<u>Числа и Строки</u>

Обсудим отдельно чиселки. И, Конечно, нас преимущественно будут интересовать вещественные.

Для начала базированная база: Вещественного числа не существует. Мы же в дискретном мире, у нас просто с какой-то точностью чиселки. При этом при операциях всё очень просто летит в тартараты

```
>>> 1.1 + 2.2
3.3000000000000003
>>> 1.1
1.1
>>> 2.2
2.2
>>> 3.3
3.3

>>> 1.1 + 2.2
3.3000000000000003
>>> 3.3 - 1.1
2.1999999999999997
>>> 1.1 + 2.2 - 1.1
2.2
```

Для представления нецелых чиселок есть специальные классы в присоединяемых модулях, как, н-р, класс дробей из модуля fractions

```
>>> from fractions import Fraction as F
>>> a, b = F(5), F(7)
>>> a
Fraction(5, 1)
>>> b
Fraction(7, 1)
```

Объекты представляют из себя как бы обыкновенную дробь с числителем и знаменателем

```
>>> str(a)
'5'

>>> a + b
Fraction(12, 1)
>>> a / b
Fraction(5, 7)
>>> a / b / 9 * 3
Fraction(5, 21)
>>> a / b / 9 * 3 * 11
Fraction(55, 21)
>>> a / b / 9 * 3 * 111
Fraction(185, 7)
```

Если нам принципиально работать с десятичными дробями, используем класс объектов повышенной точности - Decimal. Здесь можно даже задавать точность, с которой будет рассчитываться дробная часть

```
>>> from decimal import Decimal as D
>>> D(5) / D(7)
Decimal('0.7142857142857142857142857143')

>>> import decimal

>>> decimal.getcontext().prec = 30
>>> D(5) / D(7)
Decimal('0.714285714285714285714285714286')

>>> decimal.getcontext().prec = 60
>>> D(5) / D(7)
Decimal('0.714285714285714285714285714285714285714285714285714285714286')

>>> D(5) / D(7) * 10900
Decimal('7785.71428571428571428571428571428571428571428571428571428572')

>>> 5/7
0.7142857142857143

>>> d = D(5)
>>> d.sqrt()
Decimal('2.23606797749978969640917366873127623544061835961152572427090')
```

Тк у вещественных типов питончика ограниченная точность, которая ещё и едет при операциях, передавать в D и F вещественную дичь - моветон. Будет получаться дичь. Можно передать в виде строки

>>> a = F(1.1)
>>> a
Fraction(2476979795053773, 2251799813685248) # А казалось бы, 11/10, но нет, ибо истинный вид 1.1 это …
>>> a = D(1.1)
>>> a
Decimal('1.100000000000000088817841970012523233890533447265625') # … вот эта дичь)))

>>> a = D("1.2345")
>>> a
Decimal('1.2345')

Во встроенных типах питончика есть комплексные числа. Мнимая единица здесь считается через j, а не через i, и так это слишком популярный символ.
В библиотеке функций даже есть методы работы с комплексными числами

>>> x = 2 + 6j
>>> x
(2+6j)
>>> x**2
(-32+24j)

>>> from math import *
>>> abs(x)
6.324555320336759
>>> exp(x)
Traceback (most recent call last):
*** File "<stdin>", line 1, in <module>***
TypeError: must be real number, not complex

>>> import cmath
>>> cmath.exp(x)
(7.094752112585877-2.0646167911025195j)

Как же работать с обычными веществеными, если они такие кривые? Использовать не точное, а приближённое равенство через функцию ***isclose***

```
>>> a, b, c = 1.1, 3.3, 4.4
>>> a + b == c
True
>>> a, b, c = 1.1, 2.2, 3.3
>>> a + b == c
False
>>> isclose(a + b, c)
True
```

