1 Intro

Useful resource: `https://tenthousandmeters.com/tag/python-behind-the-scenes/`

this document is about learning python. The following is the hello world program:

```
print("Hello World");
```

2 Variables

Each variable is **connected** to a value.

Uppercase letters in variable names have special meaning (later)

Internally, variables are **references** to values in memory.

2.0.1 Strings

You can use both " and ' to delimit them.

Concat them with + to write them over multiple lines, and write
n to write newline.

The internal representation of strings in python is actually not that simple.

The string has 2 possible states: **compact** and **legacy**, in which compact representation basically is a list of UTF-8 characters and is used only *maximum character and size are known at creation time* (eg for string literals).

Otherwise, it will revert to the legacy representation, which, depending on the content of the string, can be of 3 *kinds*

- Latin-1
- UCS-2
- UCS-4

Reported here is the actual struct used in CPython as of PEP393

```
typedef struct {
  PyObject_HEAD
  Py_ssize_t length;
  Py_hash_t hash;
  struct {
      unsigned int interned:2;
      unsigned int kind:2;
      unsigned int compact:1;
      unsigned int ascii:1;
      unsigned int ready:1;
  } state;
  wchar_t *wstr;
} PyASCIIObject;

typedef struct {
  PyASCIIObject _base;
  Py_ssize_t utf8_length;
  char *utf8;
  Py_ssize_t wstr_length;
} PyCompactUnicodeObject;

typedef struct {
  PyCompactUnicodeObject _base;
  union {
      void *any;
      Py_UCS1 *latin1;
      Py_UCS2 *ucs2;
      Py_UCS4 *ucs4;
  } data;
} PyUnicodeObject;
```

link to the documentation: `https://peps.python.org/pep-0393/#string-creation`

We have methods to manipulate the string, like `strip`, `find`, (index), `split`, `join`, we can **use all comparisons operations lexicograhycal**,

We can also query for membership like

```
'a' in 'apple' == True
```

3 Numbers

There are 3 types of number in python: **integers**, **floating-point numbers** and **complex numbers**. The standard library also gives us `decimal.Decimal` and `fractions.Fraction`.

To create a complex number, just append the 'j' to a numeric literal

```
inum = -32432
fnum = 3.32423
cnum = 3.14 - 1j
```

Integers in python are **arbitrary-precision integers**.

```
typedef struct {
    PyObject ob_base;
    Py_ssize_t ob_size; /* Number of items in variable part */
} PyVarObject;

struct _longobject {
    PyVarObject ob_base; // expansion of PyObject_VAR_HEAD macro
    digit ob_digit[1];
};
```

the `ob_digit` member is a pointer to an array of digits. More information on this bignum arithmetic implementation `https://tenthousandmeters.com/blog/python-behind-the-scenes-8-how-python-integers-work/`

This comes with performance implications for each integer operation and the memory consumption of each integer, which is proportional to the number itself. For reference **small numbers take 28 bytes**

- a reference count `ob_refcnt`: 8 bytes
- a type `ob_type`: 8 bytes
- an object's size `ob_size`: 8 bytes
- `ob_digit`: 4 bytes.

Floating numbers are instead double precision floatin point numbers, stored in a `PyObject` type, which is a reference counted object.

```
typedef struct {
    PyObject_HEAD
    double ob_fval;
} PyFloatObject
```

Complex numbers are basically a pair of floating point numbers (double precision)

```
typedef struct {
    PyObject_HEAD
    double cval_real;   // Real part
    double cval_imag;   // Imaginary part
} PyComplexObject;
```

For each of the number types the the following operations are defined

Table 1: Mathematical Operations and Their Descriptions

Operation	Description	Notes
`x + y`	Sum of x and y	
`x - y`	Difference of x and y	
`x * y`	Product of x and y	
`x / y`	Quotient of x and y	
`x // y`	Floored quotient of x and y	(1)(2)
`x % y`	Remainder of x/y	(2)
`-x`	x negated	
`+x`	x unchanged	
`abs(x)`	Absolute value or magnitude of x	
`int(x)`	x converted to integer	(3)(6)
`float(x)`	x converted to floating point	(4)(6)
`complex(re, im)`	A complex number with real part re, imaginary part im (defaults to zero)	(6)
`c.conjugate()`	Conjugate of the complex number c	
`divmod(x, y)`	The pair $(x//y, x\%y)$	(2)
`pow(x, y)`	x to the power y	(5)
`x ** y`	x to the power y	(5)