

# Python for scientific research

## Flow control

John Joseph Valletta

University of Exeter, Penryn Campus, UK

June 2017



Researcher  
Development



# What we've done so far

- 1 Declare variables using built-in data types and execute operations on them
- 2 **Next:** Controlling the flow of a program

- Executing code one line at a time is useful but limiting
- **Flow control** commands lets us dictate the **order** in which commands are run:

- Executing code one line at a time is useful but limiting
- **Flow control** commands lets us dictate the **order** in which commands are run:
  - ① **If-else**: to change what commands are executed under certain conditions
  - ② **For loops**: to repeat the same thing  $N$  times
  - ③ **While loops**: to repeat the same thing until a specific condition is met

- Executing code one line at a time is useful but limiting
- **Flow control** commands lets us dictate the **order** in which commands are run:
  - ① **If-else**: to change what commands are executed under certain conditions
  - ② **For loops**: to repeat the same thing  $N$  times
  - ③ **While loops**: to repeat the same thing until a specific condition is met

- Executing code one line at a time is useful but limiting
- **Flow control** commands lets us dictate the **order** in which commands are run:
  - ① **If-else**: to change what commands are executed under certain conditions
  - ② **For loops**: to repeat the same thing  $N$  times
  - ③ **While loops**: to repeat the same thing until a specific condition is met

- Executing code one line at a time is useful but limiting
- **Flow control** commands lets us dictate the **order** in which commands are run:
  - ① **If-else**: to change what commands are executed under certain conditions
  - ② **For loops**: to repeat the same thing  $N$  times
  - ③ **While loops**: to repeat the same thing until a specific condition is met

- Print whether the integer  $x$  is positive, negative or zero

```
if x > 0:
    print("x is positive")
elif x < 0:
    print("x is negative")
else:
    print("x is zero")
```

- Note the lack of `{ }` used in other languages; in Python **indentation** is everything!



# For loops

- Print the integers 1 to 5

```
for x in range(5):  
    print(x+1)
```

- Loop through a list of gene names and print them in upper case

```
geneNames = ["Irf1", "Ccl3", "Il12rb1", "Ifng", "Cxcl10"]  
for gene in geneNames:  
    print(gene.upper())
```

# For loops

- Print the integers 1 to 5

```
for x in range(5):  
    print(x+1)
```

- Loop through a list of gene names and print them in upper case

```
geneNames = ["Irf1", "Ccl3", "Il12rb1", "Ifng", "Cxcl10"]  
for gene in geneNames:  
    print(gene.upper())
```

# While loops

- Print the integers 10 to 1

```
x = 10
while x > 0:
    print(x)
    x = x - 1
```

- **Note:**

- 1 Use for loops over while loops where possible
- 2 Ensure that the while condition evaluates to False at some point to avoid an infinite loop

# List Comprehensions

- List comprehensions are an optimized and readable method for creating a list
- Recall the previous example where we looped over gene names and printed them in upper case

```
geneNames = ["Irf1", "Ccl3", "Il12rb1", "Ifng", "Cxcl10"]  
for gene in geneNames:  
    print(gene.upper())
```

- What if I want to store the upper case gene names in another variable, called x for simplicity?

# List Comprehensions

- List comprehensions are an optimized and readable method for creating a list
- Recall the previous example where we looped over gene names and printed them in upper case

```
geneNames = ["Irf1", "Ccl3", "Il12rb1", "Ifng", "Cxcl10"]  
for gene in geneNames:  
    print(gene.upper())
```

- What if I want to store the upper case gene names in another variable, called x for simplicity?

# List Comprehensions

- List comprehensions are an optimized and readable method for creating a list
- Recall the previous example where we looped over gene names and printed them in upper case

```
geneNames = ["Irf1", "Ccl3", "Il12rb1", "Ifng", "Cxcl10"]  
for gene in geneNames:  
    print(gene.upper())
```

- What if I want to store the upper case gene names in another variable, called x for simplicity?

# List Comprehensions

## 1 Using for loops:

```
x = [] # create an empty list to append to
for gene in geneNames:
    x.append(gene.upper())
```

## 2 Using list comprehension:

```
x = [gene.upper() for gene in geneNames]
```

## 3 What if I want to ignore gene Ifng?

```
x = [gene.upper() for gene in geneNames if gene != "Ifng"]
```

# List Comprehensions

## 1 Using for loops:

```
x = [] # create an empty list to append to
for gene in geneNames:
    x.append(gene.upper())
```

## 2 Using list comprehension:

```
x = [gene.upper() for gene in geneNames]
```

## 3 What if I want to ignore gene Ifng?

```
x = [gene.upper() for gene in geneNames if gene != "Ifng"]
```



# List Comprehensions

## 1 Using for loops:

```
x = [] # create an empty list to append to
for gene in geneNames:
    x.append(gene.upper())
```

## 2 Using list comprehension:

```
x = [gene.upper() for gene in geneNames]
```

## 3 What if I want to ignore gene Ifng?

```
x = [gene.upper() for gene in geneNames if gene != "Ifng"]
```

# Enumerate

- When looping over lists, sometimes it's useful to keep track of the index of the iteration
- **Enumerate** is a built-in function that lets us access the iterable element but also its index
- Print the index next to upper cased gene name

# Enumerate

- When looping over lists, sometimes it's useful to keep track of the index of the iteration
- **Enumerate** is a built-in function that lets us access the iterable element but also its index
- Print the index next to upper cased gene name

# Enumerate

- When looping over lists, sometimes it's useful to keep track of the index of the iteration
- **Enumerate** is a built-in function that lets us access the iterable element but also its index
- Print the index next to upper cased gene name
  - ① Using a standard for loop:

```
i = 0 # index counter
for gene in geneNames:
    print("{0}. {1}\n".format(i+1, gene.upper()))
    i = i + 1
```

- ② Using enumerate:

```
for i, gene in enumerate(geneNames):
    print("{0}. {1}\n".format(i+1, gene.upper()))
```

# Enumerate

- When looping over lists, sometimes it's useful to keep track of the index of the iteration
- **Enumerate** is a built-in function that lets us access the iterable element but also its index
- Print the index next to upper cased gene name
  - 1 Using a standard for loop:

```
i = 0 # index counter
for gene in geneNames:
    print("{0}. {1}\n".format(i+1, gene.upper()))
    i = i + 1
```

- 2 Using enumerate:

```
for i, gene in enumerate(geneNames):
    print("{0}. {1}\n".format(i+1, gene.upper()))
```