CS1302 Introduction to Computer Programming - Notes

Lecture 1: Introduction to Computer Programming

Computer

Definition

- can perform arithmetic calculations
- can be **programmed** to perform differing tasks

Architecture

- **Peripherals** (外设): Input & Output Devices
- **Central Processing Unit**: **A**rithmetic and **L**ogic **U**nit + **C**ontrol **U**nit

Programming

CPU's Work

- instruction set
- numbers in RAM

Binary Representation

- Taking 2 Bytes as one, extends the maximum from $2^8 - 1$ to $2^{16} - 1$
- The first program, ENIAC, use Sally System (10 bits represent decimal 0 to 9)
- **Encoding**: ASCII + Unicode
- Other Representation: 2's **complement** (补码) + IEEE **floating**

Programming Languages

1st Gen: Machine Language

Written in binary sequences

2nd Gen: Assembly Language

Both Machine & Assembly language are low-level languages

Platform-specific

3nd Gen: High-level Language

Definition

- Human-understandable
- automatically translated into low-level machine code
- no low-level details (eg. no absolute memory address)

Type

- Compilation
- Interpretation

Lecture 2: Expressions and Arithmetic

Operators

- `+`,`-`,`*`,`/`,`//`,`%`,`**`
- `/` returns `<class 'float'>`
- `//` returns `<class 'int'>` when both operand is int, otherwise returns float.
- `-A%B` returns a nonnegative number.

Operators: Precedence and Associativity

Reference Blog 优先级与结合性

Operators	Precedence	Associativity
`**`	16	right
Unary `-`	14	right
`*` `/` `//` `%`	13	left
`+` `-`	12	left
`>>` `<<`	11	left
`==` `!=` `>` `>=` `<` `<=`	7	left
Unary `not`	4	right
`and`	3	~~left~~ Non
`or`	2	~~left~~ Non

Example

Expression	Parse	Result
`-10**2*3`	`(-(10**2))*3`	-300
`-10*2**3`	`-10*(2**3)`	-80
`-10**2**3`	`-(10**(2**3))`	-10000000
`-10/2*3`	`(-10/2)*3`	15.0
`-10/2**3`	`-10/(2**3)`	1.25

Augmented Assignment Operators

- `+=`, `-=`, `*=`, `/=`, `//=`, `%=`, `**=`
- `:=`: python >= 3.8

```
y = 3 * (x := 15)
x, y

---

(15, 45)
```

Lecture 2: Values and Variables

Integers

- hexadecimal: `0xF`
- decimal: `15`
- octal: `0o17`
- binary: `0b1111`

> The `sys.maxint` constant was removed, since there is no longer a limit to the value of integers.
>
> ———— What's new in Python 3.0

Strings

```
print('\U0001f600:\n\tI\'m a string.')
#     ^             ^^ ^^           ^
# In a single quote string, use \' to escape '
```

```
print("""😀:
        I'm a string.""")

print("\N{grinning face}:", "\tI'm a string.", sep="\n")
#                           ^   ^           ^
# In a double quote string, no need to escape '
---

😀:
    I'm a string.
```

Escape Symbol 转义字符

[Reference Blog]

Character	Usage
`\n`	Newline
`\t`	Tab
`\\`	`\`
`\'`	`'`
`\"`	`"`

Escape Sequence

- `\U0001f600`: Unicode + Hexadecimal
- `\N{grinning face}`

`print()` parameters

```
?print
# or print?

---

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Variables and Assignment

- tuple assignment: `x, y, z = '15', '30', 15`
- chained assignment: `x = y = z = 0`
- Deletion: `del x, y` Accessing deleted variables will raise a `NameError`.

Identifiers

> Variable Names
>
> 1. Must start with a letter or `_` (an underscore) followed by letters, digits, or `_`.
> 2. Must not be a keyword reserved by python.

> - `del` is a keyword and `Del` is not because identifiers are case sensitive.
> - Function names such as `print`, `input`, `type`, etc., are not keywords and can be reassigned.

User Input

