→ Cheat Sheet

Task	Python Module	Go Package
HTTP calls	urllib	net/http
json	json	encoding/json
CSV	csv	encoding/csv
Date & time	datetime	time
Parse command line arguments	argparse	flag
Regular expressions	re	regexp
logging	logging	log
Run external commands	subprocess	os/exec
Path manipulation	os.path	path/filepath
crypto	hashlib	crypto
Serialization	pickle	encoding/gob
Heap (priority queue)	heapq	container/heap

Types & Declarations

```python
age = 80
name = 'daffy'
weight = 62.3
loons = ['bugs', 'daffy', 'taz']
ages = {  # Correct for 2017
    'daffy': 80,
    'bugs': 79,
    'taz': 63,
}
```

```go
age := 80
name := "daffy"
weight := 62.3
loons := []string{"bugs", "daffy", "taz"}
ages := map[string]int{ // Correct for 2017
    "daffy": 80,
    "bugs":  79,
    "taz":   63,
}
```

Define A Function

```python
def add(a, b):
    """Adds a to b"""
    return a + b
```

```go
// Add adds a to b
func Add(a, b int) int {
    return a + b
}
```

list/slice

```
names = ['bugs', 'taz', 'tweety']
print(names[0])  # bugs
names.append('elmer')
print(len(names))  # 4
print(names[2:])  # ['tweety',
'elmer']
for name in names:
    print(name)

for i, name in enumerate(names):
    print('{} at {}'.format(name, i))
```

```
names := []string{"bugs", "taz",
"tweety"}
fmt.Println(names[0]) // bugs
names = append(names, "elmer")
fmt.Println(len(names)) // 4
fmt.Println(names[2:])  // [tweety elmer]
for _, name := range names {
    fmt.Println(name)
}
for i, name := range names {
    fmt.Printf("%s at %d\n", name, i)
}
```

dict/map

```
ages = {  # Correct for 2017
    'daffy': 80,
    'bugs': 79,
    'taz': 63,
}
ages['elmer'] = 80
print(ages['bugs'])  # 79
print('bugs' in ages)  # True

del ages['taz']

for name in ages:  # Keys
    print(name)

for name, age in ages.items():  # Keys
& values
    print('{} is {} years
old'.format(name, age))
```

```
ages := map[string]int{ // Correct for
2017
    "daffy": 80,
    "bugs":  79,
    "taz":   63,
}
ages["elmer"] = 80
fmt.Println(ages["bugs"]) // 79
_, ok := ages["daffy"]
fmt.Println(ok) // true
delete(ages, "taz")
for name := range ages { // Keys
    fmt.Println(name)
}
for name, age := range ages { // Keys &
values
    fmt.Printf("%s is %d years old\n",
name, age)
}
```

while loop

```
a, b = 1, 1
while b < 10_000:
    a, b = b, a + b
```

```
a, b := 1, 1
for b < 10_000 {
    a, b = b, a+b
}
```

Files

```python
with open('song.txt') as
fp:
    #  Iterate over lines
    for line in fp:

print(line.strip())
```

```go
file, err := os.Open("song.txt")
if err != nil {
    return err
}
defer file.Close()
// Iterate over lines
scanner := bufio.NewScanner(file) // file is an
io.Reader
for scanner.Scan() {
    fmt.Println(scanner.Text())
}
return scanner.Err()
```

Exceptions/Return Error

```python
def div(a, b):
    if b == 0:
        raise ValueError("b can't be
0")
    return a / b

# ...

try:
    div(1, 0)
except ValueError:
    print('OK')
```

```go
func div(a, b int) (int, error) {
    if b == 0 {
        return 0, fmt.Errorf("b can't be
0")
    }
    return a / b, nil
}

// ...

val, err := div(1, 0)
if err != nil {
    fmt.Printf("error: %s\n", err)
}
```

