5ELEN018W - Robotic Principles
Lecture 1: Introduction to the Module and Python

Dr Dimitris C. Dracopoulos

Introduction to the Module

- Syllabus
- Lectures
- Tutorials (Practicals)
- Software
- Assessment
- Schedule
- What is expected from you?
  - → Lecture Attendance
  - → Tutorial Attendance (actual not just swiping of card!)
  - → Completion of ALL Tutorial Exercises within the week (if not possible within the tutorial session then on your own time).
  - → Raise Issues Early **directly with the Module Leader!**
  - → Code of Conduct

Code of Conduct

- Do not cheat on assignments (this is *INDIVIDUAL* work and **NOT** the product of collaboration!):
  - → Discuss only general approaches not specific details of implementation
  - → Do not take written notes on other's work and do not exchange code
- Cheating is reported to university and then it is out of the module lecturers hands (independent committee decision without the participation of the module tutors)
- Possible consequences:
  - → A mark of 0 for assignment
  - → A mark of 0 for the course
  - → A permanent note on student record
  - → Suspension/Expulsion from university

Code of Conduct (cont'ed)

- Any code found in the web or textbook and used in your work should be properly referenced in comments within your code.

Academic Integrity

- The University of Westminster is committed to the highest standards of academic integrity and honesty. Students are expected to be familiar with these standards regarding academic honesty and to uphold the policies of the University in this respect. Students are particularly urged to familiarise themselves with the provisions of the Academic Regulations and in this case with Academic Misconduct Regulations (https://www.westminster.ac.uk/sites/default/public-files/general-documents/Section-10-Academic-Misconduct-v2.pdf) and avoid any behaviour which could potentially result in suspicions of cheating, plagiarism, misrepresentation of facts and/or participation in an offence. Academic dishonesty is a serious offence and can result in suspension or expulsion from the University.

Topics

- Configuration Space
- Spatial Descriptions and Transformations
- Manipulator Kinematics
- Control
- Inverse Kinematics
- Velocity Kinematics - Jacobian
- Trajectory Generation
- Motion Planning

A very mathematical object but will try to simplify! Cannot do without maths!

Introduction to Python

Why Python?

- A very popular programming language
- Important for your knowledge and after your graduation
- Robotics toolboxes and other libraries related to robotics (vision, machine learning, algorithms) are available
- Some hardware provides their manipulation through Python APIs and libraries

Variables

No declaration of variables but types still exist:

- `int`
- `float`
- `str`
- `bool`

Example:

```
x = 5*6 + 1  %\pause%
y = "Hello class" %\pause%
type(x)   %\pause%
x = 77.9
```

Some useful functions

Built-in mathematical functions:

```
abs(-32)
min(-3, 1, 0, 500, 10000)
max(-3, 1, 0, 500, 10000)
```

Extra functions are available in *modules* which need to be imported:

```
import math

math.sqrt(9), math.pi, math.cos(math.pi), math.ceil(1.1)
math.floor(1.8), math.pow(2,3), round(1.2)
```

Strings

A sequence of characters enclosed in single, double or triple quotation marks.

```
s1 = 'Python' %\pause%
s2 = "python"  %\pause%
s3 = '''Python'''  %\pause%
s4 = """Python
     """
```

Strings inside triple quotation marks can span multiple lines.

Strings can be concatenated using the + operator (all terms must be strings):

```
s1 + " is a great " + "language"
```

Casting

A type can be converted to another type if this is feasible, using casting:

```
a = float(5)
b =  int(math.pi)
float("5.76")
int('777') %\pause%

print(a, b)
print(a, b, sep = '::') %\pause%

# specifying the width of the output and
# the number of decimal digits displayed
format(math.pi, '10.3f')
```

Input

User input can be achieved with the `input` function which always returns a string:

```
>>> x = input('Enter your age: ')
Enter your age: 34

>>> type(x)
<class 'str'>
```

Conditionals - The `if` statement

Syntax:

```
if condition1:
    statements
elif condition2:expression
    statements
...
elif conditionN:
    statements
else:
    statements
```

- Unlike other programming languages (which most commonly use curly brackets), a block of statements in Python is created using **consistent** indentation.

Conditionals - The `if` statement (cont'd)

```
age = input('Enter your age: ')
age = int(age)
if age >= 18:
    print('Its time to work\n')
    print('You are old enough')
elif age > 0 and age <=5:
    print("Time to sleep...")
else:
    print('Do whatever you want')
```

Logical operators: `and`, `or`, `not`

The `while` loop

Example:

```
a = 1
b = 10
while a <= b:
    a = a + 1

print(a)
```

The `for` loop

A `for` loop is used to iterate over a sequence (e.g. list, tuple, dictionary, set, string).
It is commonly used with the `range` function which creates a sequence of integers:

```
for x in range(10):
  print(x)
```

```
for x in range(2, 10):
    print(x)
```

The step size can also be specified:

```
for x in range(2, 10, 2):
    print(x)
```

The `for` loop

An optional `else` block can be specified and it will be executed when the loop terminates (it will not be executed if the loop finishes because of a `break`:

```
for i in range(10000000):
    print(i)
else:
    print("At last finished")
```

The `else` block can also be specified as part of the `while` loop.