Теперь про Псевдослучайные числа. Во всех культурных местах, и в питоне, поэтому, тоже, схема одна и та же: есть зерно - стартовое значение генератора. В генераторе алгоритм, выдающий следующее число исходя из своего состояния и предыдущего числа. Начиная с одного и того же зерна, можно получать одинаковые генерации

```
>>> import random
>>> dir(random)
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST',
'SystemRandom', 'TWOPI', '_ONE', '_Sequence', '_Set', '__all__', '__builtins__',
'__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
'__spec__', '_accumulate', '_acos', '_bisect', '_ceil', '_cos', '_e', '_exp', '_floor', '_index',
'_inst', '_isfinite', '_log', '_os', '_pi', '_random', '_repeat', '_sha512', '_sin', '_sqrt', '_test',
'_test_generator', '_urandom', '_warn', 'betavariate', 'choice', 'choices', 'expovariate',
'gammavariate', 'gauss', 'getrandbits', 'getstate', 'lognormvariate', 'normalvariate',
'paretovariate', 'randbytes', 'randint', 'random', 'randrange', 'sample', 'seed', 'setstate',
'shuffle', 'triangular', 'uniform', 'vonmisesvariate', 'weibullvariate']
>>> random.randint(0, 20)
4
>>> random.randint(0, 20)
1
>>> random.randint(0, 20)
9
>>> random.randint(0, 20)
16

>>> [random.randint(0, 20)for i in range(20)]
[8, 0, 7, 14, 8, 20, 14, 17, 11, 5, 10, 8, 13, 8, 2, 9, 5, 13, 3, 6]
>>> [random.randint(0, 20)for i in range(20)]
[7, 7, 20, 6, 1, 10, 19, 17, 2, 12, 11, 1, 16, 3, 11, 0, 0, 10, 4, 4]
>>> [random.randint(0, 20)for i in range(20)]
[13, 16, 0, 13, 18, 3, 15, 1, 2, 3, 18, 19, 0, 18, 12, 6, 16, 12, 12, 18]
>>> [random.randint(0, 20)for i in range(20)]
[17, 16, 7, 10, 0, 11, 11, 17, 5, 0, 17, 11, 11, 5, 16, 1, 14, 4, 9, 19]
>>> [random.randint(0, 20)for i in range(20)]
[17, 19, 10, 1, 18, 7, 15, 9, 16, 13, 19, 2, 16, 18, 11, 20, 4, 1, 16, 12]
```

```
>>> [random.random() for i in range(20)]
[0.06463147929891722, 0.6783892167357154, 0.4378942222256338,
0.20238127146185747, 0.2386859420676013, 0.30885439930358094,
0.30643554695130215, 0.48325007065309544, 0.799978972221149, 0.293608128418579,
0.7953270256374334, 0.5495064646832856, 0.06112969165230997,
0.9190022036947616, 0.23868341298969686, 0.08610961799264105,
0.6226427558253601, 0.8151919688593626, 0.6056216464623453, 0.0466860124717986]

>>> random.seed(12345)
>>> [random.random() for i in range(5)]
[0.41661987254534116, 0.010169169457068361, 0.8252065092537432,
0.2986398551995928, 0.3684116894884757]
>>> [random.random() for i in range(5)]
[0.19366134904507426, 0.5660081687288613, 0.1616878239293682,
0.12426688428353017, 0.4329362680099159]
>>> [random.random() for i in range(5)]
[0.5620784880758429, 0.1743435607237318, 0.5532210855693298,
0.35490138633659873, 0.9580647850995486]

>>> random.seed(12345)
>>> [random.random() for i in range(5)]
[0.41661987254534116, 0.010169169457068361, 0.8252065092537432,
0.2986398551995928, 0.3684116894884757]
>>> [random.random() for i in range(5)]
[0.19366134904507426, 0.5660081687288613, 0.1616878239293682,
0.12426688428353017, 0.4329362680099159]
>>> [random.random() for i in range(5)]
[0.5620784880758429, 0.1743435607237318, 0.5532210855693298,
0.35490138633659873, 0.9580647850995486]

>>> random.getstate()
(3, (1060008160, 340894186, 922533364, 2934395003, 98727379, 1749811206,
494539548, 4174476334, 1419161510, 3420217662, 1337073780, 1050755561,
4082217290, 3011431138, 2545869326, 3392669912, 3794904851, 3050778150,
244545168, 909117800, 789030313, 706501281, 3470001825, 4060797168, 1658182245,
1825777204, 2771148955, 2246417524, 906182775, 412492871, 267957395, 726868342,
4069677691, 4233964023, 2981155557, 1789435770, 3848087422, 2087518455,
1145732721, 1972062083, 2996191686, 1070242083, 3705173875, 1405903102,
2167190989, 1165072499, 2895949334, 3412673471, 145733581, 2083463172,
2656247082, 1859395208, 2825382751, 2558595824, 1660920600, 2636867415,
917039092, 2785646114, 1520047366, 203223199, 266423309, 218562548, 1996463257,
386850045, 3986489735, 2976563520, 1614186575, 3526925669, 2298589419,
2010865880, 2759884456, 2940900311, 815218851, 3348787643, 3765223504,
4294939674, 3201882382, 3084086879, 755740118, 1054142518, 395755607,
4113691110, 1975703872, 1071261232, 272352503, 1808452270, 4052410596,
3055126595, 4112102137, 1777090857, 1781526410, 1935585404, 3818346254,
1867379836, 186205022, 997410821, 2332379996, 3114377948, 2543426546,
```

