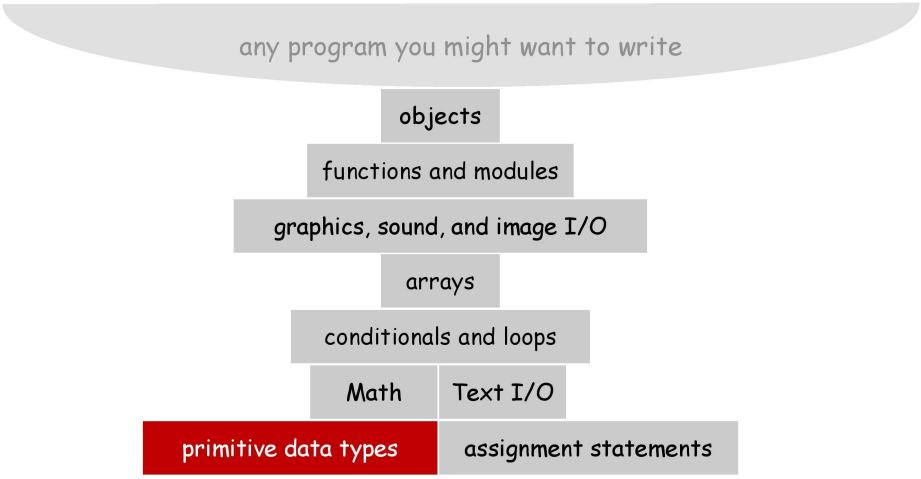
اجزای برنامه‌نویسی



انواع داده‌ای پیش‌ساخته

- نوع داده‌ای. یک مجموعه از مقادیر به همراه عملیات تعریف شده بر روی آن مقادیر.

type	*set of values*	*common operators*	*sample literals*
`int`	integers	`+ - * // % **`	`99 12 2147483647`
`float`	floating point numbers	`+ - * / **`	`3.14 2.5 6.022e23`
`bool`	true-false values	`and or not`	`True False`
`str`	sequence of characters	`+`	`'AB' 'Hello' '2.5'`

برنامه = داده + الگوریتم

اعداد صحیح

- نوع داده‌ای عدد صحیح. مفید برای بیان الگوریتم‌ها.

values	integers						
typical literals		1234	99	-99	0	1000000	
operations	sign	add	subtract	multiply	division	remainder	power
operators	+ -	+	-	*	//	%	**

اعداد اعشاری

- نوع داده‌ای **`float`**. مفید در محاسبات علمی و کاربردهای تجاری.

values	real numbers				
typical literals	3.14159	6.022e23	-3.0	1.4142135623730951	
operations	add	subtract	multiply	division	exponentiation
operators	+	-	*	/	**

مقادیر بولی

- نوع داده‌ای bool. مفید برای کنترل منطق و روند اجرای برنامه.

values	true or false
literals	True False
operations	and or not
operators	and or not

a	not a
False	True
True	False

a	b	a and b	a or b
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

In [ ]:

```
# Data Types
'''
    int
    float
    str
    complex
    bool
'''
```

In [ ]:

```
i = 5
print(type(i))      # <class 'int'>
```

In [ ]:

```
f = 3.6
print(type(f))      # <class 'float'>
```

In [ ]:

```
s = "Python"
print(type(s))      # <class 'str'>
```

In [ ]:

```
c = 2 + 7j
print(type(c))      # <class 'complex'>
```

In [ ]:

```
b = True
print(type(b))      # <class 'bool'>
```

مثال: عملیات بر روی اعداد اعشاری (معادله درجه دوم)

```
import sys
import math

b = float(sys.argv[1])
c = float(sys.argv[2])

discriminant = b*b - 4.0*c
d = math.sqrt(discriminant)

print((-b + d) / 2.0)
print((-b - d) / 2.0)
```

```
% python quadratic.py -3.0 2.0
2.0
1.0
```
$x^2 - 3x + 2 = 0$

```
% python quadratic.py -1.0 -1.0
1.618033988749895
-0.6180339887498949
```
$x^2 - x - 1 = 0$

```
% python quadratic.py 1.0 1.0
ValueError: math domain error
```
$x^2 + x + 1 = 0$

In [ ]:

```
import math

b = -3.0
c = 2.0

dscriminant = b*b - 4.0*c
d = math.sqrt(dscriminant)

print((-b + d) / 2.0)
print((-b - d) / 2.0)
```

In [ ]:

```
print('--variable names---')

print('a2'.isidentifier())      # True
print('2a'.isidentifier())      # False
print('_myvar'.isidentifier())  # True
print('my_var'.isidentifier())  # True
print('my$'.isidentifier())     # False

from keyword import iskeyword
print( iskeyword('if'))         # True
```

تبدیل نوع

- تبدیل نوع. تبدیل از یک نوع به نوع دیگر.
  - تبدیل نوع صریح: با استفاده تبدیل نوع یا توابع

function call	*description*
`str(x)`	تبدیل شی x به یک رشته
`int(x)`	تبدیل رشته یا عدد اعشاری x به یک عدد صحیح
`float(x)`	تبدیل رشته یا عدد صحیح x به یک عدد اعشاری
`round(x)`	نزدیک‌ترین عدد صحیح به عدد x

  - تبدیل نوع ضمنی (خودکار).

```
x = 10 / 4.0
```
عدد صحیح ۱۰ به صورت خودکار به یک عدد اعشاری تبدیل می‌شود

In [ ]:

```
print('---Python Casting---')

i = 4
print(float(i))     # 4.0
```

In [ ]:

```
s ='15'
print(int(s) + 1)   # 16
```

In [ ]:

```
x = 4
y = 8
y, x = x, y         # assign y value to x and x value to y
print(x)            # 8
print(y)            # 4
```

