Introduction to Bioinformatics

Variables, Data Types, Data Structures, Control Structures

Janyl Jumadinova
February 3, 2016
Data Type

- Data types are the basic unit of information storage.
- Instances of data types are unique (in Python).
Data Type

- Data types are the basic unit of information storage.
- Instances of data types are unique (in Python).
- Data stored in memory is a string of bits (0 or 1).
Data Type

- Data types are the basic unit of information storage.
- Instances of data types are unique (in Python).

- Data stored in memory is a string of bits (0 or 1).
- How the computer interprets the string of bits depends on the context.
- In Python, we don’t need explicit declaration by specifying the type of the data.
Python Data Type

These are the most basic (and most frequently used) data types:

- Integer: 0, 7, 13, 21, -1
- Floating point: 3.14, 2.718, 1.618
- String: “1”, “2.718”, “True”, “None”, “My name is ”
- Boolean: True, False
- Null: None
Variables

- **Variable** is a name for a memory’s location where a data value is stored.
- **Variable Assignment** assigns a value to the variable
  ```
  count = 0
  ```
- **Expression** is a combination of one or more operators (+, −, %, …) and operands (literals, constants, variables,...)
We can get data directly from the user and create an interactive program.

```python
person = raw_input('Enter your name: ')  
print 'Hello', person 

# age is a variable of type string  
age = raw_input('What is your age?') 
age = int(age)  # age is a variable of type int now 
age = age + 2 
print "In two years you will be ", age, " years old." 
```
We can get data directly from the user and create an *interactive program*.

```python
person = raw_input('Enter your name: ')  
print 'Hello', person

# age is a variable of type string  
age = raw_input('What is your age?')  
age = int(age)  # age is a variable of type int now  
age = age + 2  
print "In two years you will be ", age, " years old."  
print "In four years you will be ", age+2, " years old."  
```
raw_input() always returns a string;

input() evaluates the return, so it may return a number, for example - run the following example

```python
input1 = input('Enter a number: ')
print(type(input1), input1)
input2 = raw_input('Enter a number: ')
print(type(input2), input2)
```
Strings

- Strings in Python are created with paired single or double quotes.
- Multi-line strings can be created by enclosing them with three single or double quotes on each end (e.g. """This could span several lines"""").

The + and * operators work for strings, so "help" + " me" produces the string "help me", and "help" * 3 produces "helphelphelp".
Strings

- Strings in Python are created with paired single or double quotes.
- Multi-line strings can be created by enclosing them with three single or double quotes on each end (e.g. """This could span several lines"""").
- The + and * operators work for strings, so “help” + “ me” produces the string “help me”, and “help” * 3 produces “helphelphelp”.
Strings

- Strings in Python are created with paired single or double quotes.
- Multi-line strings can be created by enclosing them with three single or double quotes on each end (e.g. """This could span several lines"""").
- The + and * operators work for strings, so “help” + “ me” produces the string “help me”, and “help” * 3 produces “helphelphelp”.
- String indexing:
  
  ```python
  word = "apple"
  word[4] will give a letter e
  word[0:2] will give ap
  ```
Python Data Structures

- **List**: an ordered, zero-indexed collection of objects. For example: [1, “A”, 3.0]
- **Set**: an unordered collection of elements, guarantees each element is unique. For example: {1, 5, 3.0}
- **Dictionary**: an unordered collection of key/value pairs. Each key is unique. For example: {1:“One”, “A”:5, 3.0:“Three”}
Python Ordered Types

- **Strings** ‘this’ or “this”... - **immutable**
  - S=‘act’; s.upper(); s.lower()
  - S.replace(old, new,[count])
  - S.count(sub,[start[,end]])
  - Len(s)
  - Find
  - Index
  - Split
  - join

- **Tuples**
  - Pt=(2,4,1)
  - Pt[0]
  - immutable

- **Lists** - are **mutable**
  - L1=[1,2,3]; L1[0]
  - L2 = [‘aat’, ‘tft’, 5]
  - Set: A={1,2,3}; [2*x for x in A]
  - List(string)
  - L2=copy.copy(l1)
  - L.append(33)
  - L.insert(pos,elem)
  - L.Extend(l2) same as l1+l2
  - L1.pop(2)
  - L.remove(elem) or del list[2]
  - L.count(x)
  - L.index(x)
  - L.reverse()
  - L.sort()
Python Unordered Types

- **Dictionaries** – stores indexed data
  - 1) \{key:value, key:value\}
  - code = \{'A':'Ala', 'C':'Cys'}
  - code['C']
  - 2) code = dict(a='Ala', c='Cys')
  - Dict(d)='dist'
  - for elem in dict:
    - Print elem
  - Code.keys(); code.values()
  - Code.items()
  - Code.get('A', 'Not in dict')
  - Del code['A']

- **Sets** - like a list BUT with unique elements & no order
  - S1=set ([1,2,'pink',5])
  - S1.add('red')
  - S1.intersection(s2)
  - S1.union(s2)
  - S1-s2
  - S1.symmetric_difference(s2)
  - Min, max, len, in
  - List(set1)
  - Frozenset(list) -- immutable
Python basic language structure

- Newline terminates a command - no semicolon required.
- Indentation alone designates nested code blocks - no curly braces required.
- Functions, nested loops and conditionally evaluated code are all indicated using indentation.
- `#` denotes the start of a single line comment.
Basic numerical operations

- The +, -, *, /, % (modulo) and ** (power) all behave as expected.
- The = assigns the value on the right to the variable on the left.
- The +=, -=, *=, /= and **= perform the indicated operation between the variable on the left and the value on the right, then assign the result to the variable on the left.
Basic condition tests

- `==` tests to see if two things have the same value.
- `!=` tests to see if two things have a different value.
- The `<`, `>`, `<=`, `>=` all compare relative values.
- `is` tests to see if two things have the same identity.
- `in` tests element membership in a collection.
Any value, other than `None`, `False`, `0`, `'''`, or an empty collection evaluates to `True` in a boolean context.

The boolean operators supported by python, in order of increasing precedence, are:

- `and`
- `or`
- `not`
Algorithms

Any problem can be solved by an algorithm.

- **Algorithm** is a procedure for solving a problem in terms of the *actions* to be executed and the *order* in which those actions are to be executed.
Algorithms

Any problem can be solved by an algorithm.

- **Algorithm** is a procedure for solving a problem in terms of the *actions* to be executed and the *order* in which those actions are to be executed.

- Both the actions and their order are important.
Control Structures

Three Groups of Control Structures

1. Sequential Structure
   - It is just built into the language itself.
Control Structures

1. Sequential Structure

2. Selection Structures
   - `if` : single selection
   - `if/elif/else` : double or multiple selection
Control Structures

1. Sequential Structure
2. Selection Structures
3. Repetition Structure
   while
   for

Loop (Iteration)
Control Structures

- Python programs are built from these control structures: \texttt{if}, \texttt{while}, \texttt{for}
- You implement computer algorithms by stringing sequences of these control structures together.