1500407189, 2870477372, 256596685, 221196117, 2963342704, 921489241, 603629477, 1424913589, 467851079, 3781625277, 3510843907, 2046836961, 527228879, 240748330, 2360558442, 2856221869, 2398237398, 1707485963, 1332256471, 1266981327, 397490266, 893862880, 1305355410, 2409598228, 714309318, 87072949, 239040086, 1118877615, 2773233585, 1299440413, 4108379195, 3428569617, 3951665031, 301251225, 2685094400, 4129870479, 4052323426, 1093087922, 1023590627, 3976620844, 1133948698, 584033672, 1748100608, 2060543329, 3587004453, 2207546094, 1432974466, 2633760335, 1717883342, 1273389522, 866187418, 1839274430, 2505251036, 1530679313, 271427413, 1660496366, 871238529, 3874431356, 2991711433, 3084983387, 131894248, 3223772612, 3975434461, 4066957227, 2971954785, 2065209660, 3733886458, 2235756599, 1792541877, 337065761, 2810073193, 2293923349, 2314560957, 3397451439, 2270212441, 2662344881, 1292662217, 3222445343, 1038017253, 2703046495, 1639638741, 2710059441, 2146179955, 3584974651, 3685262921, 1918775395, 218446703, 4251937546, 874199793, 2411117066, 4244368073, 1301268052, 2543024646, 2297457851, 3098579699, 1226547557, 450935217, 851375551, 399227820, 3448025593, 3783474655, 1556932763, 3294855579, 1919834772, 2556585843, 3485020715, 1222405832, 675861215, 2142126772, 1534784096, 4270757037, 3798544487, 1562668749, 2153361337, 3763847086, 215062673, 2441817120, 769369137, 1980497617, 481212927, 3551353692, 1823809183, 3463676690, 2211899476, 2941828877, 2413986611, 3535630660, 716296627, 1904852931, 229956369, 688456259, 702723638, 396286495, 3904676761, 900518162, 3062948815, 903805172, 935699012, 2361391592, 2831424656, 2170383980, 2684525314, 1639026267, 3122185903, 1591501556, 482165548, 4268238748, 579035302, 4030629387, 237539840, 2121720903, 2220001201, 3225577512, 2436206224, 2032177843, 1229062521, 3444107472, 3199218644, 275146874, 2543068947, 4028654342, 3452961042, 2075249582, 1369372014, 2963224844, 947158057, 535366882, 2754980919, 3271721843, 1277284066, 4196870259, 520863078, 3994244400, 2194763640, 596354587, 378134212, 199551579, 3110895191, 1076767732, 18589318, 3744712993, 2166917859, 3080058184, 2690113496, 1936801553, 1905970156, 49850942, 2487832105, 3757524160, 301067492, 4249074252, 1399554750, 849221708, 104074255, 98273774, 4071623387, 1001008911, 3406483370, 1951651997, 2583052145, 1079604373, 321947739, 1726753447, 1400473303, 3004176968, 3471276932, 926959852, 2136108110, 1130786213, 1691660979, 1909525512, 3416255808, 2300280833, 2610279224, 1193377489, 3023194000, 2370514980, 282427044, 733540336, 2593370363, 1953015116, 4215918576, 1879957236, 1284216428, 690939285, 3165206485, 3687856588, 2261375533, 3575959743, 3962464322, 980499627, 1389792850, 2185453970, 3786497406, 26993284, 217820392, 2533499053, 1484746915, 1654971804, 510364313, 1171983082, 670542722, 2118172047, 3219688742, 481417600, 1916477496, 2340068221, 161540575, 1931572247, 2814256035, 548796660, 3542408362, 1868693959, 3176630136, 1931299237, 2933573125, 954075681, 2492565198, 803098210, 3650538009, 3660689083, 1760116734, 1365892218, 3363137188, 2935470708, 4001935105, 464930006, 3379308982, 2085655192, 1253562345, 859921641, 2476808654, 2617417937, 832703638, 3110754175, 2145849112, 640889595, 3047295100, 1360660369, 2423791468, 296562056, 265908347, 300959128, 2258237883, 23200535, 185824376, 245574127, 3490882382, 4129478281, 4285436801, 3573331972, 3589414576, 1901275668, 1208179281, 1007616726, 2241501704, 1568997134, 305866871, 1313689490, 1586661844, 3276278049, 885003446, 343010096, 590929623,