In [ ]:

```
a = 1
b = 2
a, b = b, a + b
print(a)           # 2
print(b)           # 3
```

عملگرهای مقایسه‌ای

- عملگرهای مقایسه‌ای. عملوندهایی از یک نوع را دریافت و یک عملوند از نوع بولی تولید می‌کنند.

op	*meaning*	true	false
==	*equal*	2 == 2	2 == 3
!=	*not equal*	3 != 2	2 != 2
<	*less than*	2 < 13	2 < 2
<=	*less than or equal*	2 <= 2	3 <= 2
>	*greater than*	13 > 2	2 > 13
>=	*greater than or equal*	3 >= 2	2 >= 3

In [ ]:

```
# Operators :

'''
Operators :
    Arithmetic : +,-,*,/,%,**,//
    Assignment : =,+=,-=,*= ,/= ,%= ,//= ,**=
    Comparison : ==,!=,>,<,>=,<=
    Logical    : and, or, not
    Membership : in , not in
    Bitwise    :  &, |, ^, ~, <<, >>
'''
```

In [ ]:

```
print(17 // 2)          # 8
print(17 % 2)           # 1
print(3 ** 2)           # 9
```

In [ ]:

```
y = 4
y //= 2               # y = y // 2
print(y)              # 2
```

In [ ]:

```
print(2 != 3)                    # True
```

In [ ]:

```
print(1<3 and 4>5)               # False
```

In [ ]:

```
a = 15
print(bin(a))                    # 0b1111
b = 9
print(bin(b))                    # 0b1001
c = a | b
print(bin(c))                    # 0b1111
```

مثال: سال کبیسه

- پرسش. آیا یک سال داده شده، یک سال کبیسه است؟
  - بله اگر (۱) بر ۴ بخش‌پذیر باشد و بر ۱۰۰ بخش‌پذیر نباشد یا (۲) بر ۴۰۰ بخش‌پذیر باشد.

```
import sys

year = int(sys.argv[1])

is_leap_year = (year % 4 == 0)
is_leap_year = is_leap_year and (year % 100 != 0)
is_leap_year = is_leap_year or  (year % 400 == 0)

print(is_leap_year)
```

```
% python leapyear.py 2016
True

% python leapyear.py 1900
False

% python leapyear.py 2000
True
```

In [ ]:

```
year = 2019

is_leap_year = (year % 4 == 0)
is_leap_year = is_leap_year and (year % 100 != 0)
is_leap_year = is_leap_year or (year % 400 == 0)

print(is_leap_year)
```

کتابخانه math در پایتون

```
In [1]: import math
```

```
In [2]: dir(math)
```

یک دستور بسیار مفید به منظور کسب اطلاعات اولیه در مورد کتابخانه‌ها

```
['__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
 'copysign',
 'cos',
 'cosh',
```

- کتابخانه math
  - توابع متداول ریاضی
  - لگاریتم و توان‌رسانی
  - توابع مثلثاتی

In [2]:

```
dir(math)
```

Out[2]:

```
['__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
 'copysign',
 'cos',
 'cosh',
 'degrees',
 'e',
 'erf',
 'erfc',
 'exp',
 'expm1',
 'fabs',
 'factorial',
 'floor',
 'fmod',
 'frexp',
 'fsum',
 'gamma',
 'gcd',
 'hypot',
 'inf',
 'isclose',
 'isfinite',
 'isinf',
 'isnan',
 'ldexp',
 'lgamma',
 'log',
 'log10',
 'log1p',
 'log2',
 'modf',
 'nan',
 'pi',
 'pow',
 'radians',
 'remainder',
 'sin',
 'sinh',
 'sqrt',
 'tan',
 'tanh',
```

```
 'tau',
```

In [1]:

```
print('# math #')
import math
print( math.sqrt(16))      # 4.0
print( math.trunc(4.6))    # 4
print( math.factorial(3))  # 6
print( math.log2(16))      # 4.0
print( math.fmod(17,2))    # 1.0
print( math.fabs(-5))      # 5.0
print( math.pow(2,4))      # 16.0
print( math.pi)            # 3.141592653589793
```

```
# math #
4.0
4
6
4.0
1.0
5.0
16.0
3.141592653589793
```

In [ ]:

```
print('# random #')
import random
print( random.randint(1, 10))
print( random.choice([1,10]))
```

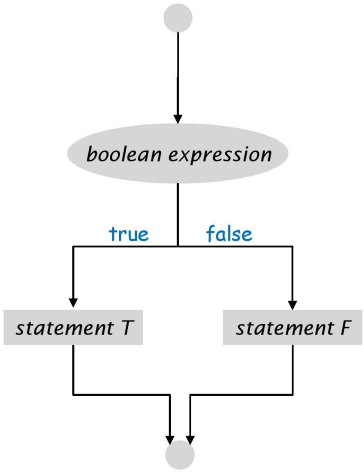
In [ ]:

```
print('# datetime #')
import datetime
now = datetime.datetime.now()
print(now)                 # 2020-05-20 03:20:13.384938
```

دستورات شرطی

دستور if

- دستور `if`. یک ساختار انشعاب متداول.
  - یک عبارت بولی ارزیابی می‌شود.
  - اگر حاصل `true` بود، آنگاه یک مجموعه از دستورات اجرا می‌شوند.
  - اگر حاصل `false` بود، آنگاه یک مجموعه دیگر از دستورات اجرا می‌شوند.

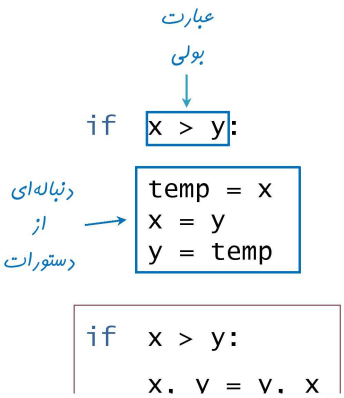



```
if boolean expression:
   statement T
else:
   statement F
```

در اینجا هر مجموعه از دستورات می‌تواند قرار بگیرد.

