s.translate({97:100}))

a d

t = { 114 : 104, 46 : None} <class, ‘tuple’>

r h . \_

s = ‘tara.’

Print(s.translate(t)) 🡪 taha

Startwith, endwith\_

S = ‘this is string example’

Print(s.startwith(‘this’)) 🡪 True

Print(s.endwith(‘example’)) 🡪 True

Print(s.startwith(‘example’)) 🡪 False

count\_

txt = ‘I love apples, apple are my favorite fruite’

x = txt.count(‘apple’)

print(x) 🡪 2

print(‘$10 $20 $30’.count(‘$’)) 🡪 3

find\_

0 1 2 3 4 5 6 7 8 9

T a h a t a j i k

name = ‘Taha tajik’

print(name.find(‘T’)) 🡪 0

print(name.find(‘t’)) 🡪 5

print(name.find(‘h’)) 🡪 2

print(name.find(‘x’)) 🡪 -1

rfind\_

txt = ‘Hello, welcome to python’

print(txt.lfind(‘e’)) 🡪 13

replace\_

fruit = ‘Strawberry’

print(fruit.replace(‘r’, ‘R’))

join\_

x = ‘ali’

y = ‘reza’

z = ‘-’.join((x, y))

i = ‘ ’.join((x, y))

j = ‘\n’.join((x, y))

print(z) 🡪 ali-reza

print(i) 🡪 ali reza

print(j) 🡪 ali

reza

\n 🡪 new line

\t 🡪 Tab

Strip\_

S = ‘..+.ali..-...’

s.strip(‘.’) 🡪 +..ali..-

s = ‘ ali ‘

s.strip() 🡪 ‘ali’

rstrip, lstrip\_

s = ‘—ali---’

print(s.lstrip(‘-’)) 🡪 ali---

print(s.rstrip(‘-’)) 🡪 --ali

split\_

name = ‘Taha Tajik’

print(name.split()) 🡪 [‘Taha’, ‘Tajik’]

s = ‘taha@gmail.com’

print(s.split(‘@’)) 🡪 [‘taha’, ‘gmail.com’]

s = ‘The world is Beautiful’

print(s.split(‘i’)) 🡪 [‘The world ’, ‘s Beaut’, ‘ful’]

rsplit\_

s = ‘C, C++, R, Python, Java’

print(s.split(‘,’, 3)) 🡪 [‘C’, ‘C++’, ‘R’, ‘Python, Java’]

print(s.rsplit(‘,’, 3)) 🡪 [‘C, C++’, ‘R’, ‘Python’, ‘Java’]

splitlines\_

txt = ‘hello\npython’

x = txt.splitlines()

print(x) 🡪 [‘hello’, ‘python’]

partition\_

txt = ‘I eat oranges, oranges are my favorite fruit.’

print(txt.partition(‘oranges’)) 🡪 [‘I eat’, ‘oranges’, ‘oranges are my favorite fruit.’]

print(txt.rpartition(‘oranges’)) 🡪 [‘I eat oranges’, ‘oranges’, ‘are my favorite fruit.’]

isalpha, isalnum

s = ‘ali’

print(s.isalpha()) 🡪 True

s = ‘ali3’

print(s.isalpha()) 🡪 False

s = ‘ali110’

print(s.isalnum()) 🡪 True

s = ‘ali 110’

print(s.isalnum()) 🡪 False

\_لیست

لیست- تاپل- دیکشنری جزو ساختمان داده های در پایتون هستند

ساختمان داده ها جمعی از چندین داده درکنار هم هستند

x = [‘apple’, ‘banana’, ‘cherry’]

<class ‘list’>

اندیس 🡪 0 1 2

Apple banana cherry

-3 -2 -1

lst = [5, -3, 12]

print(lst) 🡪 [5, -3, 12]

print(lst[0]) 🡪 5

print(lst[2]) 🡪 12

جایگزینی

lst = [‘spider’, ‘frog’, ‘bat’]

lst[2] = ‘snail’

print(lst) 🡪 [‘spider’, ‘frog’, ‘snail’]

lst = [‘spider’, ‘frog’, ‘bat’]

lst[1:3] = [‘cow’, ‘danky’]

print(lst) 🡪 [‘spider’, ‘cow’, ‘danky’]

a = [1, 2, 3]

b = [1, 2, 3]

print(a == b) 🡪 True

print(a is b) 🡪 False

a = [5, 6, 7]

b = a

print(a == b) 🡪 True

print(a is b) 🡪 True

برش slicing

lst = [15, 17, 19, 13, 18]

print(lst) 🡪 [15, 17, 19, 13, 18]

print(lst[:]) 🡪 [15, 17, 19, 13, 18]

print(lst[0:]) 🡪 [15, 17, 19, 13, 18]

print(lst[0:3]) 🡪 [15, 17, 19]

print(lst[:3]) 🡪 [15, 17, 19]

