Save a copy

Handout: Python cheat sheets

Introduction

This is a reference handout for the Python elements covered in this unit. The sheets include short explanations, brief notes, syntax, and selected examples.

The content has been grouped into categories:

- Variables, assignments, operators, and expressions
- Output and input
- Libraries: randomness and time
- Selection
- Iteration

National Centre for Computing Education

Output

The `print` function displays literals (e.g. numbers, text) and the values of variables and expressions.

Syntax

```
print(comma-separated literals, variables, expressions)
```

Examples

```
print("Hello world")
```

Display the string literal `"Hello world"`

```
print("Hello", user)
```

Display a string literal and the value of the `user` variable

```
print(x, "times two is", 2*x)
```

Display, among others, the value of the expression `2*x`

Input

The `input` function reads a line of text from the keyboard and **returns** it.

Syntax

```
input()
```

Notes

Assign the value returned by `input` to a variable, if you need to refer to that value later in your program.

Use the `int` function to convert the text returned by `input` to an integer.

Use the `float` function to convert the text returned by `input` to a floating-point number.

Examples

```
name = input()
```

Read text from the keyboard and assign it to the `name` variable

```
years = int(input())
```

Read text from the keyboard, convert it to an integer, and assign it to the `years` variable

```
input()
```

Read text from the keyboard and discard it (useful for pausing execution until Enter is pressed)

National Centre for Computing Education

Assignment

An assignment statement evaluates an expression and associates its value with the name of a variable (an identifier).

Syntax

```
variable name = expression
```

Notes

Do not interpret the = sign as an equation. Assignments are actions to be performed.

Read assignments from right to left, i.e. evaluate the expression and then assign the value to the variable.

A variable name can only refer to a single value. A new assignment to a variable **replaces** the previous value of the variable.

Examples

```
name = "Ada"
```

Assign the string literal `"Ada"` to the `name` variable

```
days = 365*years
```

Evaluate the expression `365*years` and assign the value to the `days` variable

```
dice = randint(1,6)
```

Call the `randint` function and assign the value it returns to the `dice` variable

```
count = count+1
```

Evaluate the expression `count+1` and assign the value to `count`, i.e. increase `count` by 1

```
a = 2*a
```

Evaluate the expression `2*a` and assign the value to `a`, i.e. double the value of `a`

National Centre for Computing Education

Operators and expressions

Arithmetic

Perform calculations with numbers. The result of these operations is also a number.

Addition:	`+`
Subtraction:	`-`
Multiplication:	`*`
Division:	`/`
Integer division:	`//`
Remainder:	`%`
Exponent:	`**`

Notes

Logical expressions evaluate to either `True` or `False`.

'Logical expression' is a synonym for **condition**. To evaluate a logical expression is to check a condition.

Relational (comparisons)

Compare the values of expressions. The result of these operations is either `True` or `False` (so relational operators form logical expressions).

Equal to:	`==`
Not equal to:	`!=`
Less than:	`<`
Less than or equal to:	`<=`
Greater than:	`>`
Greater than or equal to:	`>=`

Logical

Negate or combine logical expressions. The result of these operations is either `True` or `False`.

Negation:	`not`
Conjunction:	`and`
Disjunction:	`or`

Examples

`3 + 13 * 3`	An arithmetic expression involving operators and literals
`2**8 - letters - numbers - symbols`	An arithmetic expression involving operators, literals, and variables
`applications <= positions`	A logical expression, comparing the values of two variables
`a + b == c - d`	A logical expression, checking if the values of two expressions are equal
`user != "Ada" and logins < 3`	A logical expression, which is the conjunction of two simpler logical expressions

National Centre for Computing Education

Modules

Modules are libraries of existing code.

They extend the functionality of the language by offering components (such as functions) that can be imported and used in programs.

Syntax

```
from variable import component
```

Note

It is standard practice that you place all `import` statements at the beginning of the program.

Examples

The `random` module

docs.python.org/3/library/random.html

Provides functionality for generating random numbers

```
from random import randint
dice = randint(1,6)
```

Call the `randint` function to generate a random integer from 1 to 6 and assign the value that it returns to the `dice` variable

```
from random import randint
coin = randint(0,1)
```

Call the `randint` function to generate a random integer from 0 to 1 and assign the value that it returns to the `coin` variable

The `time` module

https://docs.python.org/3/library/time.html

Provides functionality for time and date handling

```
from time import sleep
sleep(3)
```

Call the `sleep` function to pause program execution for 3 seconds

```
from time import localtime
year = locatime().tm_year
```

Use the `localtime` function to retrieve the current year and assign it to the `year` variable

National Centre for Computing Education

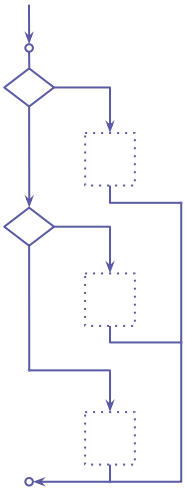
Selection

The `if` statement creates **branches** in the flow of program execution.

At runtime, a **condition** or a sequence of conditions are checked, to **select** which one of the possible branches will be followed.

Syntax

```
if condition:
    block of statements
    (the if block)
elif condition:
    block of statements
    (an elif block — optional, there may be many)
else:
    block of statements
    (the else block — optional)
```



Notes

Out of the different blocks of statements contained in a selection structure, **at most one** block will be executed at runtime.

The blocks of statements can contain **nested** `if` and `while` statements.

Examples

```
if dice1 == dice2:
    print("A double roll")
    total = 4*sum
else:
    total = sum
```

Check if the values of the `dice1` and `dice2` variables are equal and perform the appropriate actions, depending on the outcome

There are two possible, mutually exclusive branches.

```
if temperature < 4:
    print("Freezing")
elif temperature < 18:
    print("Tolerable")
else:
    print("Nice and warm")
```

Check the range in which the value of the `temperature` variable lies and print an appropriate message, depending on the outcome

There are three possible, mutually exclusive branches.

```
max = x
if y > max:
    max = y
if z > max:
    max = z
```

Compute `max`, the greatest value among `x`, `y`, and `z`

These `if` statements compare `y` and `z` to the current `max` and raise `max`, if necessary.

Without an `elif`, the two `if` statements are not mutually exclusive.

National Centre for Computing Education

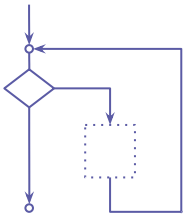
Iteration

The `while` statement creates a **loop** in the **flow** of program execution.

At runtime, a set of actions is repeated and a **condition** is checked to determine if the loop should continue.

Syntax

```
while condition:
    block of statements
    (the while block)
```



Notes

The block of statements in the iterative structure may be executed many times, once, or even not executed at all (if the `while` condition is `False` when it is first checked).

The block of statements can contain **nested** `if` and `while` statements.

Examples

```
# display a count from 1 to 10
count = 1
while count <= 10:
    print(count)
    count = count+1
```

Repeat the indented block of statements while `count` does not exceed 10

```
print("What is your name?")
name = input()
# only take "Ada" for an answer
while name != "Ada":
    print("I was expecting Ada")
    print("What is your name?")
    name = input()
# end of loop, welcome user
print("Welcome")
```

Repeat the indented block of statements while `name` does not equal `"Ada"`

```
non_zero = True
while non_zero == True:
    a = int(input())
    if a != 0:
        # display inverse of a
        print(1/a)
    else:
        non_zero = False
```

Repeat the indented block of statements while the Boolean flag `non_zero` remains `True`