دستور if

- دستور if. یک ساختار انشعاب متداول.
  - یک عبارت بولی ارزیابی می‌شود.
  - اگر حاصل true بود، آنگاه یک مجموعه از دستورات اجرا می‌شوند.
  - اگر حاصل false بود، آنگاه یک مجموعه دیگر از دستورات اجرا می‌شوند.




```
if  x < 0:
    x = -x
```

```
if  x > y:
    maximum = x
else:
    maximum = y
```

In [ ]:

```
'''
Control statements:
    if
    if else
    elif
'''
```

In [ ]:

```
import math
n = -9
if n < 0 :
   n = abs(n)

print(math.sqrt(n))        # 3.0
```

In [ ]:

```
a = 9
if a % 2 == 0:
    print('evevn')
else:
    print('odd')         # odd
```

In [ ]:

```
print('conditional expression')

a = 3
b = 8

#if a < b:
#    m = a
#else:
#    m = b

m = a if a < b else b

print(m)                # 3
```

In [ ]:

```
grade = 7
s = 'fail' if grade < 10 else 'pass'
print(s)     # fail
```

دستور if

- مثال. شبیه‌سازی پرتاب یک سکه.

```
import random


if random.randrange(0, 2) == 0:
    print('Heads')
else:
     print('Tails')
```

```
% python flip.py
Heads

% python flip.py
Heads

% python flip.py
Tails

% python flip.py
Heads
```

In [ ]:

```
import random

if random.randrange(0,2)==0:
    print('H')
else:
    print('T')
```

In [ ]:

```
today = 'holiday'
b = 40

if today == 'holiday':
    if b > 50:
        print('shopping')
    else:
        print('watch TV')               # watch TV
else:
    print('normal working day')
```

In [ ]:

```
score = 82

if score >= 90:
    l = 'A'
else:
    if score >= 80 :
        l = 'B'
    else:
        if score>= 70:
            l = 'C'
        else :
            l = 'D'

print(l)         # B

# or
if score >= 90:
    l = 'A'
elif score >= 80 :
    l = 'B'
elif score>= 70:
    l = 'C'
else :
    l = 'D'
```

دستورات if تو در تو

- مثال. پرداخت یک نرخ مالیات خاص بر مبنای سطح درآمد.

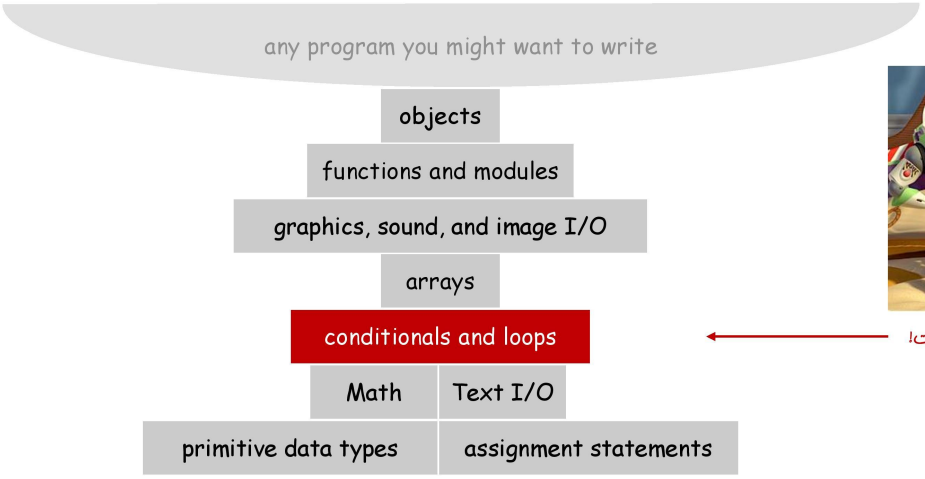
```
if   income < 0:      rate = 0.00
elif income < 8925:   rate = 0.10
elif income < 36250:  rate = 0.15
elif income < 87850:  rate = 0.23
elif income < 183250: rate = 0.28
elif income < 398350: rate = 0.33
elif income < 400000: rate = 0.35
else:                 rate = 0.396
```

Income	Rate
< 0	0 %
0 - 8,925	10%
8,925 - 36,250	15%
36,250 - 87,850	23%
87,850 - 183,250	28%
183,250 - 398,350	33%
398,350 - 400,000	35%
400,000 -	39.6%

In [ ]:

```
income = 8000

if income <0:          rate = 0.00
elif income < 8925:    rate = 0.10
elif income < 36250:   rate = 0.15
elif income < 87850:   rate = 0.23
elif income < 183250:  rate = 0.28
elif income < 398350:  rate = 0.33
elif income < 400000:  rate = 0.35
else:                  rate = 0.396

print(rate * 100)
```

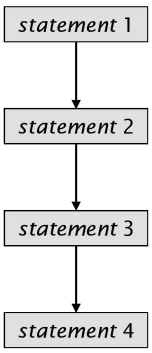
اجزای برنامه‌نویسی



به سوی بی‌نهایت!

جریان کنترل

- جریان کنترل.
  - دنباله‌ای از دستورات در برنامه که واقعاً اجرا می‌شوند.
  - دستورات شرطی و حلقه‌ها: تغییر جریان کنترل.

