Python Documentation

Getting Started

Every program you write with us will always include the following code to define and execute a `main()` function:

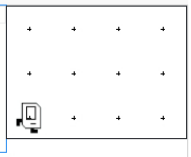
```
def main():
    # Your code goes here

if __name__ == '__main__':
    main()
```

```
from karel.stanfordkarel import *

def main():
    move()
    move()

if __name__ == '__main__':
    main()
```



▶ Run

Variables

Assigning a Variable

```
variable_name = value
```

Accessing the Current Value of a Variable

```
message = "Hello, World!"

# message is replaced with its current assigned value, which is "Hello, World!"
print(message)
```

Variable Types

Type	Stores	Example
`int` (integer)	Whole numbers	`137`
`float` (floating point)	Decimal numbers	`137.0`
`str` (string)	text/character sequence between `“”` or `‘’`	`"Hello, World"` *Note: The quotes won't appear when printing a string*
`bool` (boolean)	`True` or `False`	`True`

Print

Printing a string literal

```
# Remember to surround your message in single '' or double "" quotes
print("Hello, World!")
```

▶ Run >_ Show

Printing a variable

```
# Change the value of data and see what happens
data = 137.2

print(data)
```

▶ Run >_ Show

Multi-Argument Print:

```
name = "Kyler"

# Python automatically separates each argument with a space
print("Hello, my name is", name)
```

▶ Run >_ Show

Input

Storing user input as a variable

```
user_input = input("This Message Will Is Displayed When Prompting The User For Input")
```

Storing numerical inputs

```
# input always returns a string by default, so you have to convert numerical inputs manually
integer_input = int(input("Enter an integer:"))

float_input = float(input("Enter a floating point value:"))
```

Operators

Arithmetic Operations

Operation	Symbol	Example
Addition	`+`	`>>> 20 + 11` `31`
Subtraction	`-`	`>>> 17 - 23` `-6`
Multiplication	`*`	`>>> 15 * 12` `180`
Division	`/`	`>>> 10 / 2` `5.0`
Integer Division	`//`	`>>> 10 // 4` `2`

Integer Division	//	2

Exponentiation	**	>>> 2 ** 3
		8
		>>> 2 ** -1
		0.5

Modulus	%	>>> 8 % 3 2

Negation	-	>>> x = 0.5
		>>> -x
		-0.5

Comparison Operations

Comparison Operator	Meaning	True Example	False Example
==	Equal	1 + 1 == 2	2 == 3
!=	Does not equal	3.2 != 3	5 - 5 != 0
<	Less than	5 < 7	2 < 1
>	Greater than	4 > 2	1 > 9
<=	Less than or equal to	90 <= 100	18.4 <= 4
>=	Greater than or equal to	5.0 >= 5.0	11 >= 20

Logical Operations

Operator	True Example	False Example
and	(3 < 5) and (-1 == -1)	(7 == 8) and (12 > 2)
or	(0 == 1) or (10 <= 15)	(0 > 5) or (6 == 3)
not	not False	not 1 > 0

Writing expressions with variables

Expressions are evaluated before assignment. The right side of the = becomes a single value which then gets assigned to the variable name.

Example Math Expression

```
age = 22
age_after_birthday = age + 1
print(age_after_birthday)
```

▶ Run >_ Show

Example Logical Expression

```
silly = True
funny = True
mean = False

if funny and silly:
    print("Nice to meet you!")

if not mean:
    print("Thank you!")
```

▶ Run >_ Show

Order of Precedence

Highest
( ) parentheses
** exponentiation
- negation
* / // %
+ -
comparison
not
and or
Lowest

Conditionals & Loops

If Statements

```
if condition:
   # Do something
```

While Loop

```
while condition:
    # Repeat this until condition is false
```

For Loop

```
for i in range(n):
    # Repeat this n times
```

Functions

Anatomy of a Function

```
def name_of_function(parameters): #This line is the function header
    # Function Body

    return value # Optional Return Statement
```

. . .

