Python Cheat Sheet

Python Basic

Data Types

Type	E.g.	Desc
str	#"Hello"	text data
int	#67,-106	whole number
float	#3.14	decimal number
bool	#True,False	True/False
list	#[1,"A",True]	ordered,mutable
dict	#{"name":"Alice"}	key-value pairs
tuple	#(10,20)	ordered,immutable
Set	#{1,2,3}	unique,unordered
None	#None	no value

You can use the **type()** built-in function which return the type of variable to check a data type of any variable

Comments

Single line - #

Multi line - ''' '''

E.g.

```
# This is a comment
'''
This comment is
multi line
'''
```

Casting

Use **Casting** to change a value type

E.g.

```
int("5")  #"5"→5
str(0.1)  #0.1→ "0.1"
list("hi")#["h","i"]
```

Input/Output (I/O)

Output - Display

```
print("Age:",21) #Basic output

pi = 3.14159
print(f"Pi is {pi}") #f-string format

print("A","B",sep="-")#A-B, sep goes between multiple values

print("Hello", end="!")#Hello!, end defines what is at the end of a line
```

Input - Read User Input

```
name = input("Enter your name: ")
#By default always return a string
#Use casting to return other types of input

number=int(input("Enter your age:"))

#Now the it will return an integer after user input a value
```

Variables

Naming Conventions

- **Snake case**

All lowercase, words seprated by _

```
student_age = 18
```

- **Pascal case**

All words capitalized, no _

```
StudentAge = 18
```

- **Camel case**

First word lowercas afterward uppercase

```
studentAge = 18
```

- **Uppercase**

All uppercase, _ between each words

```
STUDENT_AGE = 18
```

Naming Rules

- Variable naming must start with either letter or underscore
- Variable can contain letters, numbers and underscores
- Beware of case-sensitivity like age != Age
- Avoid reserved keywords like (if , for , class , ...)
- Be descriptive when naming like userAge > ua

Functions

- **No Return Type**

```
def function_name(parameter/no parameter):
    # statement
```

- **Return Type**

```
def function_name(parameter/no parameter):
    # statement
    return expression
```

- **Function Calling**

```
function_name(argument/no argument)
variable = function_name(argument/no argument)
```

E.g.

```
def calc(a , b=0):
        return a+b

result = calc(3 , 6)
print(f"Result is {result}")
```

Loop

- **For Loop**

```
for X in x:
    # code
```

- **While Loop**

```
while (condition):
    # code
```

Loop ControlKeywords:

break: exit loop immediately

continue: skip to the next iteration

Operators

Arithmetic

Operator	Name
x + y	Addition
x - y	Subtraction
x * y	Multiplication
x / y	Division
x // y	Floor Division
x % y	Modulus
x ** y	Exponentiation

Comparison

Operator	Name
==	Equal
!=	Not Equal
>	Greater than
>=	Greater or Equal
<	Lesser than
<=	Lesser or Equal

Assignment

Operator	E.g.
x = y	x = 5
x += y	x += 2
x -= y	x -= 2
x *= y	x *= 2
x /= y	x /= 2
x %= y	x %= 2

Logical

Operator	E.g.
and	x>5 and x<7
or	x>5 or x<7
not	not(x>5 and x<7)

Identity & Membership

Operator	E.g.	Operator	E.g.
is	x is y	in	x in y
is not	x is not y	not in	x not in y

Error Handling

```
try:
    # Attempt risky code
except ErrorType:
    # Handle error
else:
    # Run if no error
finally:
    # always runs
```

Error Types:

- **ValueError:** wrong value (int("abc"))
- **TypeError**: wrong type ("5" + 2)
- **ZeroDivisionError:** divide by zero
- **FileNotFoundError:** missing file
- **IndexError:** bad list index
- **KeyError:** bad dict key

Conditional Statement

If/Elif/Else

```
if (condition):
     # code
elif (condition):
     # code
else:
     # code
```

E.g. x = 10

```
if (x > 10 and x < 20):
     print(x - 20)
elif (x = 10)
     print(x)
else:
     print(x + 20)
```

if/elif/else: evaluates conditions

Match: matches exact values or patterns

Match

```
match value:
     case pattern1:
          '''
          code
          if pattern1 = value
          '''
     case pattern2:
          '''
          code
          if pattern2 = value
          '''
     case_:

          # Default
```

E.g. x = 2

```
match x:
     case 5: print("Equal")
     case 10: print("Big")
     case _: print("Other")
```

String Basics

Concatenation - "Hi" + "!!" : Hi!
Repetition - "Ha" * 3 : HaHaHa

```
s = "Python"
```

- **Indexing & Slicing**

```
s[0] = "P"        # first char
s[-1] = "n"       # last char
s[0:3] = "Pyt"    # slice
s[0::2] = ""Pto   # step
```

- **String Methods**

- **s.upper():** all uppercase
- **s.lower():** all lowercase
- **s.strip():** remove whitespace
- **s.split(""):** separate into list
- **s.replace("old","new"):** replace text
- **len(s):** length of string
- **s.count(""):** count occurences
- **s.find(""):** find index of substring
- **s.isalpha():** only letters
- **s.isdigit():** only digits

Collection Types

Tuples

Tuples are ordered, immutable and allow duplication

Syntax: `t = ("Damn",21,"dog")`

Use: Fixed data

Method:
- **t.count():** count occurences
- **t.index():** find position

Lists

Lists are ordered, mutable and allow duplication

Syntax: `l = ["A","B","C]`

Use: Editable sequence

Method:
- **l.append():** add
- **l.remove():** delete
- **l.pop():** remove last
- **l.sort():** sort
- **l.reverse():** reverse order

Sets

Sets are unordered, mutable and no duplication

Syntax: `s = {"X","Y","Z"}`

Use: Unique collection

Method:
- **s.add():** add
- **s.remove():** delete
- **s.union():** combine
- **s.intersection():** common
- **s.difference():** subtract

Dictionaries

Disctionaries are unordered, mutable and no duplication key-value pairs

Syntax: `d = {"X":1,"Y":2}`

Use: Map key to value

Method:
- **d.pop():** remove
- **d.get():** safe access
- **d.update():** add/change
- **d.keys():** inspect keys

Class and object

Class

```
Class Cookie:
  def __init__(self, shape):
      self.shape = shape

  def eat(self):
      print("Yummy! 🍪")
```

Object

```
cookie1 = Cookie("Star")
cookie2 = Cookie("Heart")
```

Constructor

Constructor a is a special method that run automaticlly when an object is created.

E.g.

```
Class Person:
  def __init__(self, name, age):
      self.name = name
      self.age = age
```

File Handling

- **Open & Close**

```
f = open("file_Name", "Mode") #openfile
f.close()   # close file
```

- **Modes**

- "r" : read
- "w" : write (overwrite)
- "a" : append
- "x" : create new
- "b" : binary

- **With Statement**

```
with open("File_Name", "r") as f:
    data = f.read()
```

with Statement automatically closes the file when the block ends

Read

```
f.read()   # all content
f.readline()  # one line
f.readlines()  # list of lines
```

Write

```
f.write("Hello!")  # write text
```

Group 2 - Team 2 Members

- **Heng Sovannreach**
- **Yann LaiE**
- **Born Chansothearith**

References

- **W3Schoools**
- **learnpython.org**
- **Youtube - Bro Code**