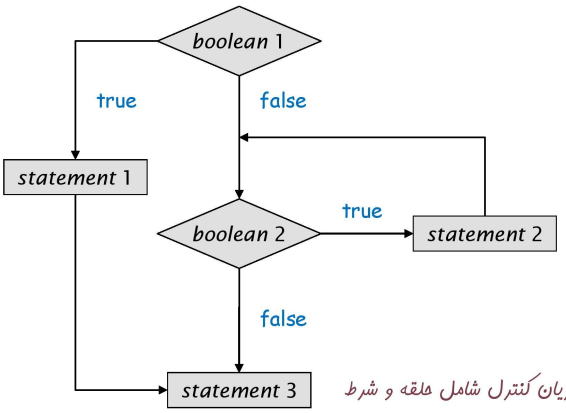


جریان کنترل خط مستقیم

دستورات یکی پس از دیگری به ترتیب مشخص شده اجرا می‌شوند

جریان کنترل

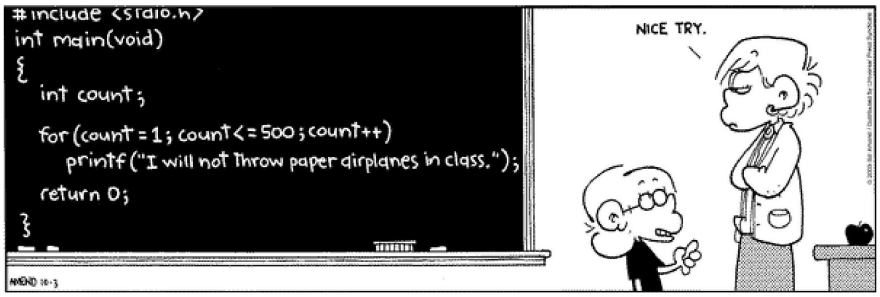
- جریان کنترل.
  - دنباله‌ای از دستورات در برنامه که واقعاً اجرا می‌شوند.
  - دستورات شرطی و حلقه‌ها: تغییر جریان کنترل.



جریان کنترل شامل حلقه و شرط

بسیاری از برنامه‌ها به جریان کنترل پیچیده‌تری نیاز دارند

حلقه for



In [ ]:

```
"""
    loop :
          for
          while
"""
```

In [ ]:

```
for i in range(3):
    print(i , end = ' ')      # 0 1 2

for _ in range(3):
    print('hello')
```

In [ ]:

```
for j in range(6,11,2):
    print(j , end = ' ' )    # 6 8 10
```

ساختارهای تو در تو

In [ ]:

```
for i in range(2,5):
    for j in range(1,i):
        print(j , end = ' ')
    print()

'''
1
1  2
1  2  3
'''
```

In [ ]:

```
print('\n # break #')

for i in range(6):
    if i == 4 :
        break
    else:
        print(i,end=' ')   # 0 1 2 3
```

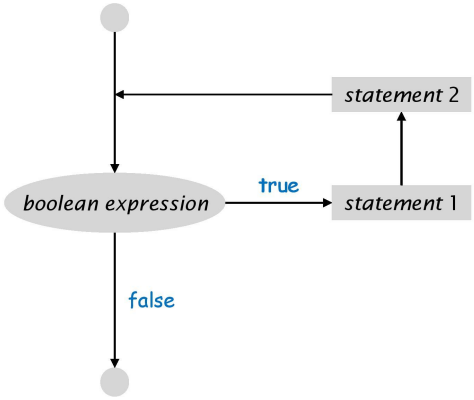
In [ ]:

```
print('\n # continue #')

for i in range(6):
    if i == 4 :
        continue
    else:
        print(i,end=' ')   # 0 1 2 3 5
```

while حلقه

حلقه while

- حلقه `while` یک ساختار تکرار متداول.
  - یک عبارت بولی ارزیابی می‌شود.
  - اگر حاصل `true` بود، تعدادی دستور اجرا می‌شود.
  - تکرار.




```
while boolean expression:
    statement 1
    statement 2
```
← بدنه حلقه

In [ ]:

```
print('\n # while #')

i = 1
while i <= 4:
    print(i , end= ' ')    # 1 2 3 4
    i += 1
```

In [ ]:

```
n = 9
while n > 2:
    n -= 1
    if n == 4:
        break
    print(n , end = ' ')    # 8 7 6 5
```

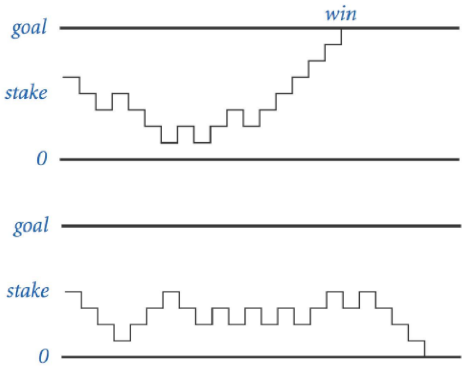
In [ ]:

```
n = 9
while n > 2:
    n -= 1
    if n == 4:
        continue
    print(n , end = ' ')    # 8 7 6 5 3 2
```

شبیه‌سازی مونت-کارلو

ورشکستگی قمارباز

- مسئله. یک قمارباز با مبلغ $stake شروع می‌کند و هر بار بر روی $1 شرط می‌بندد تا این که کل پول خود را ببازد یا موجودی وی به مقدار $goal برسد.
  - شانس برد چقدر است؟
  - برای بردن به چند شرط‌بندی نیاز است؟



- یک رویکرد. شبیه‌سازی مونت کارلو
  - شیر یا خط کن و نتیجه را بررسی کن.
  - عمل فوق را تکرار و آمار مورد نیاز را محاسبه کن.

```
import sys
import random

stake  = int(sys.argv[1])
goal   = int(sys.argv[2])
trials = int(sys.argv[3])

bets = 0
wins = 0
for t in range(trials):
    cash = stake
    while 0 < cash < goal:
        bets += 1
        if random.randrange(0, 2) == 0:
            cash += 1
        else:
            cash -= 1
    if cash == goal:
            wins += 1

print(str(100 * wins // trials) + '% wins')
print('Avg # bets: ' + str(bets // trials))
```