Accessing Elements in Sequences

```
s = "Robotics" %\pause%
s[0]     # 'R'  %\pause%
len(s)   # 8  %\pause%
s[4]     # t
```

Indices can be negative, which means they will start from the end of the sequence:

```
s[-1]    # 's'
s[-2]    # 'c'
```

Strings are immutable:

```
s[0] = 'r'    # Error!
```

The Colon : Operator

It can be used to select parts of a sequence:

```
s = "Robotics" %\pause%
s[1:5]  %\pause%
s[2:-1] %\pause%
s[2:]   %\pause%
s[4:1:-1]  # step is -1  %\pause%
s[:5]   %\pause%
s[:]
```

Objects, Equality and References

```
s1 = "I, robot"
s2 = "I, robot"
```

Two distinct string objects are created in memory.

```
s1 == s2  # True - compare values  %\pause%
s1 is s2  # False - compare memory addresses
```

```
s3 = s2  # s3 and s2 references point to the same object i
s3 is s2  # True %\pause%
s3 is s1  # False  %\pause%
s3 is not s1  # True
```

Checking if a substring is part of a string:

```
"rob" in s1  # True
```

Lists

A list is a sequence of objects ordered from left to right.

```
m = [2, 1, 5, 10, 7]
k = [61, 'a day in the autumn', 77.23]
```

Operations with lists:

```
>>> m + k
[2, 1, 5, 10, 7, 61, 'a day in the autumn', 77.23]
```

Accessing elements:

```
m[0]
m[2]
m[-3]
```

Lists are mutable:

```
>>> m[2] = 88  %\pause%
>>> m
[2, 1, 88, 10, 7]
```

Lists (cont'd)

Selecting parts of a list:

```
m[2:5]  %\pause%
m[5:2:-1]  %\pause%
m[::-1]   %\pause%
m[3:100]
```

```
>>> m2 = [55, 'abc', [30, 10, 21]]  %\pause%
>>> len(m2)  %\pause%
3
>>> m2[2]   %\pause%
[30, 10, 21]  %\pause%
>>> m2[2][1]  %\pause%
10
```

Deleting elements:

```
>>> m[2:4] = []  %\pause%
>>> m
[2, 1, 7]
```

Functions for Lists

```
m = [2, 1, 5, 10, 7]
sum(m)  # 25 %\pause%
max(m)  # 10  %\pause%

L = sorted(m)  %\pause%
L == m   # False  %\pause%
sorted(m, reverse=True)
```

Methods available for lists:

```
dir(list)  %\pause%
m.sort()  %\pause%
m.insert(2, 6)
```

Tuples

Tuples are similar to lists, however they are immutable:

```
a = (10, 5, 1, 20, 19)  %\pause%
b = 1, 4, 2   %\pause%
type(b)
c = (9)   %\pause%
type(c)  %\pause%
d = (10,) %\pause%
type(d)
```

Operations with tuples create new tuples:

```
(10, 30, 20) + (5, 6, 1)
```

Accessing elements in a tuple:

```
a[0]
a[1:3]
```

Iterating over Sequences

```
mylist = [1, 10, 5, 77, 16]
for i in mylist:
    print(i)  %\pause%

mytuple = (99, 88, 1, 5, 100)
for x in mytuple:
   print(x)
```

List Comprehensions

Creating a list from an iterable object:

```
mylist = [1, 10, 5, 77, 16]  %\pause%
mylist2 = [x + 5 for x in mylist]
```

A condition can also be specified as part of a list comprehension:

```
M = [x**2 for x in range(1,10) if x%2 == 0]
```

Dictionaries

Similar to maps in Java, storing pairs of keys and values:

```
my_contacts = {"James": '0208-3447558', "Jane": '0792-34563
                "George": '0203-9511000'} %\pause%
my_contacts['Jane']  %\pause%
my_contacts['Bob'] = '0207-7776666'
```

Iterating over dictionaries:

```
for k in my_contacts.keys():
    print(my_contacts[k])

for k in my_contacts:
    print(my_contacts[k])

for v in my_contacts.values():
   print(v)
```

Functions

Defining of a function:

```
def my_calculation(x, y):
    result = x**2 + y
    return result
```

Calling a function:

```
my_calculation(5, 6)
```

Returning multiple values:

```
# calculate and return min, max and average
def min_max_avg(data):
   a = min(data)
   b = max(data)
   c = sum(data)/len(data)
   return a, b, c
mi, ma, avg =  min_max_avg([100, 300, 200])
```

Files

Reading from a text file line by line:

```
>>> f = open('myfile.txt')
>>> for li in f:
        print(li)
>>> f.close()
```

Assuming a file `myfile.txt'`:

```
Line 1 1
Line 2 8
Line 3 27
```

Files (cont'd)

Writing to a text file:

```
f2 = open('myfile2.txt', 'w')  # open in 'write' mode %\pa
for i in range(1,10): %\pause%
    f2.write('Line ' + str(i) + ': ' + str(i**2) + '\n') %\
f2.close()
```

File `myfile2.txt` is created:

```
Line 1: 1
Line 2: 4
Line 3: 9
Line 4: 16
Line 5: 25
Line 6: 36
Line 7: 49
Line 8: 64
Line 9: 81
```