معکوس کردن لیست

Ls = [1, 2, 3, 4]

Print(ls[::-1]) 🡪 [4, 3, 2, 1]

Print(ls[::-2]) 🡪 [4, 2]

اتصال لیست با عملگر +

Lst\_1 = [‘Taha’, 1389]

Lst\_2 = [‘ali’, 1390]

Lst = lst\_1 + lst\_2

Print(lst) 🡪 [‘Taha’, 1389, ‘ali’, 1390]

طول لیست

Lst = [‘Taha’, ‘Tajik’]

Print(len(lst)) 🡪 2

\_متد های کار با لیست

Clear\_

x = [‘apple’, ‘banana’, ‘cherry’]

x.clear()

print(x) 🡪 []

x = [‘apple’, ‘banana’, ‘cherry’]

del x

print(x) 🡪 TypeErore

index\_

name = [‘ali’, ‘sara’, ‘farshid’]

I = name.index(‘sara’)

Print(i) 🡪 1

Count\_

x = [‘apple’, ‘banana’, ‘cherry’]

c = x.count(‘banana’)

print(c) 🡪 1

score = [11, 13, 12, 19, 17, 8, 20, 13, 6]

c = score.count(13)

print(c) 🡪 2

remove\_

x = [‘apple’, ‘banana’, ‘cherry’]

x.remove(‘banana’)

print(x) 🡪 [‘apple’, ‘cherry’]

pop\_

x = [‘apple’, ‘banana’, ‘cherry’]

x.pop(1)

print(x) 🡪 [‘apple’, ‘cherry’]

insert\_

x = [‘apple’, ‘banana’, ‘cherry’]

x.insert(2, ‘watermelon’)

print(x) 🡪 [‘apple’, ‘banana’, ‘watermelon’, ‘cherry’]

append\_

lst = [‘spider’, ‘frog’, ‘bat’]

lst.append(‘bear’)

print(lst) 🡪 [‘spider’, ‘frog’, ‘bat’, ‘bear’]

extend\_

lst = [‘ali’, ‘taha’, ‘sara’]

lst.extend([‘farid’, ‘maryam’])

print(lst)

reverse\_

lst = [13, 16, 18, 16, 20]

lst.reverse()

print(lst) 🡪 [20, 16, 18, 16, 13]

sort\_

lst = [13, 16, 18, 15, 20]

lst.sort()

print(lst) 🡪 [13, 15, 16, 18, 20]

lst = [13, 16, 18, 16, 20]

lst.sort(reverse = True)

print(lst) 🡪 [20, 18, 16, 15, 13]

copy\_

x = [‘apple’, ‘banana’, ‘cherry’]

y = x.copy()

print(y) 🡪 [‘apple’, ‘banana’, ‘cherry’]

join\_

x = [‘C’, ‘Python’, ‘Java’]

y = ‘-’.join(x)

print(y) 🡪 C-Python-Java

توابع sum, max

X = [1, 2, 3, 4, 5]

S = sum(x)

Print(s) 🡪 15

X = [15, 10, 12, 20, 13, 18]

لیست متداخل

X = [11, [12, 13, [4, ‘ali’, 15]], ‘sara’]

Print(x[0]) 🡪 11

Print(x[1]) 🡪 [12, 13, [4, ‘ali’, 15]]

Print(x[2]) 🡪 sara

Print(x[-1]) 🡪 sara

\_دستور های شرطی

برای اگر چیزی برقرار بود – استفاده میشود. یعنی اگر در قسمت بالا شرط درست بود بلوک زیر شرط اجرا میشود

If

if condition :

if\_block

مثال

Age = 25

If age > 20 :

Print(‘Ur too old!’)

Output🡪 Ur too old!

مثال

Txt = ‘python is a programming language.’

if ‘programming’ in txt :

print(‘yes’)

output🡪 yes

If, else

If condition :

If\_block

Else :

Else\_block

مثال

X = False

Y = True

If x and y :

Print(‘Both x and y are True’)

Else :

Print(‘x is False or y is false or both x and y are false’)

مثال

if a > b :

print(‘>’)

else :

print(‘<’)

معادل

Print(‘>’) if a > b else print(‘<’)

دستور شرطی تو در تو

X = int(input())

If x > 0 :

If x < 20 :

Print(‘0<x<20’)

Else :

Print(‘20=<x’)

Else :

Print(‘x<=0’)

elif

if <expr> :

<statement(s)>

Elif <expr> :

<statement(s)>

Elif <expr> :

<statement(s)>

Else :

<statement(s)>

مثال

If (type(x) == int) :

Print(‘Integer’)

If (type(x) == float) :

Print(‘Float’)

If (type(x) == bool) :

Print(‘Bool’)

If (type(x) == str) :