ورشکستگی قمارباز

stake goal trials

```
% python gambler.py 10 20 1000
49% wins
Avg # bets: 99

% python gambler.py 50 250 100
21% wins
Avg # bets: 12712

% python gambler.py 500 2500 100
21% wins
Avg # bets: 1002424
```

In [ ]:

```
import random

stake = 10
goal = 20
trials = 1000

bets = 0
wins = 0

for t in range(trials):
    cash = stake
    while 0 < cash < goal:
        bets += 1
        if random.randrange(0, 2) == 0:
            cash += 1
        else:
            cash -=1
    if cash == goal:
        wins += 1

print(str(100 * wins // trials) + '% wins')
print('Avg # bets: ' + str (bets // trials))
```

شبیه‌سازی و تحلیل نتایج

- حقیقت. احتمال برنده شدن = $stake \div goal$

تفاوت میان موجودی اولیه و هدف

- حقیقت. تعداد مورد انتظار شرط‌بندی‌ها = $stake \times desired\ gain$

- مثال.
  - شانس تبدیل ۵۰۰ دلار به ۲۵۰۰ دلار برابر است با ۲۰ درصد.
  - همچنین تعداد متوسط شرط‌بندی‌ها برابر است با ۱ میلیون!

- ملاحظه. هر دو حقیقت بالا به صورت ریاضی قابل اثبات هستند؛ اما برای سناریوهای پیچیده‌تر، شبیه‌سازی کامپیوتری اغلب بهترین (تنها) روش است.

جریان کنترل: خلاصه

- جریان کنترل.
  - دنباله‌ای از دستورات در برنامه که واقعاً اجرا می‌شوند.
  - دستورات شرطی و حلقه‌ها: تغییر جریان کنترل.

جریان کنترل	توضیحات	مثال‌ها
خط مستقیم	تمام دستورات برنامه به ترتیب داده شده اجرا می‌شوند	

دستورات شرطی	برخی از دستورات برنامه بسته به مقدار بعضی از متغیرهای خاص اجرا می‌شوند	`if`
		`if-else`
		`if-elif`

حلقه‌های تکرار	تا زمانی که شرایط خاصی برقرار باشد، برخی از دستورات به طور مکرر اجرا می‌شوند	`while` `for`

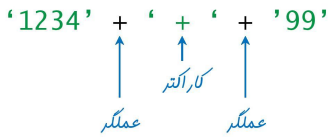
کار با داده‌های متنی

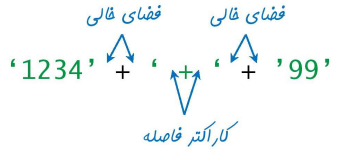
- نوع داده‌ای رشته. مفید برای عملیات ورودی و خروجی برنامه و پردازش متن.

values	sequence of characters
typical literals	'Hello, world'
operation	concatenation
operator	+

expression	*value*
'Hi, ' + 'Bob'	'Hi, Bob'
'1' + ' 2 ' + '1'	'1 2 1'
'1234' + ' + ' + '99'	'1234 + 99'
'1234' + '99'	'123499'

هشدار. معنای کاراکترها به محل استفاده آنها بستگی دارد.





In [ ]:

```
'''
  string
        len , is..  ,find , count  , title , ljust , startswith , replcae ,
        strip , split , join , format , ...
'''
```

In [ ]:

```
print('\n---String---')

s = 'python'

print(len(s))               # 6
print('th' in s)           # True

print(s.islower())          # True
print(s.isalpha())          # True
print(s.isdigit())          # False

print(s.find('o'))          # 4
print(s.count('o'))         # 1

print(s.title())            # Python
print(s.upper())            # PYTHON

print(s.ljust(8,'+'))       # python++
print(s.startswith('py'))   # True

print(s.replace('thon','ramid'))   # pyramid
```

In [ ]:

```
s = '$python$$'
print(s.strip('$'))     # python
```

In [ ]:

```
s = 'Python created by Rossum'
a = s.split(' ')
print(a)            # ['Python', 'created', 'by', 'Rossum']
```

In [ ]:

```
b = ['Python', 'created', 'by', 'Rossum']
c =' '.join(b)
print(c)
```

In [ ]:

```
print('# format #')

s = 'python'

print(f'name : {s}')           # name : python
print('name:{}'.format(s))     # name : python
```

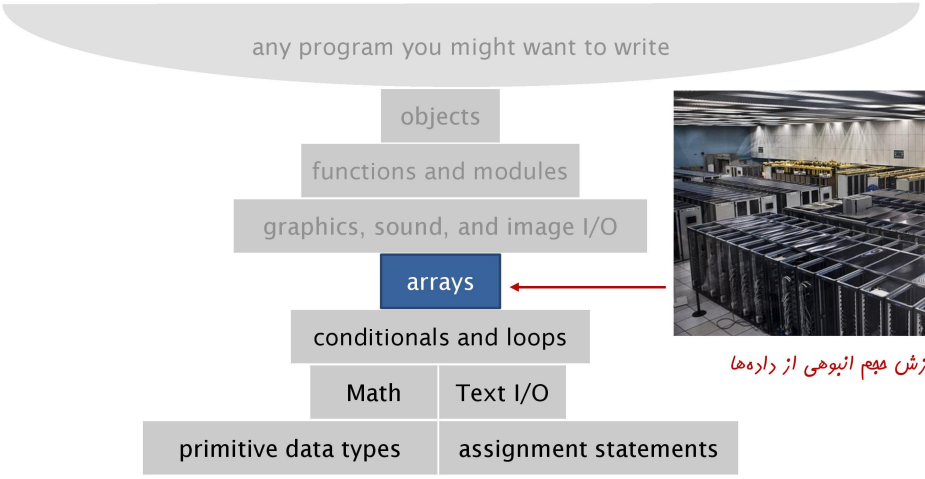
In [ ]:

```
a = 'farshid'
b = 'shirafkan'

print('name:{0}   family:{1}'.format(a, b))

# name:farshid   family:shirafkan
```

