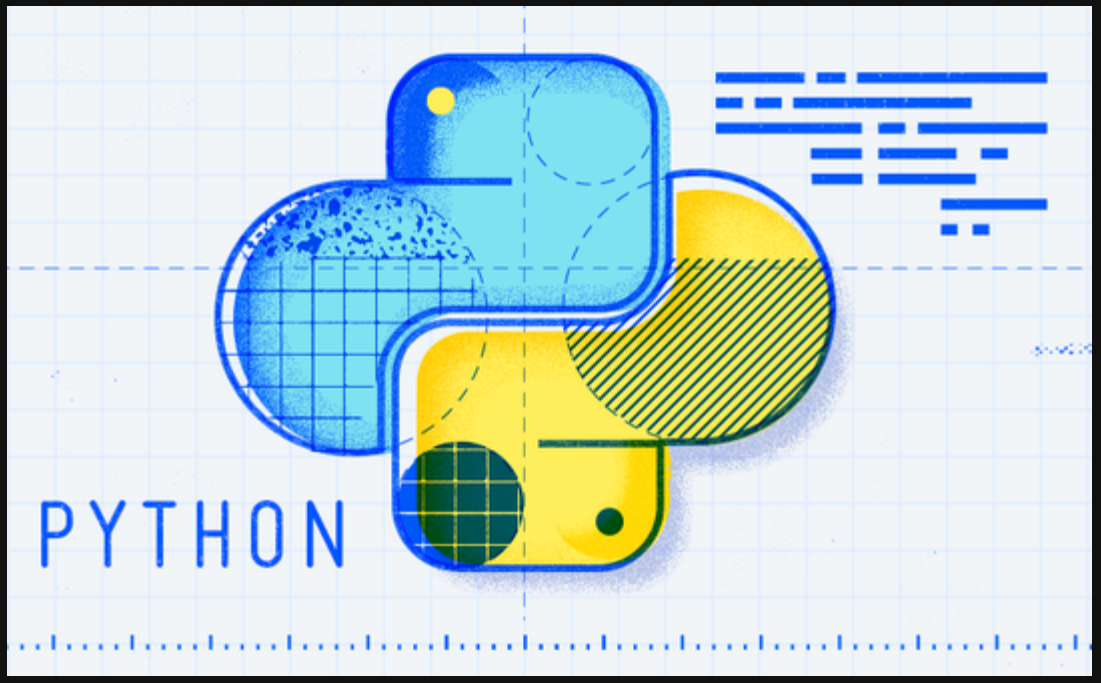
Programming with Python

Getting Started

SHELL



interpreter 📢

An interpreter is a kind of program that executes other programs

running

reads and executes commands interactively

```
$ python
Python 3.13.2 (main, Feb  5 2025, 08:05:21) [GCC 14.2.1 20250128] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

exit

```
>>> exit()
$
```

Shortcuts

- Control-D on *nix 🐧
- Control-Z on Windows 🪟

comments

code are for 💻 computer

```
# this is the first comment
# and this is the second comment
"# This is not a comment because it's inside quotes."
```

comment are for humans 🤷

calculator

```
>>> 1
1
>>> 3 + 5
8
>>> 6 * 7
42
>>> 2.2 * 3
6.600000000000005
>>> round(2.2 * 3, 2)
6.6
```

advance calculator

```
>>> pow(2, 3)
8
>>> 2 ** 3
8
>>> 22 / 7
3.142857142857143
>>> 22 // 7
3
>>> 22 % 7
1
>>> divmod(22, 7)
(3, 1)
>>> 3 * 7 + 1
22
```

scientific notation

```
>>> 1e-2
100.0
>>> 1e0
1.0
>>> 1e-1
0.1
>>> 1e-4
0.0001
>>> 1e-5
1e-05
```

summary

1. statements execute from top to bottom
2. comments are ignored

VARIABLES

place to store information

value

```
>> v
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'v' is not defined
```

```
v = None
```

```
v = 0
```

```
v = 100
```

print 

```
var = 1
print(var)
print(var, var)
```

```
1
1 1
```

or directly

```
print('hi!')
print()  # creates new line
print(1)
```

```
hi!