Concurrency

```python
thr = Thread(target=add, args=(1, 2), daemon=True)
thr.start()
```

```go
go add(1, 2)
```

Communicating between threads/goroutines

```python
from queue import Queue

queue = Queue()

# ...

# Send message from a thread
# (in Go this will block until someone reads)
queue.put(353)

# ...

# Get message to a thread
val = queue.get()
```

```go
ch := make(chan int)

// ...

// Send message from a goroutine
// (this will block is there no one reading)
ch <- 353

// ...

// Read message in a goroutine
// (this will block is nothing in channel)
val := <-ch
```

Sorting

```python
names = ['bugs', 'taz', 'daffy']
# Lexicographical order
names.sort()
# Reversed lexicographical order
names.sort(reverse=True)
# Sort by length
names.sort(key=len)
```

```go
names := []string{"bugs", "taz", "daffy"}
// Lexicographical order
sort.Strings(names)
// Reverse lexicographical order
sort.Sort(sort.Reverse(sort.StringSlice(names)))
// Sort by length
sort.Slice(names, func(i, j int) bool {
    return len(names[i]) < len(names[j])
})
```

Web Server

```python
from flask import Flask

app = Flask(__name__)


@app.route('/')
def index():
    return 'Hello Python'


if __name__ == '__main__':
    app.run(port=8080)
```

```go
package main

import (
    "fmt"
    "log"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello Go")
}

func main() {
    http.HandleFunc("/", handler)
    if err := http.ListenAndServe(":8080", nil); err != nil {
        log.Fatal(err)
    }
}
```

HTTP Request

```python
url = 'https://httpbin.org/ip'
try:
    with urlopen(url) as fp:
        reply = json.load(fp)
except HTTPError as err:
    msg = 'error: cannot get {!r} -
{}'.format(url, err)
    raise SystemExit(msg)
except ValueError as err:
    msg = 'error: cannot decode reply -
{}'.format(err)
    raise SystemExit(msg)

print(reply['origin'])
```

```go
url := "https://httpbin.org/ip"
resp, err := http.Get(url)
if err != nil {
    log.Fatalf("error: can't get %q -
%s", url, err)
}
defer resp.Body.Close()
dec := json.NewDecoder(resp.Body)
var reply struct {
    Origin string `json:"origin"`
}
if err := dec.Decode(&reply); err !=
nil {
    log.Fatalf("error: can't decode
reply - %s", err)
}
fmt.Println(reply.Origin)
```

Encode/Decode JSON

```python
data = '''{
    "name": "bugs",
    "age": 76
}'''
obj = json.loads(data)

json.dump(obj, stdout)
```

```go
// We can also use anonymous struct
type Loon struct {
    Name string `json:"name"`
    Age  int    `json:"age"`
}

// ...

var data = []byte(`{
    "name": "bugs",
    "age": 79
}`)
loon := Loon{}
if err := json.Unmarshal(data, &loon); err != nil {
    return err
}
enc := json.NewEncoder(os.Stdout)
if err := enc.Encode(loon); err != nil {
    return err
}
```

Print Object for Debug/Log

```python
daffy = Actor(
    name='Daffy',
    age=80,
)
print(f'{daffy!r}')
```

```go
daffy := Actor{
    Name: "Daffy",
    Age:  80,
}
fmt.Printf("%#v\n", daffy)
```

Object Oriented

```python
class Cat:
    def __init__(self, name):
        self.name = name

    def greet(self, other):
        print("Meow {}, I'm
{}".format(other, self.name))

# ...

grumy = Cat('Grumpy')
grumy.greet('Grafield')
```

```go
type Cat struct {
    Name string
}

func NewCat(name string) *Cat {
    return &Cat{Name: name}
}

func (c *Cat) Greet(other string) {
    fmt.Printf("Meow %s, I'm %s\n",
other, c.Name)
}

// ...

c := NewCat("Grumpy")
c.Greet("Grafield")
```