In [ ]:

```
a = 15
b = 17.9999

print('{:d} {:.1f}'.format(a, b))    # 15  18.0
```

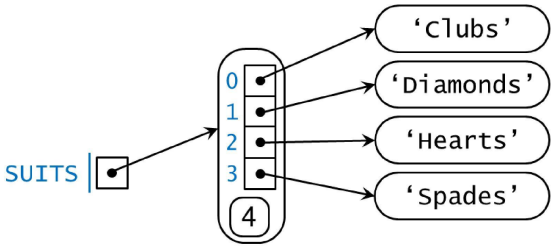
اجزای برنامه‌نویسی



توانایی ذخیره و پردازش حجم انبوهی از داده‌ها

آرایه‌ها

- ساختمان داده.
  - روش ذخیره کردن داده‌ها در حافظه (به منظور دسترسی و پردازش آسان‌تر و کارآتر داده‌ها)
- آرایه. یک ساختمان داده به منظور ذخیره‌سازی یک دنباله از (ارجاع‌ها به) اشیا
  - برای دسترسی به عناصر آرایه از شماره‌گذاری و اندیس‌گذاری استفاده می‌کنیم.



ساختمان داده آرایه

آرایه‌ها

- آرایه. یک دنباله اندیس‌گذاری شده از (ارجاع به) اشیا.

index	value
0	2♥
1	6♠
2	A♦
3	A♥
...	
49	3♣
50	K♣
51	4♠

- مثال‌ها.
  - ۵۲ کارت بازی در یک دسته ورق.
  - ۱۰ هزار دانشجوی کارشناسی در دانشگاه تبریز
  - ۱ میلیون پیکسل در یک تصویر دیجیتال
  - ۴ میلیارد نوکلئوتید در یک رشته DNA
  - ۷۳ میلیارد پرس و جوی گوگل در یک سال
  - ۸۶ میلیاد نورون در مغز
  - ۵۰ تریلیون سلول در بدن انسان

پردازش تعداد زیادی مقدار از یک نوع

۱۰ مقدار، بدون استفاده از آرایه

```
# tedious and error-prone
a0 = 0.0
a1 = 0.0
a2 = 0.0
a3 = 0.0
a4 = 0.0
a5 = 0.0
a6 = 0.0
a7 = 0.0
a8 = 0.0
a9 = 0.0
…
a4 = 3.0
…
a8 = 8.0
…
x = a4 + a8
```

۱۰ مقدار، با استفاده از آرایه

```
# easy alternative
a = [0] * 10
…
a[4] = 3.0
…
a[8] = 8.0
…
x = a[4] + a[8]
```

یک میلیون مقدار، با استفاده از آرایه

```
# handle huge amounts of data
a = [0] * 1000000
…
a[123456] = 3.0
…
a[987654] = 8.0
…
x = a[123456] + a[987654]
```

In [ ]:

```
"""
    list
    'index' ,'count','insert','remove','pop','reverse' , 'sort' ,'extend'
    , 'append','clear','copy',...

"""
```

In [ ]:

```
a = [8, 2, 12]

print(type(a))      # <class 'list'>

print(len(a))       # 3

print(a.index(2))   # 1

print(a[1])         # 2
a[1] = 7            # list is mutable
```

In [ ]:

```
a = [13, 5, 30, 8, 6, 25]

print(a[1:4])        # [5, 30, 8]
print(a[0:3])        # [13, 5, 30]
print(a[3:])         # [8, 6, 25]
print(a[::-1])       # [25, 6, 8, 30, 5, 13]

print(a[0:7:2])      # [13, 30, 6]
```

کار کردن با آرایه‌ها در پایتون

- حلقه زدن بر روی عناصر یک آرایه.

محاسبه میانگین عناصر یک آرایه

```
total = 0.0

for v in a:
    total += v

average = total / len(a)
```

محاسبه میانگین عناصر یک آرایه

```
total = 0.0

for i in range(len(a)):
    total += a[i]

average = total / len(a)
```

In [ ]:

```
a = [13, 5, 30, 8, 6, 25]

total = 0.0
for i in range(len(a)):
    total += a[i]

average = total / len(a)
print(average)
```

In [ ]:

```
a = [13, 5, 30, 8, 6, 25]

total = 0.0
for v in a:
    total += v

average = total / len(a)
print(average)
```

In [ ]:

```
child = ['sara','mahsa']
for i in child:
    print(i)

# or
for i in range(len(child)):
    print(child[i])
```

In [ ]:

```
a = [3, 5]
b = a*2
print(b)           # [3, 5, 3, 5]
```

In [ ]:

```
a = [1, 2 , 3]
b = ['a', 'b']
c = a + b
print(c)           # [1, 2, 3 ,'a', 'b']
```

In [ ]:

```
a = [15 , 5 , 67 , 3 ]

print(max(a))     # 67
print(min(a))     # 3
print(sum(a))     # 90

print(a.count(5)) # 1

a.insert(2,8)
print(a)          # [15, 5, 8, 67, 3]

a.remove(67)
print(a)          # [15, 5, 8, 3]

print(a.pop())    # 3
print(a)          # [15, 5, 8]

print(a.pop(1))   # 5
print(a)          # [15, 8]

del a[1]
print(a)          # [15]
```