1
```

delete-variable

```
v = 1
print(v)
del v
print(v)
```

data-types

showing default values

```
# numbers
print(int())
print(float())
print(complex())

# strings
print(str())

# logic
print(bool())
```

```
0
0.0
0j

False
```

duck typing 🦆

`type()` function tell what type value is

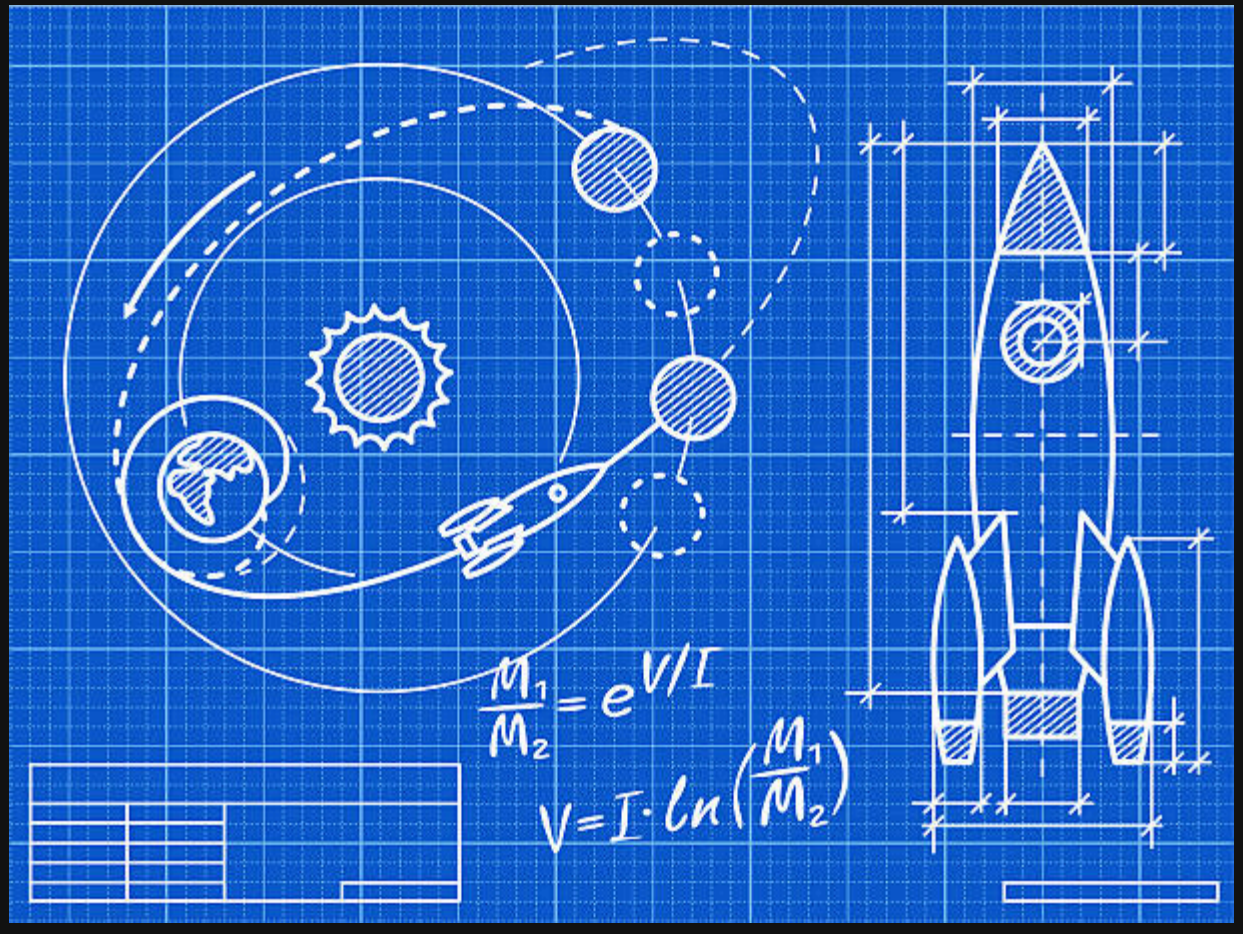
```
1 print(type(1))
2 print(type(1.0))
3 print(type(1j))
4 print(type(True))
5 print(type('1'))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
<class 'str'>
<class 'bool'>
```

dynamic-typing