Print(‘Strign’)

If (type(x) == list) :

Print(‘List’)

\_دستور های تکرار

While condition :

Block

مثال

N = 1

While n <= 3 :

Print(n)

N += 1

F = False

While not f :

X = int(input())

If x == 13 :

F = True

Print(x \* 2)

For i in range(1, 4) :

Print(i)

تعداد گام

For j in range(1, 6, 2) :

Print(i)

چاپ محتویات لیست

X = [5, 2, 9]

For I in x :

Print(i) 🡪 5, 2, 9

For I in range(len(x)) :

Print(x[i]) 🡪 5, 2, 9

Lst = []

For I in range(0, 3) :

N = int(input())

Lst += [n]

For item in lst :

Print(item, end=’ ’) 🡪 5 8 20

End= ‘\n’ 🡪 new line

\t 🡪 tab

Nums = [1, 2, 3, 4]

For I in range(len(nums)-1, -1, -1) :

Print(nums[i]) 🡪 4 3 2 1

Continue, break

While <expr> :

<statement>

<statement>

Break

<statement> X Doesn’t run

<statement> X Doesn’t run

<statement>

این دستور زمانی که در بلاک یک دستور تکرار بکار می رود. دستورات زیر ان دیگر اجرا نمیشوند و برنامه از حلقه خارج میشود. و دستورات دیگری که زیر کل ساختمان حلقه قرار دارد را اجرا می کند

While <expr> :

<statement>

<statement>

continue

<statement> X Doesn’t run

<statement> X Doesn’t run

<statement>

این دستور زمانی که در بلاک یک دستور تکرار بکار برود. دستورات زیر آن اجرا نمیشود و دوباره برنامه بر می گردد به اول حلقه ایکه در ان بود. و زمانی که خود حلقه به پایان رسید – به طور طبیعی از حلقه خارج میشود و دستورات زیر کل ساختمان تکرار اجرا میشود.

مثال

For I in range(1, 7) :

If I == 5 :

Break

Print(i)

Output🡪

1 2 3 4

For I in range(1, 7) :

If I == 5 :

Continue

Print(i)

Output🡪

1 2 3 4 6

حلقه تو در تو

For row in range(1, 4) :

For col in range(1, 4) :

Print(row \* col, end=’\t’)

Print()

Row col

1 1, 2, 3

2 1, 2, 3

3 1, 2, 3

Output🡪

1 2 3

2 4 6

3 6 9

حلقه وابسته

For I in range(1, 4) :

For j in range(0, i) :

Print(I, end=’\t’)

Print()

Output🡪

1

2 2

3 3 3

(Tuple)\_تاپل

مروری بر دیتا تایپ های ساختمان داده در پایتون :

...لیست

...تاپل

...مجموعه

...دیکشنری

از تاپل برای ذخیره چند آیتم در یک تغیر به کار می رود. مانند لیست – با این تفاوت که به جای 2 بریک از پرانتز استفاده می کنیم.

t = (18, 15, 19)

print(t) 🡪 (18, 15, 19)

اندیس 🡪 0 1 2

(18, 15, 19)

-3 -2 -1

Print(t[2]) 🡪 19

Print(t[-1]) 🡪 19

تفاوت با لیست

T = (18, 15, 19)

T[0] = 20

Output 🡪

TypeError: ‘tuple’ obj doesn’t support item assignment

شما نمی توانید مانند لست – در تاپل به غیر از مقدار دهی اولیه در هنگام برنامه تغیری در آن ایجاد کنید!

\_متد کار با تاپل

index

t = (‘C++’, ‘C’, ‘Python’, ‘Java’)

print(t.index(‘Python’)) 🡪 2

count

t = (‘C++’, ‘C’, ‘Python’, ‘Java’, ‘Python’)

print(t.count(‘Python’)) 🡪 2

عملگر جمع

T1 = (‘Python’, ‘SQL’)

T2 = (‘R’)

T = t1 + t2

Print(t) 🡪 (‘Python’, ‘SQL’, ‘R’)

(set)\_مجموعه

List 🡪 []

Tuple 🡪 ()

Set 🡪 {}

X = {13, 18, 20}

Print(13 in x) 🡪 True

Print(15 in x) 🡪 False

مقایسه مجموعه با لیست

S = ‘ali’

Print(list(s)) 🡪 [‘a’, ‘l’, ‘i’]

Print(set(s)) 🡪 {‘l’, ‘i’, ‘a’}

در لیست جایگاه قرار گیری هر ایتم مهم است. اما در مجموعه همچین چیزی مهم نیست – حتی اندیس ها در مجموعه کاربردی ندارد.

\_نمی توان عناصر تکراری در مجموعه قرار داد!