2548817465, 1999918245, 1506539771, 3714800530, 366436908, 2696915624, 3426549663, 3564046075, 1019428989, 3795801427, 2674221410, 2315358149, 1821237455, 2903475706, 4067126092, 2297377259, 200737083, 2224917483, 828138154, 1372700024, 2425340107, 1756132206, 1955362541, 204055162, 3693045602, 4023213516, 3526736166, 2389932016, 2777193610, 2815190118, 2150820807, 3266574316, 2981828332, 792261847, 2258699311, 2729938465, 874131744, 3101980149, 1939957603, 332487142, 829197772, 3345757870, 463270549, 3746723239, 3043447698, 54880535, 1327421806, 95613163, 330708823, 2393023519, 3321098334, 1650551363, 3276534414, 3565459456, 2334521143, 2740845806, 3564577816, 176458628, 608104237, 1911733395, 593764068, 3027270362, 1305548617, 3711321019, 1455833193, 3594329852, 2794102466, 2842174320, 1591211582, 956552773, 2711329531, 2194343354, 3099415873, 719813783, 83402108, 3078569641, 1887125903, 2760384428, 324499939, 354608105, 2629002370, 1206586088, 4291903350, 197303978, 1910287883, 1503181495, 1598701070, 552074501, 3989683809, 3747418267, 1239954049, 2089521656, 1119606760, 499249930, 1873982850, 2616722230, 2304485852, 1986225547, 826475342, 1628747648, 2974712764, 3445788700, 3218231236, 398227218, 3736959629, 2652041042, 562060819, 3039730246, 1982537028, 3923134005, 1325884647, 1982434696, 1898895647, 3167249999, 711950290, 605028415, 3258920404, 51792897, 632946205, 1682495113, 216206763, 852106337, 935670518, 1455715875, 1660439874, 2400674723, 2603085572, 362084975, 2442418225, 2374730220, 3787124643, 4082366259, 3067909050, 3908396895, 793977728, 2111883933, 1562423376, 323382996, 2628070155, 2807798926, 2142221462, 2215653893, 1645010381, 3944009467, 1812292737, 3464567740, 3824539086, 3724882623, 1620994501, 2021246775, 2099612277, 1436195830, 1320674937, 464492240, 188659579, 2922094759, 1088011743, 462448289, 714796753, 3915041705, 1035355075, 2590539134, 1565475745, 3098865245, 1740370872, 2302085200, 3211811620, 2879133658, 306426490, 1146129976, 1893256634, 1402066002, 3852320211, 2623607913, 856891062, 2075670029, 1171697441, 2308321202, 2764423596, 1318781647, 1226497071, 117177350, 656703632, 1998620196, 2211828831, 1754688218, 2612874286, 2145056919, 1977565122, 2718517844, 3321727185, 1307277477, 2450400086, 3900720973, 1362939935, 1661678446, 243907285, 2709416988, 833479994, 777071535, 3526643563, 1017948659, 3007780343, 692758575, 1525831482, 1678994643, 2256706618, 4169240418, 1463038132, 301771967, 1626090426, 2737506585, 2489760243, 3575103340, 2950248269, 4160040590, 3069649376, 1737383228, 3844873816, 3281513078, 229380103, 30), None)

```
>>> random.choice("qwertyuiop")
'w'
>>> random.choice("qwertyuiop")
'o'
>>> random.choice("qwertyuiop")
'u'

>>> random.sample("qwertyuiop", k = 5)
['p', 'o', 'e', 'w', 'u']
>>> random.sample("qwertyuiop", k = 5)
```

```
['w', 'r', 'y', 'e', 'q']
>>> random.sample("qwertyuiop", k = 5)
['i', 'y', 'q', 't', 'o']
>>> random.sample("qwertyuiop", k = 5)
['u', 'q', 'o', 'y', 'w']

>>> random.choices("qwertyuiop", k = 5)
['i', 'w', 'q', 't', 'o']
>>> random.choices("qwertyuiop", k = 5)
['y', 'i', 'e', 'y', 'i']
>>> random.choices("qwertyuiop", k = 5)
['o', 'o', 'e', 'o', 'e']
>>> random.choices("qwertyuiop", k = 5)
['o', 'y', 'w', 'y', 'y']
>>> random.choices("qwertyuiop", k = 5)
['p', 'q', 'w', 'p', 'o']
>>> random.choices("qwertyuiop", k = 5)
['w', 'e', 'y', 'w', 'r']

>>> random.choices("qwerty", k = 10)
['q', 't', 'q', 'y', 'y', 't', 't', 'e', 't', 't']
>>> random.choices("qwerty", k = 10)
['e', 'q', 'r', 'w', 'r', 'e', 'y', 'w', 't', 't']
>>> random.choices("qwerty", k = 10)
['r', 'q', 'q', 'y', 'w', 'q', 'w', 'r', 'r', 'q']
>>> random.choices("qwerty", k = 10)
['q', 'e', 'w', 'r', 't', 'q', 'r', 'e', 't', 'y']
>>> random.choices("qwerty", k = 10)
['e', 'w', 'y', 'q', 'w', 'e', 'w', 't', 'r', 'w']
>>> random.choices("qwerty", k = 10)
['e', 't', 'w', 'q', 'e', 'w', 'r', 'w', 'q', 'y']

>>> random.choices("qwerty",weights = (1, 10, 10, 1, 1, 1),  k = 10)
['w', 'w', 'e', 'e', 'e', 'e', 'w', 'w', 'w', 'e']
>>> random.choices("qwerty",weights = (1, 10, 10, 1, 1, 1),  k = 10)
['e', 'e', 'w', 'w', 'e', 'q', 'e', 'e', 'w', 'w']
>>> random.choices("qwerty",weights = (1, 10, 10, 1, 1, 1),  k = 10)
['w', 'e', 'e', 'w', 'q', 'w', 't', 'e', 'e', 't']
>>> random.choices("qwerty",weights = (1, 10, 10, 1, 1, 1),  k = 10)
['q', 'e', 'w', 'w', 'w', 'w', 'r', 'e', 'e', 'w']
>>> random.choices("qwerty",weights = (1, 10, 10, 1, 1, 1),  k = 10)
['w', 'e', 'q', 'w', 'w', 'e', 'w', 'w', 'q', 'w']
>>> random.choices("qwerty",weights = (1, 10, 10, 1, 1, 1),  k = 10)
['e', 'e', 'w', 'w', 'q', 'w', 'w', 'w', 'e', 'w']
```