```
v = 5
v = 5.0
v = 5j
v = True
```

BASICS



1. input

```
variable = input('enter something: ')
print(variable)
print(type(variable))
print(len(variable))
```

```
enter something: Hello World!
Hello World!
<class 'str'>
12
```

```
s = input('say something: ')
print(s, 'has', len(s), 'charaters')
```

```
say something: Python is Easy
Python is Easy has 14 charaters
```

2. area of the circle

```
import math

print(math.pi)
```

```
enter the radius : 1
area : 3.141592653589793
```

3. value of a character

```
print(hex(10))
print(ord('A'))
print(chr(2325))
```

```
input a character: 1
dec: 49
bin: 0b110001 110001
hex: 0x31 31
```

4. range

range(start, stop, step)

```
r = range(10)
print(r)
print(list(r))
```

```
range(0, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
for i in range(3, 5):
    print('lets repeate this time', i)
```

help

```
import math
help(math)
```

press q to quit help

summary

1. using variables store values
2. read input from user
3. few built in functions

LOOPS

syntax

c-lang 😒

```c
#include <stdio.h>

int main() {
    for (int i=0; i < 5; i++)
        printf("%d\n", i);
    return 0;
}
```

```
0
1
2
3
4
```

😎 py style

```python
for c in range(5):
    print(c)
```

```
0
1
2
3
4
```

1. factorial of a number

```
enter a number: 0
1
```

2. print all Latin alphabets

```
print('A'.lower(), 'z'.upper())
```

```
a Z
```

```
# showing horizontally only for slide
Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm Nn Oo Pp Qq Rr Ss Tt Uu Vv Ww Xx Yy
```

3. place power of number

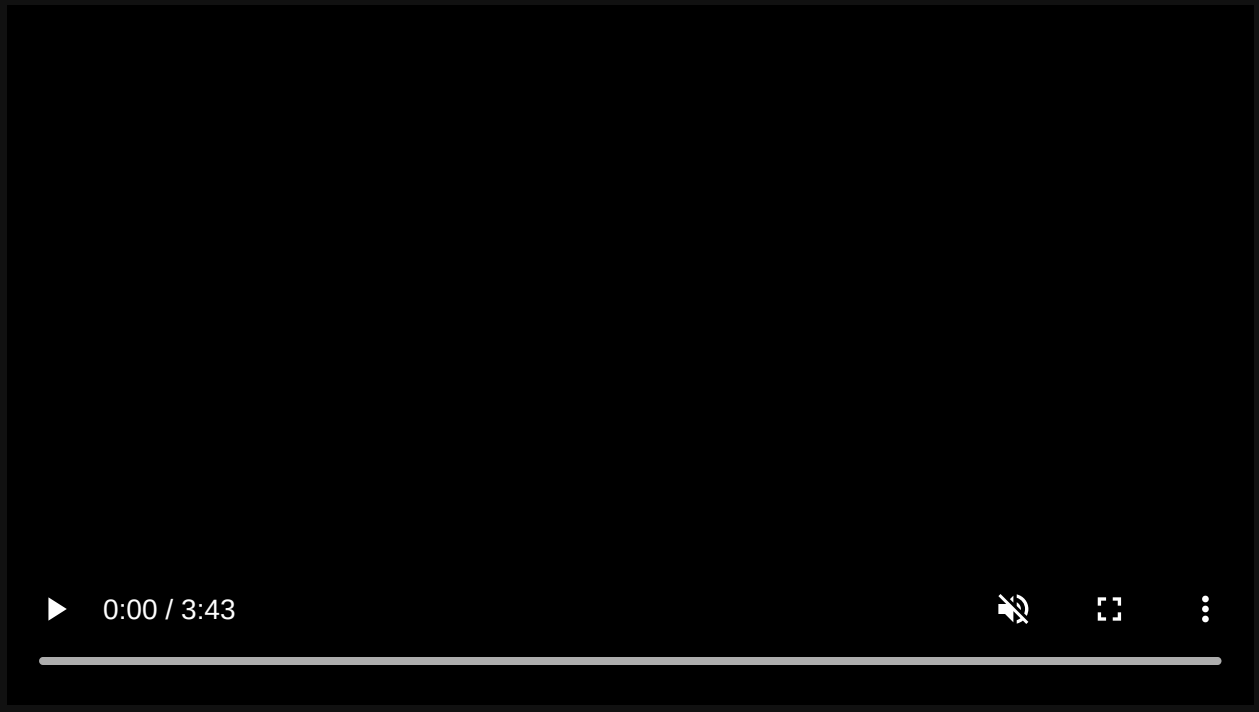
```
for c in reversed('hello'):
    print(c, end='')  # print endswith new line
```

