Integers

Constructors and Bases

(Part 1)

An integer number is an object – an instance of the `int` class

The `int` class provides multiple constructors

```
a = int (10)
a = int (-10)
```

Other (numerical) data types are also supported in the argument of the `int` constructor:

```
a = int(10.9)          ⟶  truncation: a → 10
a = int(-10.9)         ⟶  truncation: a → -10
a = int(True)                         a → 1
a = int(Decimal("10.9"))  ⟶  truncation: a → 10
```

As well as strings (that can be parsed to a number)

```
a = int("10")                         a → 10
```

Number Base

`int("123")` → $(123)_{10}$

When used with a string, constructor has an optional second parameter: `base` 2 <= base <= 36

If base is not specified, the default is base 10 – as in the example above

`int("1010", 2)` → $(10)_{10}$

`int("1010", base=2)` → $(10)_{10}$

`int("A12F", base=16)` → $(41263)_{10}$

`int("534", base=8)` → $(348)_{10}$

`int("a12f", base=16)` → $(41263)_{10}$

`int("A", base=11)` → $(10)_{10}$

```
int("B", 11)   ValueError: invalid literal for int() with base 11: 'B'
```

Reverse Process: changing an integer from base 10 to another base

built-in functions:

```
bin()     bin(10) → '0b1010'

oct()     oct(10) → '0o12'

hex()     hex(10) → '0xa'
```

The prefixes in the strings help document the base of the number `int('0xA', 16)` → $(10)_{10}$

These prefixes are consistent with literal integers using a base prefix (no strings attached!)

```
a = 0b1010     a → 10

a = 0o12       a → 10

a = 0xA        a → 10
```

What about other bases? Custom code

n: number (base 10)

b: base (target base)

$$\frac{?}{b^7} \quad \frac{?}{b^6} \quad \frac{?}{b^5} \quad \frac{?}{b^4} \quad \frac{?}{b^3} \quad \frac{?}{b^2} \quad \frac{?}{b^1} \quad \frac{?}{b^0}$$

```
n = b * (n // b) + n % b
```

→

```
n = (n // b) * b + n % b
```

n = 232 b = 5

$$\frac{?}{5^3}\quad\frac{?}{5^2}\quad\frac{?}{5^1}\quad\frac{?}{5^0}$$

too big

$$\frac{?}{5^3}\quad\frac{?}{5^2}\quad\frac{46}{5^1}\quad\frac{2}{5^0}$$

too big

$$\frac{?}{5^3}\quad\frac{9}{5^2}\quad\frac{1}{5^1}\quad\frac{2}{5^0}$$

$$\frac{1}{5^3}\quad\frac{4}{5^2}\quad\frac{1}{5^1}\quad\frac{2}{5^0}$$

$$\frac{1}{5^3}\quad\frac{4}{5^2}\quad\frac{1}{5^1}\quad\frac{2}{5^0}$$

$$232 = (232 \,//\, 5) \times 5 + 232 \,\%\, 5 = 46 \times 5 + 2$$

$$= [46 \times 5^1] + [2 \times 5^0]$$

$$= [((46 \,//\, 5) \times 5 + 46 \,\%\, 5) \times 5^1] + [2 \times 5^0]$$

$$= [(9 \times 5 + 1) \times 5^1] + [2 \times 5^0]$$

$$= [9 \times 5^2] + [1 \times 5^1] + [2 \times 5^0]$$

$$= [((9 \,//\, 5) \times 5 + 9 \,\%\, 5) \times 5^2] + [1 \times 5^1] + [2 \times 5^0]$$

$$= [(1 \times 5 + 4) \times 5^2] + [1 \times 5^1] + [2 \times 5^0]$$

$$= [1 \times 5^3] + [4 \times 5^2] + [1 \times 5^1] + [2 \times 5^0]$$

div, 3rd mod, 2nd mod, 1st mod

$$= [((1 \,//\, 5) \times 5 + 1 \,\%\, 5) \times 5^3] + [4 \times 5^2] + [1 \times 5^1] + [2 \times 5^0]$$

$$= [(0 \times 5 + 1) \times 5^3] + [4 \times 5^2] + [1 \times 5^1] + [2 \times 5^0]$$

$$= [0 \times 5^4] + [1 \times 5^3] + [4 \times 5^2] + [1 \times 5^1] + [2 \times 5^0]$$

stop, 4th mod, 3rd mod, 2nd mod, 1st mod

Base Change Algorithm

```
n = base-10 number (>= 0)     b = base (>= 2)

if b < 2 or n < 0: raise exception
if n == 0: return [0]

digits = [ ]
while n > 0:
    m = n % b
    n = n // b
    digits.insert(0, m)
```

n = 232
b = 5

digits → [1, 4, 1, 2]

n = 1485
b = 16

digits → [5, 12, 13]

This algorithm returns a list of the digits in the specified base b (a representation of n_{10} in base b)

Usually we want to return an encoded number where digits higher than 9 use letters such as A..Z

We simply need to decide what character to use for the various digits in the base.

Encodings

Typically, we use 0-9 and A-Z for digits required in bases higher than 10

But we don't have to use letters or even standard 0-9 digits to encode our number.

We just need to map between the digits in our number, to a character of our choice.

```
0 → 0
1 → 1
  …
9 → 9

10 → A
11 → B
  …
36 → Z
```

```
0 → 0
1 → 1
  …
10 → A
11 → B
  …
37 → a
38 → b
  …
62 → z
```

```
0 → a
1 → b
  …
9 → i
10 → #
11 → !
  …
36 → *
```

Python uses `0-9` and `a-z` (case insensitive)

and is therefore limited to base <= 36

Your choice of characters to represent the digits, is your encoding map

Encodings

The simplest way to do this given a list of digits to encode, is to create a string with as many characters as needed, and use their index (ordinal position) for our encoding map

```
base b (>=2)

map = ' … ' (of length b)

digits = [ … ]

encoding = map[digits[0]] + map[digits[1]] + …
```

Example: Base 12

map = '0123456789ABC'

digits = [4, 11, 3, 12]

encoding = '4B3C'

Encoding Algorithm

```
digits = [ … ]
map = ' … '

encoding = ''
for d in digits:
    encoding += map[d]    (a += b → a = a + b)
```

or, more simply:

```
encoding = ''.join([map[d] for d in digits])
```

we'll cover this in much more detail in the section on lists

INTEGERS

CONSTRUCTORS AND BASES

(PART 2)

NEXT VIDEO