```
print('Your name is', input('Please input your name: '))
```

- above use `sep = ' '`
- Function `print` return nothing (the value got is `None`, as seen in below)

Type Conversion

```
type(15), type(print), type(print()), type(None), type(input), type(type),
type(type(type))

---

<class 'int'> <class 'builtin_function_or_method'> <class 'NoneType'> <class
'NoneType'> <class 'method'> <class 'type'> <class 'type'>
```

- `int()`: `str`->`int`
- `str()`: `int`->`str`

Error

- SyntaxError
- TypeError: can only be detected **runtime**

```
TypeError: can only concatenate str (not "int") to str
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- ValueError: raised when a `str` cannot be converted into `int`

```
ValueError: invalid literal for int() with base 10: 'str'
```

> Python is a *strongly-and-dynamically-typed* language:
>
> - *Strongly-typed*: Python does not force a type conversion to avoid a type error.

> - *Dynamically-typed*: Python checks data type only at runtime after translating the code to machine code.
>
> | Language | Type |
> | --- | --- |
> | Python | Strongly Dynamically |
> | Java | Weakly Statically |
> | C++ | Strongly Statically |

Floating Point Numbers

- scientific notation (`±x.yez`) and fraction (`/` result) have type `float`
- `float()` can convert an `int` or a `str` to a `float`

Size Limitations

> ```
> sys.float_info
>
> ---
>
> sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308,
> min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15,
> mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
> ```
>
> any number larger than `max` will become `inf`.

Precision Limitations

Refer to IEEE-754 Floating Point Converter.

`round()`

```
round(2.665, 2), round(2.675, 2), round(150, -2), round(250, -2)

---

(2.67, 2.67, 200, 200)
```

Use rounding method in IEEE-754, whether to float or integer.

The value stored in **binary** might have an **error** compared with **decimal** values.

String Formatting

Reference Doc

Number

```
x = -10000 / 3
print(f"{{:{align}{sign}{'' if width==0 else width}{grouping}.
{precision}f}}".format(x))
```

- `align`: if `width` > `len(str(x))`, number will be aligned.

  whitespaces below are converted into `_` to make reading easier.

Align	Effects	Example of `{:{align}11.2f}`
`<`	Left aligned	`-3333.33___`(8+3)
`>` default	right aligned	`___-3333.33`(3+8)
`=`	`sign` left-most number right-most	`-___3333.33`(1+3+7)
`^`	centered right>=left	`_-3333.33__`(1+8+2)

- `sign`: when set to `+`, positive numbers print `+`. Negative numbers always print `-`.
- `grouping`: every 3 digits in integer part is separated by `grouping`, like `-3,333.33` when set to `,`
- `precision`: decimal places. Number will be rounded.
- outer `{{` and `}}`: to produce character `{` and `}`. The complete `{}` will be replaced by `x` in format.

Placeholders

```
hour = 12
minute = 34
second = 56

print("The time is " + str(hour) + ":" + str(minute) + ":" + str(second)+".")

message = "The time is {}:{}:{}."
print(message.format(hour, minute, second))

---

The time is 12:34:56.
```

- If you print directly like `print("The time is ", hour, ":", minute, ":", second, ".")`, there will be additional whitespaces between items, as `sep = ' '` by default.
- `{}` can be assigned like function parameters.

```
print("You should {0} {1} what I say instead of what I {0}.".format("do",
"only"))
print("The surname of {first} {last} is {last}.".format(first="John",
last="Doe"))

---

You should do only what I say instead of what I do.
The surname of John Doe is Doe.
```

Strings

```
s = "A\nBC"
print(f"{{:{fill}{align}{'' if width==0 else width}}}".format(s))
```

- `fill`: fill the whitespaces with `fill`. Must be one character.
- `align`: refer to this part in #Number. However, using `align='='` for multiline strings will cause `ValueError`.

When `align` is `None`, the string will be printed as-is.

In this case, when `fill` is set to some character, will cause `ValueError`.