```
for i, c in enumerate('hi'):
    print(i, c)
```

```
0 h
1 i
```

```
enter a number: 361
1 1 1
```

4. fibonacci series




```
0 1 1 2 3 5 8 13 21 34
```

5. count type of char

```
'a'.isalpha()
'1'.isdigit()
```

```
enter a phrase: this is python 3.12
alphabets: 12
numbers: 3
others: 4
```

summary

- python style loops
- use of variables
- enumerator and index

STRINGS

traverse

c style 😒

```
string = 'loops'

# (i=0; i<len(string); i++)
for i in range(len(string)):
    print(string[i])
```

```
l
o
o
p
s
```

😎 py style

```
string = 'loops'


for c in string:
    print(c)
```

```
l
o
o
p
s
```

escape

```
## escape sequence \x \t \n
print('\\t \t is tab')
print('\\x \x41 is hex value')
print('\\n \n is new line')
print('\\ can be use to escape \' string quote')
print("\\ can be use to escape \" string quote")
```

```
\t       is tab
\x A is hex value
\n
 is new line
\ can be use to escape ' string quote
\ can be use to escape " string quote
```

```
print('क', chr(0x0915), '\u0915', '\U00000915')
print('🐍', chr(0x1f40d), '\N{SNAKE}', '\U0001f40d')
print('🚀', chr(0x1f680), '\N{ROCKET}', '\U0001f680')
```

1. देवनागरी

```
print(chr(0x0915))
print(ord('ह'))
```

```
क
2361
```

```
# showing horizontally only for slide
क ख ग घ ङ च छ ज झ ञ ट ठ ड ढ ण त थ द ध न ऩ प फ ब भ म य र ऱ ल ळ ऴ व श ष स ह
```

placeholder

way to include variables in a string

```
print('using {} version {}'.format('Python', 3.11))
print('using {1} version {0}'.format(3.11, 'Python'))
```

```
using Python version 3.11
using Python version 3.11
```

```
print('{:.4f}'.format(3.12))
print('{:.1f}'.format(3.12))
print('{:05}'.format(3.12))
```

```
3.1200
3.1
03.12
```

2. multiplication table

```
print('precision {:05}'.format(1))
```

```
precision 00001
```

```
07 x 12 =  84
08 x 12 =  96
09 x 12 = 108
10 x 12 = 120
```

3. operations

```
s = 'py' + 'thon'
print(s)
print(s * 3)
s += ' '
print(s * 3)
```

```
python
pythonpythonpython
python python python
```

```
*
* *
* * *
* * * *
* * * * *
```

count

```
s = (
    'You need a programming language to program a program,\n'
    'How do you program a programming language to program,\n'
    'a program to program programs ?'
)

print(s)
print(len(s))
print(s.count('a'))  # single char
print(s.count('program'))  # multi char
```

```
You need a programming language to program a program,
How do you program a programming language to program,
a program to program programs ?
139
17
9
```

4. vowel count

```
enter a phrase: vowels
vowels: 2
```

types

```
print('क')  # unicode
print(r'he\t\nllo\n')  # raw form
print(b'hello')  # bytes stream
```

```
क
he\t\nllo\n
b'hello'
```

```
print('C:\some\path')   # here \n means newline!
print('C:\\fix\\with\\escape\\sequence') # normal fix
print(r'C:\fix\with\raw')  # note the r before the quote
```