In [ ]:

```
a = [3,7,5,4]
a.reverse()
print(a)          # [4,5,7,3]

a.sort()
print(a)          # [3,4,5,7]
```

In [ ]:

```
a = [1,2]
a.append(9)
print(a)          # [1, 2, 9]
```

In [ ]:

```
x = [1, 2]
y = ['a','b']
x.append(y)
print(x)          # [1, 2, ['a','b']]
print(len(x))     # 3
```

In [ ]:

```
x = [1, 2]
y = ['a','b']
x.extend(y)
print(x)          # [1, 2,'a','b']
print(len(x))     # 4
```

In [ ]:

```
a = [1, 2, 3, 4]
a.clear()
print(a)          # []
print(len(a))     # 0
```

In [ ]:

```
a = [1,2]
b = a.copy()
print(b)         # [1, 2]
```

In [ ]:

```
a = []
for i in range(3):
    a.append(i)

print(a)          # [0, 1, 2]

# or
a = [i for i in range(3)]
print(a)          # [0, 1, 2]
```

In [ ]:

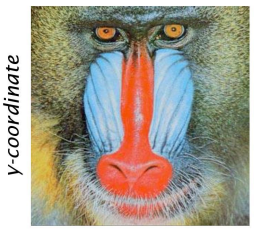
```
grade = [5, 18, 20, 4, 7]

a = [i for i in grade if i > 10]
print(a)                          # [18,20]
```

آرایه‌های دو بعدی

- آرایه‌های دو بعدی.
  - ماتریس‌ها در محاسبات ریاضی.
  - یک جدول از نمرات به ازای هر دانشجو و تمرینات.
  - یک جدول از داده‌ها به ازای هر آزمایش و نتایج آن.
  - تراکنش‌های مربوط به مشتریان یک بانک.
  - پیکسل‌ها در یک تصویر دیجیتال.
  - داده‌های جغرافیایی.

grade

student ID	0	1	2	3	4	5	...
0	A	A	C	B	A	C	
1	B	B	B	B	A	A	
2	C	D	D	B	C	A	
3	A	A	A	A	A	A	
4	C	C	B	C	B	B	
5	A	A	A	B	A	A	
...							



In [ ]:

```
print('---- matrix ----')

m  = [
       [1,2,3],
       [4,5,6],
       [7,8,9]
      ]

print(len(m)) # 3
```

In [ ]:

```
print(m[0])   # [1, 2, 3]
```

In [ ]:

```
for i in m:
    print(i)
```

In [ ]:

```
for i in m:
    print(i[0],end=' ')   # 1  4  7
```

In [ ]:

```
for i in range(0,3) :
    print(m[i][i],end=' ')  # 1  5  9
```

In [ ]:

```
for i in range(0,3):
    print(m[i][2-i],end=' ') # 3  5  7
```

In [ ]:

```
a = []
a.extend([sum(i) for i in m])
print(a)    # [6, 15, 24]
```

In [ ]:

```
b = []
for col in range(3):
    b.append(sum(i[col] for i in m))
print(b)    # [12, 15, 18]
```

جمع ماتریسی

- جمع ماتریسی. با داشتن دو ماتریس a و b با ابعاد n در n، ماتریس c یک ماتریس n در n است به گونه‌ای که:

```
c[i][j] = a[i][j] + b[i][j]
```

```
for i in range(n):
    for j in range(n):
        c[i][j] = a[i][j] + b[i][j]
```

a[][]					b[][]					c[][]			
	.70	.20	.10			.80	.30	.50			1.5	.50	.60
	.30	.60	.10		+	.10	.40	.10		=	.40	1.0	.20
	.50	.10	.40			.10	.30	.40			.60	.40	.80

In [ ]:

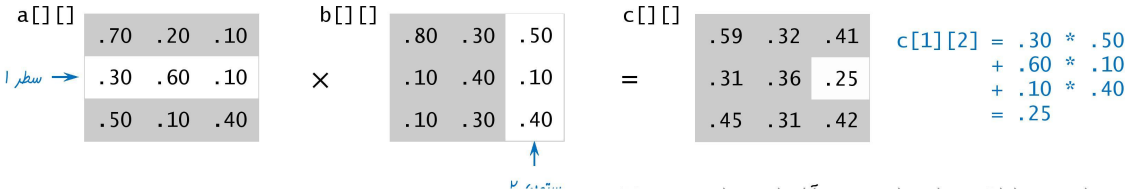
```
a  = [
       [1,2,3],
       [4,5,6],
       [7,8,9]
      ]

b  = [
       [10,11,12],
       [13,14,15],
       [16,17,18]
      ]

c  = [[0,0,0],
      [0,0,0],
      [0,0,0]
     ]

for i in range(3):
    for j in range(3):
        c[i][j] = a[i][j] + b[i][j]
print(c)
```

ضرب ماتریسی

- ضرب ماتریسی. با داشتن دو ماتریس a و b با ابعاد n در n، ماتریس c یک ماتریس n در n است به گونه‌ای که در آن c[i][j] برابر است با ضرب داخلی سطر i از ماتریس a[][] در ستون j از ماتریس b[][].

```
for i in range(n):
    for j in range(n):
        for k in range(n):
            c[i][j] += a[i][k] * b[k][j]
```

ضرب داخلی
سطر i از ماتریس a[][] در
ستون j از ماتریس b[][]



In [ ]:

```
a  = [
       [1,2,3],
       [4,5,6],
       [7,8,9]
      ]

b  = [
       [10,11,12],
       [13,14,15],
       [16,17,18]
      ]

c  = [[0,0,0],
      [0,0,0],
      [0,0,0]
     ]

for i in range(3):
    for j in range(3):
        for k in range(3):
            c[i][j] += a[i][k] * b[k][j]
print(c)
```