Lab 1: Binary

Two's complement 补码

Value and calculation

Dec	Bin	Dec	Bin
0	000	-1	111
1	001	-2	110
2	010	-3	101
3	011	-4	100

$$\overbrace{011_2}^{3}+\overbrace{100_2}^{-4}=\underbrace{111_2}_{-1}$$

$$\overbrace{011_2}^{3}+\overbrace{110_2}^{-2}=\underbrace{1001_2}_{1}$$

- Ignore the overflowing digit

Calculate the complement

Take 8-bit signed int as an example:

Decimal	Sign-Magnitude 原码	1's Complement 反码	2's Complement 补码
-5	`1000 0101`	`1111 1010`	`1111 1011`
-0	`1000 0000`	`1111 1111`	`0000 0000`
-128			`1000 0000`

Lecture 3: Conditional Execution

Boolean Expressions

Comparison Operators

> `==` `!=` `<` `<=` `>` `>=`

[Precedence and Associativity]

- Precedence: `+` `-` > Comparison Operators (All Same)
- Non-associativity
- `1 <=2 < 3 != 4` is interpreted as `(1 <= 2) and (2 < 3) and (3 != 4)`

Comparing between Different Types

- Compare between `int` and `float` for their value
- Compare between `str` and `str` by ascii
- Can only use `!=` or `==` to compare undefined cases, not `<`

```
10 == 10.
"A" < "a"
"aBcd" < "abd"
"A" != 64
"A" < 64

---

True, True, True, True
TypeError: '<' not supported between instances of 'str' and 'int'
```

Comparing with Float Error

```
math.isclose(a, b, rel_tol=1e-9, abs_tol=0)
```

$$|a-b| \leq \max\left\{\delta_{\text{rel}} \max\{a,b\}, \delta_{\text{abs}}\right\}$$

When both a and b is close to 0, specify `abs_tol`.

Boolean Operators

> `and` `or` `not`

[Precedence and Associativity]

- Precedence: Comparison Operators > `not` > `and` > `or`

> **Example**
>
> - A: `True or (False and True)` = `True`
> - B: `(True and False) and True` = `False and True` = `False`
> - C: `True or (True and False)` = `True`

Short-circuit Evaluation 短路求值

- `x or y`: only executes `y` when `x` returns `False` (-> When `x` is `True`, `y` is not executed.)
- `x and y`: only executes `y` when `x` returns `True`
- `or` is evaluated before `and`

```
def f(x):
    return x > 0 or (x := -1) and (x := 2) # (x > 0) or ((x := -1) and (x := 2))
def g(x):
    return x > 0 or (x := 0) and (x := -1) # (x > 0) or ((x := 0) and (x := -1))

---
f(1) = True
f(0) = 2
g(1) = True
g(0) = 0
```

In the case above, if `x <= 0`,

- In `f(x)`, both `(x := -1)` and `(x := 2)` is executed.
- In `g(x)`, `(x := 0)` is interpreted as `False` and `(x := -1)` is not executed.

> - `None`, numeric zero of all types, empty string, empty container is interpreted as `False` in Boolean Operation.
> - All above is seen as `False` in Boolean operations (`if x`), but **not equal to `False` in Comparison operations** (`if x == False`)
> - An example use to simplify `... if ... else ...`
>
> ```
> print("You have entered", a if (a := input()) else "nothing")
> print("You have entered", input() or "nothing")
> ```

Conditional Constructs

(Omitted)

Lecture 7: Lists and Tuples

Constructing Sequences

- `(0,)` is tuple `(0)`, `(0)` is element `0`
- for list, no need to write `[0,]`. `[0]` is list
- `*range(2)` are **two elements** `0, 1`, `*"23"` are **two elements** `"2", "3"`

```
a = (0, [0])
a[1][0] = 1
```

> - `int` is immutable. When a `int` changed from `0` to `1`, you are getting a **different memory location pointed to** `int=1`
> - `list` is mutable. It actually **points to a list of memory location**, the location can be changed.
> - `tuple` is immutable. `a[1]` points to the location of a `list=[0]`. The location of list cannot be changed. But **when element in list change, list stays the same location**.
> - So you can change the `list` inside a `tuple` because `list` itself is not changed. The location in the `list` can have location change.