```
C:\some\path
C:\fix\with\escape\sequence
C:\fix\with\raw
```

create

```
1 single = '🥇 can contain "double" quote'; print(single)
2 double = "🥈 can contain 'single' quote"; print(double)
3 triple = """🥉 can contain multi-line which can contain
4 'single' or "double" quotes and triple single '''
5 """
6 print(triple)
7 print(''' wrapped in """single""" quote ''')
```

```
🥇 can contain "double" quote
🥈 can contain 'single' quote
🥉 can contain multi-line which can contain
'single' or "double" quotes and triple single '''

 wrapped in """single""" quote
```

summary

- what are string
- string types
- escape sequences
- basic string methods
- string formatting

CONDITIONS

syntax

```
i = 0
print(i == 0)
print(1 < False)
```

```
True
False
```

```
if 1 != 1.0:  # True
    print('values are different')
else:
    print('values are equal')
```

```
values are equal
```

1. even or odd

HINT: even number is divisible by 2

```
enter a number: 33
odd
odd
```

else if ladder

c style 😒

```
if x == 0:
    print('zero')
else:
    if x == 1:
        print('one')
    else:
        if x == 2:
            print('two')
        else:
            if x == 3:
                print('three')
```

😎 py style

```
if x == 0:
    print('zero')
elif x == 1:
    print('one')
elif x == 2:
    print('two')
elif x == 3:
    print('three')
```

```
# single statement can be put in same line
if   x == 0: print('zero')
elif x == 1: print('one')
elif x == 2: print('two')
elif x == 3: print('three')
```

2. calculate division

<33: fail; **33-60**: 2nd; **60-80**: 1st; **>80**: distinction

```
subjects = ['Math', 'Science', 'Computer']
marks = [80, 77, 95]
print('{} %'.format(90))
```

```
percentage: 84.0 %
distinction
```

3. count letter frequency

```
import string
print(*string.ascii_lowercase)
count = [0] * 26  # similar like in string
count[1] = 5  # changing the value
```

```
phrase: the quick brown fox jumps over the lazy dog
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

4. print patterns

EXTRA: 😭 draw 'X' and 'Z' pattern

summary

- if, else, elif syntax
- ternary operator

INDEX


We need to talk
OK
RUN CMD
You seem to have more time for your computer than me. I want to know how important I am to you
You're my number 1 in my life
Just that I start counting from 0

list

```
a = [[1, 2], 'abc',  3.14, True, None]  # all types possible
print(a)
print(type(a))
print(a[0])  # element in first index
print(a[0][1])
```

```
a = [1]
a.append(2)
a.extend([3, 4])
print(a)
a.insert(0, 0)  # insert in first position
print(a)
```

```
[1, 2, 3, 4]
[0, 1, 2, 3, 4]
```

first and last


```
s = 'string'
print('first:', s[0])
print('length', l:=len(s))
print('last:', s[l-1], s[-1])
```

```
first: s
length 6
last: g g
```

1. big-ones

How many 2-digit numbers have tens digit smaller than it unit digit i.e suppose for number 56, 5 is tens position, 6 is ones position 5 < 6

[12, 13, 14, 15, 16, 17, 18, 19, 23, 24, 25, 26, 27, 28, 29, 34, 35, 36, 3
36

split

```
s = 'string\nslicing and dicing'
print(s.split())  #
print(s.split('\n'))
print(s.splitlines())
```

```
['string', 'slicing', 'and', 'dicing']
['string', 'slicing and dicing']
['string', 'slicing and dicing']
```

2. slice ✂️

```
s = 'select a sub string'
print(i:=s.index('sub'))
print(s[i:i+3])  # slice

# start stop step
a = list(range(10))
print(a)
print(a[0:-1:2])
```

```
9
sub
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8]
```

get the odd Index

3. vowels frequency

```
s = 'string'
print(i:=s.index('g'))
print(s[i])
```

```
5
g
```

```
print('c' in 'character')
print('x' in 'character')
```

```
True
False
```

```
enter a phrase: communicate
a e i o u
```

4. palindrome

```
# variable[start, stop, interval]
print('1234'[::-1])
```

```
4321
```

रहर, मलम, नयन, hannah, racecar

```
given: racecar
palindrom: True
palindrom: Yes it is
```