Теперь Строки. Ещё раз напоминаем: строки состоят из строк

```
>>> s = 'qwertyuiop'
>>> s[7]
'i'
>>> s[7][0]
'i'

>>> ord(s[3]) # выдаёт unicode символа
114

>>> import sys
>>> sys.getsizeof('qwerty')
55
>>> sys.getsizeof('qwertyu')
56
>>> sys.getsizeof('qwertyы')
88
>>> # появился юникодный символ - всё ушло в юникод. а до этого было в Аскии
```

in в строках расширен до поиска подстроки алгоритмом Кнутта-Морриса-Пратта (т.е. за линию времени)

```
>>> 'rt' in 'qwertyuiop'
True
```

Методов у строк до попы, рассмотрим два - split и join. split пилит строку на список “слов”, разделяя их по указанному символу (по умолчанию- пробел). join, наоборот, объединяет список строк в одну через заданную строку-коннектор

```
>>> s = "qwerfgujn      bjvhkv                  fhjcvnhvlkjj ugkfkfvn"
>>> s.split()
['qwerfgujn', 'bjvhkv', 'fhjcvnhvlkjj', 'ugkfkfvn']

>>> type(s.split())
<class 'list'>
>>> res = s.split()
>>> res
['qwerfgujn', 'bjvhkv', 'fhjcvnhvlkjj', 'ugkfkfvn']

>>> '<>'.join(res)
'qwerfgujn<>bjvhkv<>fhjcvnhvlkjj<>ugkfkfvn'
```

Ещё фишка - raw str. Вообще в строку заложены спецсимволы типа \n \t итд. Если мы хотим, чтобы в строке были не эти спецсимволы а просто комбинации \n \t итд, то указываем, что строка должна быть “Сырая”, без преобразований

```
>>> s = 'qwer\nt\tyu\tiop'
>>> s
'qwer\nt\tyu\tiop'
>>> print(s)
qwer
t   yu  iop
>>> s = r'qwer\nt\tyu\tiop'
>>> s
'qwer\\nt\\tyu\\tiop'
>>> print(s)
qwer\nt\tyu\tiop
```

Последняя фича на сегодня - форматные строки. Мы буквально задаём формат вывода, после указываем параметры и вуаля - получаем результат

```
>>> a, b = 10, 20

>>> '***%d***%d***' % (a, b)
'***10***20***'

>>> f'***{a}***{b}***'
'***10***20***'0

>>> f'***{a*2 + 1}***{str(b) * 8}***'  # оно само считает фигурные скобки, Карл!
'***21***20202020202020***'
```