PYTHON – NUMBERS

Numbers

- Integers and floating-point numbers
- Complex numbers
- Fixed-precision decimal numbers
- Rational fraction numbers
- Sets
- Booleans
- Unlimited integer precision
- A variety of numeric built-ins and modules

Literal	Interpretation
`1234, -24, 0, 99999999999999`	Integers (unlimited size)
`1.23, 1., 3.14e-10, 4E210, 4.0e+210`	Floating-point numbers
`0o177, 0x9ff, 0b101010`	Octal, hex, and binary literals in 3.X
`0177, 0o177, 0x9ff, 0b101010`	Octal, octal, hex, and binary literals in 2.X
`3+4j, 3.0+4.0j, 3J`	Complex number literals
`set('spam'), {1, 2, 3, 4}`	Sets: 2.X and 3.X construction forms
`Decimal('1.0'), Fraction(1, 3)`	Decimal and fraction extension types
`bool(X), True, False`	Boolean type and constants

Integer and floating-point literals

- Integers are just strings of decimal digits
- Floating-point -> decimal point and/or signed exponent
- Int() and float() convert back and forth

```
>>>
>>> y=3e3
>>> y
3000.0
>>> x=3e-3
>>> x
0.003
>>> _
```

Hexadecimal, octal, and binary literals

- Hex 0x or 0X
- Octal 0o or 0O
- Binary 0b or 0B
- Built-in calls
  - hex(I), oct(I), and bin(I) convert between
  - int(str, base) converts a string /base to a decimal integer

Processing Number Objects

- Expression operators (+, -, *, /, >, =>,**, &, etc.)
- Built-in mathematical functions (pow, abs, round, int, hex, bin, etc.) - help(__builtins__)
- Utility modules: random, math, etc.

- Numbers are usually processed with expressions, built-ins, and modules - however they also have a handful of type-specific methods

Q: how would you see which methods are available for a literal type?

Binary/Hex

```
>>> X = 0b0001            # Binary literals
>>> X << 2                # Shift left
4
>>> bin(X << 2)           # Binary digits string
'0b100'

>>> bin(X | 0b010)        # Bitwise OR: either
'0b11'
>>> bin(X & 0b1)          # Bitwise AND: both
'0b1'
```

Binary/Hex

```
>>> X = 0xFF                  # Hex literals
>>> bin(X)
'0b11111111'
>>> X ^ 0b10101010            # Bitwise XOR: either but not both
85
>>> bin(X ^ 0b10101010)
'0b1010101'

>>> int('01010101', 2)        # Digits=>number: string to int per base
85
>>> hex(85)                   # Number=>digits: Hex digit string
'0x55'
```

Exercise

Manually convert 34561 base 8 to base 4.

Show your work.

Python Expression Operators

- Expression: a combination of numbers (or other objects) and operators that computes a value when executed

Operators	Description
`yield x`	Generator function send protocol
`lambda args: expression`	Anonymous function generation
`x if y else z`	Ternary selection (x is evaluated only if y is true)
`x or y`	Logical OR (y is evaluated only if x is false)
`x and y`	Logical AND (y is evaluated only if x is true)
`not x`	Logical negation

Precedence

`x in y, x not in y`	Membership (iterables, sets)
`x is y, x is not y`	Object identity tests
`x < y, x <= y, x > y, x >= y`	Magnitude comparison, set subset and superset;
`x == y, x != y`	Value equality operators
`x | y`	Bitwise OR, set union
`x ^ y`	Bitwise XOR, set symmetric difference
`x & y`	Bitwise AND, set intersection
`x << y, x >> y`	Shift x left or right by y bits

Use parentheses!

- Ops execution order in expressions follow precedence.. When precedence is the same the flow is from left to right..
- However, when you enclose subexpressions in parentheses, you override Python’s precedence rules!

Unlimited Precision

```
>>> 2**400
25822498780869085896559191720030118743297057928292235128306593565406476220168411
94629645353280137831435903171972747493376
>>>
>>>
```

math module

```
>>> import math
>>> math.pi, math.e                           # Common constants
(3.141592653589793, 2.718281828459045)

>>> math.sin(2 * math.pi / 180)              # Sine, tangent, cosine
0.03489949670250097

>>> math.sqrt(144), math.sqrt(2)             # Square root
(12.0, 1.4142135623730951)

>>> pow(2, 4), 2 ** 4, 2.0 ** 4.0            # Exponentiation (power)
(16, 16, 16.0)

>>> abs(-42.0), sum((1, 2, 3, 4))            # Absolute value, summation
(42.0, 10)

>>> min(3, 1, 2, 4), max(3, 1, 2, 4)         # Minimum, maximum
(1, 4)
```

`random` module

```
>>> import random
>>> random.random()
0.5566014960423105
>>> random.random()
0.051308506597373515

>>> random.randint(1, 10)
5
>>> random.randint(1, 10)
9

>>> random.choice(['Life of Brian', 'Holy Grail', 'Meaning of Life'])
'Holy Grail'
>>> random.choice(['Life of Brian', 'Holy Grail', 'Meaning of Life'])
'Life of Brian'
```

 # Random floats, integers, choices, shuffles

Exercise

Testing the randomness algorithm in Python..

Loop 10,000 times:

- Generate a random integer between 0-9
- Populate a dictionary DB of 10 keys (0-9) where the value of each key is the number of occurrences of that integer.
- Print the dictionary as follows:

  Value =

  # of hits =

  % deviation from ideal =

- How random does the function appear to be?

```
>>>
Enter iterations  1000000
0     100238   percent   1.0
1     99971   percent   0.99999
2     100165   percent   0.99998
3     99603   percent   0.99997
4     100079   percent   0.99996
5     100357   percent   0.99995
6     100198   percent   0.99994
7     100016   percent   0.99993
8     99692   percent   0.99992
9     99680   percent   0.99991
>>>
```