5. palindrome-number

show all 2-digit numbers palindrome in base 2, 8

```
27 0b11011 0o33
45 0b101101 0o55
63 0b111111 0o77
65 0b1000001 0o101
73 0b1001001 0o111
5
```

summary

datatype	contains	mutable
string	character	❌
tuple	anything	❌
list	anything	✅

WHILE LOOP

AND PROGRAM LOSES ITS MIND

syntax

indefinite iteration

```
i = 0
while i < 5:
    print(i)
    i += 1
```

1. infinity and beyond

numbers accumulator

```
enter any number : 5
sum: 5.0
enter any number : 3
sum: 8.0
...
..
```

Ctrl + c to manually interrupt the loop

2. quit on 'q'

```
for i in range(5):
    continue
print(i)
```

```
4
```

```
for i in range(5):
    break
print(i)
```

```
0
```

```
enter any number (quit=q): 5
sum: 5.0
enter any number (quit=q): q
```

3. ⚠ invaild input

```
try:
    1/0
except Exception as e:
    print('handle:', e)
```

```
enter any number (quit=q): 5
sum: 5.0
enter any number (quit=q): a
enter any number (quit=q): 2
```

4. average

```
enter any number (quit=q): 5
sum: 5.0
enter any number (quit=q): 10
sum: 15.0
enter any number (quit=q): q
avg: 7.5
```

5. min and max

```
l = list()
l.append(1)
```

```
l = [3, 1, 4, 2, 8]
min(l); max(l)
```

```
enter any number (quit: q): -5
sum: -5.0
enter any number (quit: q): 6
sum: 1.0
```

6. min and max without list

```
enter any number (quit: q): -5
sum: -5.0
enter any number (quit: q): 6
sum: 1.0
```

summary

- un-deterministic loop
- flow control
- exception handling
- float ranges

FUNCTIONS

built in functions

Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

Built-in Functions

A
abs()
aiter()
all()
anext()
any()
ascii()

B
bin()
bool()
breakpoint()
bytearray()
bytes()

C
callable()
chr()
classmethod()

E
enumerate()
eval()
exec()

F
filter()
float()
format()
frozenset()

G
getattr()
globals()

H
hasattr()
hash()
help()

L
len()
list()
locals()

M
map()
max()
memoryview()
min()

N
next()

O
object()
oct()
open()
ord()

R
range()
repr()
reversed()
round()

S
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

T
tuple()
type()

user-defined

```
def functionName(): # defining the function
    print('doing common things')

functionName()             # calling the function
print(functionName)        # function bluprint
print(type(functionName)) # checking type
functionName()             # calling again
print('yes, We can reuse it many times!')
```

```
doing common things
<function functionName at 0x71405dfea340>
<class 'function'>
doing common things
yes, We can reuse it many times!
```

1. draw a flag 

```
*
* *
* * *
* * * *
*
* *
* * *
* * * *
```

2. argument

```
def foo(argument):
    print(argument)

foo(3)
foo(4)
```

```
*
* *
* * *
*
* *
```

3. argument defaults

```
def foo(x, y=2, z=3):
    print(x, y, z)

foo(1)              # x required and positional
foo(4, 5, z=6)      # y and z is optional
foo(z=7, y=9, x=8)  # explicit position
print('\U+1F319')
print(ord(🌙))
```

return

```
def foo():
    pass  # doing nothing just for syntax

def boo():
    return None

print(foo(), boo())
```

```
def moo():
    return 'moo'

print(type(moo))
print(type(moo()))
m = moo()
print(m)
```

```
<class 'function'>
<class 'str'>
moo
```

4. caesar cipher

```
def ceasor(c, shift):
    NotImplemented

for c in 'Tcxlsr 3 mw EAIWSQI!!':
    print(ceasor(c, -4), end='')
```

Python 3 is AWESOME!!

5. human readable

```
import os

for file in os.scandir():
    if file.is_dir(): continue
    size = os.path.getsize(file)
    print(truncate(file.name, 20), readable(size))
```

```
03 pythonic.org  50.05 Kb
  02 start.html  42.45 Kb
         org.py   1.31 Kb
   02 start.org  51.68 Kb
```

summary

1. reusable units
2. changable parameters

PROJECT 01

matters of life and death