حذف عناصر تکراری در لیست

Lst = [1, 2, 3, 4, 3, 5]

S = set(lst)

Print(list(s)) 🡪 [1, 2, 3, 4, 5]

Len

S = {13, 18, 20}

Print(len(s)) 🡪 3

X = {12, ‘ali’, (1, 2), 18.5}

Print(len(x)) 🡪 4

متد کار با مجموعه

Add\_

A = {12, 20, 13}

a.add(14)

a.add(8)

a.add(12)

print(a) 🡪 {8, 12, 13, 14, 20}

update\_

a = {12, 20, 13}

a.update([14, 8])

print(a) 🡪 {8, 12, 13, 14, 20}

remove\_

a = {12, 20, 13}

a.remove(12)

print(a) 🡪 {13, 20}

clear\_

a = {12, 20, 13}

a.clear()

print(a) 🡪 set()

copy\_

a = {1, 2, 3}

b = a.copy()

print(a) 🡪 {1, 2, 3}

print(b) 🡪 {1, 2, 3}

**اشتراک** و اجتماع دو مجموعه

A = {1, 2, 3, 4}

B = {3, 4, 5, 6}

1 2 3 4 5 6

A B

A & B = {3, 4} اشتراک

a.intersection(b) 🡪 {3, 4}

A | B = {1, 2, 3, 4, 5, 6}

a.union(b) 🡪 {1, 2, 3, 4, 5, 6}

تفاضل دو مجموعه

A = {1, 2, 3}

B = {3, 4, 5}

1 2 3 4 5

A B

1 2 3 4 5

A B

1 2 3 4 5

A B

A – b = {1, 2}

B – a = {4, 5}

a.difference(b)

a & b = {3}

a ^ b = (a-b) U (b-a)

زیر مجموعه بودن

X1 = {1, 2}

3 2 1

X2 = {1, 2, 3}

X2 X1

Print(x1.issubset(x2)) 🡪 True

Pint(x1 <= x2) 🡪 True

X1 = {1, 3, 5}

X2 = {2, 4, 6}

Print(x1 & x2) 🡪 set()

Print(x1.isdisjoint(x2)) 🡪 True

\_دیکشنری

Dictionary

مجموعه ای از اشیا که هر آیتم دارای کلید و یک مقدار وابسته به آن است.

sport = {‘Ali’ : ‘Football’,

‘Taha’ : ‘Badminton’,

‘Amin’ : ‘Baseball’}

جایگزینی

Sport[‘Amin’] = ‘volleball’

متد های کار با دیکشنری\_

Keys\_

G = {0 : ‘male’ , 1 : ‘female’}

Print(g.keys()) 🡪 dict\_keys([0, 1])

Values\_

g.values() 🡪 dict\_values([‘male’, ‘female’])

g.items()

dict\_items([(0 : ‘male’), (1, ‘female’)])

clear\_

G = {0 : ‘male’ , 1 : ‘female’}

Print(g) 🡪 {0 : ‘male’ , 1 : ‘female’}

g.clear()

print(g) 🡪 {}

copy\_

d = {1: ‘one’, 2: ‘two’}

c = d.copy()

del d[1]

print(d) 🡪 {2 : ‘two’}

print(c) 🡪 {1: ‘one’, 2: ‘two’}

update\_

d = {2 : ‘two’}

u = {2 : ‘two’, 3 : ‘three’}

d.update(u)

print(d) 🡪 {2 : ‘two’, 3 : ‘three’}

pop\_

d = {1 : ‘one’, 2 : ‘two’, 3 : ‘three’}

x = d.pop(1)

print(x) 🡪 one

print(d) 🡪 {2 : ‘two’, 3 : ‘three’}

ماژول های داخلی پایتون\_

ماژول ها بر دو نوع هستند:

نوع اول\_ ماژول های داخلی

نوع دوم\_ ماژول های تعریف شده توسط کاربر

ماژول های داخلی 🡪

Math

Random

Statistics

Datatme

Math\_

Import math

Print(math.log10(100)) log10010 = 2

Print(math.pow(2, 4)) 16

Print(math.floor(1.9)) [1.9] = 1

Print(math.ceil(1.9)) [1.9] = 2

Print(math.pi) 3.141592

Print(math.) 2.7

ابتدا باید کتابخانه را ایمپورت کرد سپس از موارد داخل ان استفاده کرد.

From math import \* ابتدا تمام موارد داخل کتابخانه را بیاور

M = math() آبجکت سازی

m.log10 به جای مف از ابجکت ساخته شده استفاده می کنیم

m.pi

Statistics\_

Import statistics

Statistics.median([1, 2, 3, 8, 9]) 🡪 3

Statistics.median([1, 2, 3, 7, 8, 9]) 🡪 5

Statistics.mean([1, 2, 3, 4, 5]) 🡪 3 میانگین

Statistics.mode([1, 2, 3, 4, 2]) 🡪 2