Python Cheatsheet

Key Structures

- Sequence
- Variables and Assignment
- Boolean and arithmetic expressions (operators and comparators)
- Control Flow
- Looping
- Functions/Routines

Variables

```
lvalue = rvalue
```

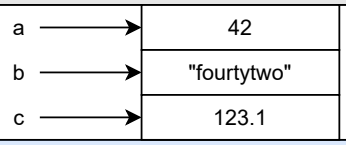
`lvalue` is a name

`rvalue` is an expression

Name should start with letter or underscore, can contain letters, underscores, or digits

Assignment

```
a = 42
b = "fourtytwo"
c = 123.1
```



Objects

Any piece of data is called an **object**, stored in **memory**, and a **variable** is an alias to that object.

All objects have:

- Identity
- Type
- Attributes
- 0+ aliases (variables)

Expressions

```
42
x + 2
x > 4
```

Literal: a literal Python value hard coded in the code

Operator: An arithmetic or logical operation
Comparator: A comparison operator, usually resulting in a boolean

Types

```
type(<expression>)
```

Some basic types:

- **`int`**: integers
- **`float`**: floating point (decimal) numbers
- **`str`**: strings, a sequence of characters
- **`bool`**: boolean value **`True`** or **`False`**

Operators / Comparators

- <,>,<=,>=,==,!=
  lt, gt, lte, gte, equal, not equal
- +,-,*,/
  add, subtract, multiply, divide
- % modulus (remainder)
- ** exponentiation
- // integer division (integer quotient)

None

None is a special type that means "nothing"

Control Flow

```
if EXPRESSION:
    indented block executed if EXPRESSION is True

    Note that the indentation is required,
    and that EXPRESSION can represent more
    than one individual expression
Program continues here
```

```
if EXPRESSION:
    indented block executed if EXPRESSION is True
else:
    indented block executed if EXPRESSION is False
Program continues here
```

```
if EXPRESSION:
    indented block executed if EXPRESSION is True
elif EXPRESSION_2:
    indented block executed if EXPRESSION is False
    and EXPRESSION_2 is True
elif EXPRESSION_3:
    indented block executed if EXPRESSION is False,
    EXPRESSION_2 is False, and EXPRESSION_3 is True
else:
    indented block executed if all EXPRESSIONs
    are False, i.e. the "Default" case
Program continues here
```

Looping

```
while (EXPRESSION):
    if EXPRESSION is true, execute indented block
    Repeat until the EXPRESSION evaluates to false
Program continues here
```

```
for item in ITERABLE:
    item will be the next element in ITERABLE
    on each subsequent loop
Program continues here
```

`break`	Immediately exit the current loop
`continue`	Immediately jump to the next iteration

Functions

```
def foo(a, b=1, *args, **kwargs):
    print(args, kwargs, a + b)
foo(1)                    # (), {}, 2
foo(a=1)                  # (), {}, 2
foo(b=1, a=2)             # (), {}, 3
foor(1, 2, 3, 4)          # (3, 4), {}, 3
foo(a=1, b=2, c=3, d=4) # (), {"c":3,"d":4}, 3
```

- Accept 0 or more **arguments**
- **Return** 0 or more return values
- Return immediately returns from function
- Arguments passed by **position** or **keyword**
- Arguments can have **default values**
- **Variable positional** arguments defined with * "packed" into a tuple
- **Variable keyword** arguments defined with ** "packed" into a dictionary
- If no return, return value is **None**

Global Functions

- `type(expr)`: Get the **type** of the expression/variable/literal
- `id(expr)`: Get the **identity** of the expression/variable/literal
- `dir(expr)`: Get the **attributes** of the type of the expression/variable/literal
- `print(str)`: Print the input
- `input(str)`: Print the input and prompt the user to type in input (returns a `str`)

Lists

```
x = [1, 2, 3]
x.append(4)            # [1, 2, 3, 4]
for elem in x:
    print(elem)
for i in range(len(x)):
    x[i] = x[i] + 1
print(x)               # [2, 3, 4, 5]
```

`append(item)`: add item to end
`extend(items)`: add all items to end
`insert(pos, item)`: insert item at position
`remove(item)`: remove item by value
`pop(index)`: remove item by position
`clear()`: remove all items
`sort()`: sort list in place
`index(item)`: get position of item
`copy()`: create a copy of the list

Dicts

```
x = {"a": 1, "b": 2, "c": 3}
x["d"] = 4
for key in x:
    print(x[key])
for k, v in x.items():
    x[k] = v + 1
print(x)
# {"a":2, "b":3, "c":4, "d":5}
```

`items()`: iterate through key, value
`keys()`: get keys
`values()`: get values
`get(key, default)`: get value by key, return default if missing
`pop(key)`: pop value by key
`update(dct)`: update with another dict
`copy()`: create copy of the dict

Sets

```
x = {"a", "b", "c"}
x.add("a")  # {"a", "b", "c"}
print(x.intersection({"a", "b"}))
# {"a", "b"}
print(x.union({"d"})
# {"a", "b", "c", "d"}
print(x - {"a", "b"})
# {"c"}
```

`add(item)`: add item to set
`remove(item)`: remove item from set
`union(s)`: union with set s
`intersection(s)`: intersect with set s
`difference(s)`: difference with s
`copy()`: make a copy of the set

Imports

```
import math
print(math.sin(math.pi / 2 ))
import math as MATH
print(MATH.sin(MATH.pi / 2 ))
from math import sin, pi
print(sin(pi / 2))
from math import sin, pi as PI
print(sin(PI / 2))
```

Files

```
fp_in = open("in.txt", "r")
fp_out = open("out.txt", "w")
for line in fp_in:
    fp_out.write(line)
fp_in.close()
fp_out.close()
```