Python for scientific research

Built-in data types

John Joseph Valletta

University of Exeter, Penryn Campus, UK

June 2017

Declaring variables

- Variables that contain data (numbers, characters etc.) are the fundamental building blocks of any language
- Variables in Python are dynamically typed; called duck typing

  If it walks like a duck and talks like a duck, then it's a duck!
- You can name your variables pretty much anything you want but:

Declaring variables

- Variables that contain data (numbers, characters etc.) are the fundamental building blocks of any language
- Variables in Python are dynamically typed; called duck typing

  If it walks like a duck and talks like a duck, then it's a duck!

- You can name your variables pretty much anything you want but:

Declaring variables

- Variables that contain data (numbers, characters etc.) are the fundamental building blocks of any language
- Variables in Python are dynamically typed; called duck typing

  If it walks like a duck and talks like a duck, then it's a duck!
- You can name your variables pretty much anything you want but:
  - Be **consistent** in your naming convention (see PEP 8)
  - Python is **case sensitive**; `genename` and `geneName` are different variables
  - You cannot use Python **reserved words/keywords**
  - If you're an R user, **DO NOT** use '.' in your variable names i.e `gene.name` is not a valid variable name

Declaring variables

- Variables that contain data (numbers, characters etc.) are the fundamental building blocks of any language
- Variables in Python are dynamically typed; called duck typing

  If it walks like a duck and talks like a duck, then it's a duck!
- You can name your variables pretty much anything you want but:
  - Be **consistent** in your naming convention (see PEP 8)
  - Python is **case sensitive**; `genename` and `geneName` are different variables
  - You cannot use Python **reserved words/keywords**
  - If you're an R user, **DO NOT** use '.' in your variable names i.e `gene.name` is not a valid variable name

Declaring variables

- Variables that contain data (numbers, characters etc.) are the fundamental building blocks of any language
- Variables in Python are dynamically typed; called duck typing

  If it walks like a duck and talks like a duck, then it's a duck!

- You can name your variables pretty much anything you want but:
  - Be **consistent** in your naming convention (see PEP 8)
  - Python is **case sensitive**; `genename` and `geneName` are different variables
  - You cannot use Python **reserved words/keywords**
  - If you're an R user, **DO NOT** use '.' in your variable names i.e `gene.name` is not a valid variable name

Declaring variables

- Variables that contain data (numbers, characters etc.) are the fundamental building blocks of any language
- Variables in Python are dynamically typed; called duck typing

  If it walks like a duck and talks like a duck, then it's a duck!

- You can name your variables pretty much anything you want but:
  - Be **consistent** in your naming convention (see PEP 8)
  - Python is **case sensitive**; `genename` and `geneName` are different variables
  - You cannot use Python **reserved words/keywords**
  - If you're an R user, **DO NOT** use '.' in your variable names i.e `gene.name` is not a valid variable name

Declaring variables

- Variables that contain data (numbers, characters etc.) are the fundamental building blocks of any language
- Variables in Python are dynamically typed; called duck typing

  If it walks like a duck and talks like a duck, then it's a duck!

- You can name your variables pretty much anything you want but:
  - Be **consistent** in your naming convention (see PEP 8)
  - Python is **case sensitive**; `genename` and `geneName` are different variables
  - You cannot use Python **reserved words/keywords**
  - If you're an R user, **DO NOT** use '.' in your variable names i.e `gene.name` is not a valid variable name

Core data types

- **Integers:** `int`

```
NGenes = 1500 # number of genes measured
type(NGenes) # int
```

- **Floating point:** `float`

```
Km = 0.015 # Michaelis constant for chymotrypsin
type(Km) # float
```

- **Complex numbers:** `complex`

```
power = 10 + 2j # 10=active/real power; 2=reactive power
type(power) # complex
```

Core data types

- **Integers:** `int`

```
NGenes = 1500 # number of genes measured
type(NGenes) # int
```

- **Floating point:** `float`

```
Km = 0.015 # Michaelis constant for chymotrypsin
type(Km) # float
```

- **Complex numbers:** `complex`

```
power = 10 + 2j # 10=active/real power; 2=reactive power
type(power) # complex
```

Core data types

- **Integers:** `int`

```
NGenes = 1500 # number of genes measured
type(NGenes) # int
```

- **Floating point:** `float`

```
Km = 0.015 # Michaelis constant for chymotrypsin
type(Km) # float
```

- **Complex numbers:** `complex`

```
power = 10 + 2j # 10=active/real power; 2=reactive power
type(power) # complex
```

Core data types

- **Boolean:** `bool`

```
isTransFactor = True # is protein a transcription factor?
type(isTransFactor) # bool
```

- **Strings:** `str`

```
motif = "AATCAGTT" # DNA sequence motif
type(motif) # str
```

Core data types

- **Boolean:** `bool`

```
isTransFactor = True # is protein a transcription factor?
type(isTransFactor) # bool
```

- **Strings:** `str`

```
motif = "AATCAGTT" # DNA sequence motif
type(motif) # str
```