Lists

Ordered: ***Yes***

Mutable: ***Yes***

Allowed Duplicates: ***Yes***

Defining a list of known values:

```
# Individual elements are seperated by commas
name_of_list = [item1, item2, item3]
```

Defining a List of a Certain Size With Default Values

```
# This creates a list where every element is equal to initial_value. The number of
# elements in that list is equal to size
name_of_list = [initial_value]*size
```

Defining an Empty List:

```
name_of_list = []
```

Accessing Elements of a List:

```
name_of_list[index_of_element]
```

Getting a Slice of a List:

```
# A list of elements from index beginning up to but not including index end
name_of_list[beginning:end]

# A list of elements from the front of the list up to but not including index end
name_of_list[:end]

# A list of elements from index beginning to the end of the list, including the last element
name_of_list[beginning:]
```

Adding Elements to a List:

```
name_of_list.append(element)
```

Removing Elements from a List:

```
name_of_list.remove(element)
```

Checking if a List Contains an Element:

```
# Logical expression evaluates to true if name_of_list contains element
element in name_of_list
```

Finding the Index of an Item in a List:

```
# NOTE: Program will crash if element is not in name_of_list
index = name_of_list.index(element)
```

Size of a List:

```
size = len(name_of_list)
```

Dictionaries:

Ordered: ***Yes**** *(Technically, though you can treat them as if they are not)*

Mutable: ***Yes***

Allowed Duplicates: ***No***

Key-Value Pair

```
key: value
```

Defining a list of known key-value pairs:

```
# Individual key-value pairs are seperated by commas
name_of_dict = {key1: value1, key2: value2, key3: value3, ...}
```

Defining an Empty Dictionary:

```
name_of_dict = {}
```

Accessing Elements of a Dictionary:

```
name_of_dict[key]
```

Adding/Editing Elements in a Dictionary:

```
# Adds key-value pair to the dictionary if key does not currently exist
# Edits value of key if key is already in the dictionary
name_of_dict[key] = value
```

Removing Elements from a Dictionary:

```
# Deletes a key-value pair and returns value
deleted = name_of_dict.pop(key)

# Delete a key-value pair and returns nothing
```

```
# Deletes a key-value pair and returns nothing
del name_of_dict[key]

# Deletes every key-value pair and leave an empty dictionary
name_of_dict.clear()
```

Checking if a Dictionary Contains an Element:

```
# Logical expression evaluates to true if name_of_dict contains key
key in name_of_list
```

Size of a Dictionary:

```
size = len(name_of_dict)
```

Get a List of all Keys or Values:

```
# List of keys in name_of_dict
keys = name_of_dict.keys()

# List of values in name_of_dict
values = name_of_dict.values()
```

Tuples

Ordered: ***Yes***

Mutable: ***No***

Allowed Duplicates: ***Yes***

```
# your code here
my_tuple = ("A", "B", "C", "D")
print(my_tuple(1))
print()
```

▶ Run >_ Show

Defining a Tuple:

```
# Individual elements are seperated by commas
name_of_tuple = (item1, item2, item3)
```

Defining an Empty Tuple:

```
name_of_tuple = ()
```

Accessing Elements of a Tuple:

```
name_of_tuple[index_of_element]
```

Getting a Slice of a List:

```
# A list of elements from index beginning up to but not including index end
name_of_list[beginning:end]

# A list of elements from the front of the list up to but not including index end
name_of_list[:end]

# A list of elements from index beginning to the end of the list, including the last element
name_of_list[beginning:]
```

Adding Elements to a List:

```
name_of_list.append(element)
```

Removing Elements from a List:

```
name_of_list.remove(element)
```

Checking if a List Contains an Element:

```
# Logical expression evaluates to true if name_of_list contains element
element in name_of_list
```

Finding the Index of an Item in a List:

```
# NOTE: Program will crash if element is not in name_of_list
index = name_of_list.index(element)
```

Size of a List:

```
size = len(name_of_list)
```

Strings

Ordered: ***Yes***

Mutable: ***No***

Allowed Duplicates: ***Yes***

Defining a String:

```
# Use either single or double quotes
single_quote_string = 'YOUR TEXT GOES HERE'
double_quote_string = "YOUR TEXT GOES HERE"
```

Defining an Empty String:

```
# Use either single or double quotes
single_quote_string = ''
double_quote_string = ""
```

Accessing Characters in a String:

```
# Characters are actually just strings of length 1
name_of_string[index_of_character]
```

Getting a Substring:

```
# A string from index beginning up to but not including index end
name_of_string[beginning:end]

# A string from the front of the string up to but not including index end
name_of_string[:end]

# A string from index beginning to the end of the original string, including the last element
name_of_string[beginning:]
```

Checking if a String Contains a Substring:

```
# Logical expression evaluates to true if name_of_string contains substring
substring in name_of_string
```

Finding the Index of a Substring in a String:

```
# NOTE: Program will crash if substring is not in name_of_list
index = name_of_list.index(substring)
```

Length of a String:

```
length = len(name_of_string)
```

Unicode Representation of a character

```
# character must be a string of length 1
ord(character)
```

String Comparison:

```
# == checks the equality of two strings based on unicodes (Case-Sensitive)
string1 == string2

# (<, >, <=, >=) check the alphabetical order of two string based on unicodes
string1 <= string2
```