In [ ]:

```
"""
    Tuple:
    len , index, sum , min , max , count , ...
"""
```

In [ ]:

```
t = ('English', 'Art', 'Mathematics')

print(type(t))   # <class 'tuple'>

print(len(t))    # 3

print(t[1])      # Art

print(t[1:3])    # ('Art', 'Mathematics')

print(t.index('Art'))      # 1

print( 'Art' in t)         # True

# t[0] = 'history'         # error : tuples are immutable
```

In [ ]:

```
t = (1, 7, 5)

print(sum(t))      # 13
print(max(t))      # 7
print(min(t))      # 1

print(t.count(9))  # 0

print(tuple(reversed(t)))  # (7,5,1)
```

In [ ]:

```
t = (1, 2)
a = list(t)
a.append(3)
t = tuple(a)
print(t)           # (1, 2, 3)
```

In [ ]:

```
t = (14, 7, 3, 19)
x = list(t)
x.remove(3)
t = tuple(x)
print(t)          # (14, 7, 19)
```

In [ ]:

```
t = ('red', 8)
x,y = t
print(x)         # red
print(y)         # 8
```

In [ ]:

```
a = (1, 2)
b = ('x', 'y')

c = zip(a,b)

print(list(c))           # [(1, 'x'), (2, 'y')]

# unzip
x = [(1, 'x'), (2, 'y')]
u = zip(*x)
print(list(u))          # [(1, 2), ('x','y')]
```

In [ ]:

```
"""
   dictionary:
       len , get , keys , values , items , pop , popitems ,...
"""
```

In [ ]:

```
print('# dict #')


d = {
     'brand' : 'b' ,
     'model' : 'm' ,
     'color' : 'red' ,
     'year'  : 2020
     }

# or
# d = dict( brand = 'b' , model='m' , color = 'white' , 'year' = 2020)
```

In [ ]:

```
d = {'x': 14, 'y': 32, 'z': 11, 'w': 7}

print(type(d))           # <class dict>

print(len(d))            # 4

print( d['y'])           # 32

print(d.get('y'))        # 32

print(list(d.keys()))    # ['x', 'y', 'z', 'w']

print(list(d.values()))  # [14, 32, 11, 7]

print(list(d.items()))   # [('x', 14), ('y', 32), ('z', 11), ('w', 7)]

for k,v in d.items():
    print(k,':',v)
'''
x : 14
y : 32
z : 11
w : 7
'''

d.pop('y')
print(d)                 # {'x': 14, 'z': 11, 'w': 7}

d.popitem()
print(d)                 # {'x': 14, 'z': 11}

d.clear()
print(d)                 # {}

del d
```

In [ ]:

```
d = {'x': 14, 'y': 32, 'z': 11, 'w': 7}

import operator

k = operator.itemgetter(1)

print(sorted(d.items(),key = k))

# [('w', 7), ('z', 11), ('x', 14), ('y', 32)]
```

In [ ]:

```
# combine
d1 = {'x' : 3 , 'y': 2 , 'z':1}
d2 = {'w' : 8 , 't': 7 }
d = {}

d = d1.copy()
d.update(d2)

print(d)      # {'x': 3, 'y': 2, 'z': 1, 'w': 8, 't': 7}

# or
d = {**d1 , **d2}
```

In [ ]:

```
k = ['a' , 'b']
v = [4 , 8]

z = zip(k,v)

d = dict(z)
print(d)       # {'a': 4, 'b': 8}
```

In [ ]:

```
### Nested dict

myfamily = {
        'child1': {'name':'taha'  , 'age' : 8}  ,
        'child2': {'name':'mahsa' , 'age' : 20}
        }
```

In [ ]:

```
p = {
     'name'     : 'farshid',
     'children' : ['mahsa', 'taha'],
     'phone'    : {'home':'021-4455', 'mobile':'0912-1972028'}
    }

print(len(p))                # 3

print(p['phone']['mobile'])  # 0912-1972028

print(p['children'][0])      # mahsa
```

In [ ]:

```
"""
    Set :
        len , add, update , remove , discard , pop , copy , clear,
        intersection , union , difference , update , issubset , isdisjoint , ...
"""
```

In [ ]:

```
S = {'a','e','o','i'}

print(type(S))     # <class 'set'>

print(len(S))      # 4

print('u' in S)    # False

S.add('u')
print(S)           # {'i', 'u', 'e', 'a', 'o'}
```

In [ ]:

```
S.remove('i')
print(S)           # {'u', 'e', 'a', 'o'}
```

In [ ]:

```
c = S.copy()
print(c)          # {'a', 'o', 'e', 'u'}
```

In [ ]:

```
S.clear()
print(S)                 # set()
print(len(S))            # 0
```

In [ ]:

```
X = {1, 2, 3 ,4 , 5}
Y = {2, 4}

print(X.intersection(Y))    # {2, 4}
print(X & Y)                # {2, 4}
```

In [ ]:

```
print(X.union(Y))           # {1, 2, 3, 4 , 5}
print(X | Y)                # {1, 2, 3, 4 , 5}
```

In [ ]:

```
print(X.difference(Y))      # {1, 3, 5}
print(X-Y)                  # {1, 3, 5}
```

In [ ]:

```
X = {'A', 'M'}
Y = {'A','C','M','F'}
print(X.issubset(Y))     # True
```

دانشگاه شهید مدنی آذربایجان
برنامه نویسی پیشرفته با پایتون
امین گلزاری اسکوئی
۱۴۰۰-۱۴۰۱

Codes and Projects (click here) (https://github.com/Amin-Golzari-Oskouei/Python-Programming-Course-Advanced-2021) slides and videos (click here) (https://drive.google.com/drive/folders/1Dx3v7fD1QBWL-MNP2hd7ilxaRbeALkkA)