Container data types

- **Lists:** `list`

```
# Interesting genes
geneNames = ["Irf1", "Ccl3", "Il12rb1", "Ifng", "Cxcl10"]
type(geneNames) # list
```

- **Tuples:** `tuple`

```
# Interesting genes
geneNames = ("Irf1", "Ccl3", "Il12rb1", "Ifng", "Cxcl10")
type(geneNames) # tuple
```

- **Dictionary:** `dict`

```
# My favourite gene
favGene = {"Symbol": "Ifng", "Name": "Interferon", "ID": 3458}
type(favGene) # dict
```

Container data types

- **Lists:** `list`

```
# Interesting genes
geneNames = ["Irf1", "Ccl3", "Il12rb1", "Ifng", "Cxcl10"]
type(geneNames) # list
```

- **Tuples:** `tuple`

```
# Interesting genes
geneNames = ("Irf1", "Ccl3", "Il12rb1", "Ifng", "Cxcl10")
type(geneNames) # tuple
```

- **Dictionary:** `dict`

```
# My favourite gene
favGene = {"Symbol": "Ifng", "Name": "Interferon", "ID": 3458}
type(favGene) # dict
```

Container data types

- **Lists:** `list`

```
# Interesting genes
geneNames = ["Irf1", "Ccl3", "Il12rb1", "Ifng", "Cxcl10"]
type(geneNames) # list
```

- **Tuples:** `tuple`

```
# Interesting genes
geneNames = ("Irf1", "Ccl3", "Il12rb1", "Ifng", "Cxcl10")
type(geneNames) # tuple
```

- **Dictionary:** `dict`

```
# My favourite gene
favGene = {"Symbol": "Ifng", "Name": "Interferon", "ID": 3458}
type(favGene) # dict
```

Container data types

- **Sets:** `set, frozenset`

```
# Sets are mutable
languages = set(["Python", "R", "MATLAB", "C"])
type(languages) # set

# Frozensets are immutable
languages = frozenset(["Python", "R", "MATLAB", "C"])
type(languages) # frozenset
```

- **Range:** `range`

```
# Create immutable sequence of numbers from 0 to 4
x = range(5)
type(x) # range
```

Container data types

- **Sets:** `set, frozenset`

```
# Sets are mutable
languages = set(["Python", "R", "MATLAB", "C"])
type(languages) # set

# Frozensets are immutable
languages = frozenset(["Python", "R", "MATLAB", "C"])
type(languages) # frozenset
```

- **Range:** `range`

```
# Create immutable sequence of numbers from 0 to 4
x = range(5)
type(x) # range
```

Mutable vs immutable objects

- **Mutable** objects (`list, dict, set`) can be changed once assigned

```
# Lists are mutable
geneList = ["Irf1", "Ccl3", "Il12rb1"]
geneList[0]="Irf2" # change first gene to Irf2
```

- **Immutable** objects *cannot* be changed once assigned

```
# Tuples are immutable
geneTuple = ("Irf1", "Ccl3", "Il12rb1")
geneTuple[0]="Irf2"
TypeError: 'tuple' object does not support item assignment
```

- However you can replace an **immutable** object with a new one

```
geneTuple = ("Irf1", "Ccl3", "Il12rb1")
# Replace with a new object
geneTuple = ("Irf2", "Ccl3", "Il12rb1")
```

Mutable vs immutable objects

- **Mutable** objects (`list, dict, set`) can be changed once assigned

```
# Lists are mutable
geneList = ["Irf1", "Ccl3", "Il12rb1"]
geneList[0]="Irf2" # change first gene to Irf2
```

- **Immutable** objects *cannot* be changed once assigned

```
# Tuples are immutable
geneTuple = ("Irf1", "Ccl3", "Il12rb1")
geneTuple[0]="Irf2"
TypeError: 'tuple' object does not support item assignment
```

- However you can replace an **immutable** object with a new one

```
geneTuple = ("Irf1", "Ccl3", "Il12rb1")
# Replace with a new object
geneTuple = ("Irf2", "Ccl3", "Il12rb1")
```

Mutable vs immutable objects

- **Mutable** objects (`list, dict, set`) can be changed once assigned

```
# Lists are mutable
geneList = ["Irf1", "Ccl3", "Il12rb1"]
geneList[0]="Irf2" # change first gene to Irf2
```

- **Immutable** objects *cannot* be changed once assigned

```
# Tuples are immutable
geneTuple = ("Irf1", "Ccl3", "Il12rb1")
geneTuple[0]="Irf2"
TypeError: 'tuple' object does not support item assignment
```

- However you can replace an **immutable** object with a new one

```
geneTuple = ("Irf1", "Ccl3", "Il12rb1")
# Replace with a new object
geneTuple = ("Irf2", "Ccl3", "Il12rb1")
```

Methods of objects

- A Python variable is called an **object**
- Every object has **methods** (functions) associated with it
- These methods are called using the dot notation (‘.’)

```
# DNA sequence motif
motif = "AATCAGTT"

# Use the count method to count occurrence of nucleotide "T"
motif.count("T") # 3

# Use the lower method to convert to lower case
motif.lower() # "aatcagtt"
```