Theory of Algorithms

ian.mcloughlin@gmit.ie

Topics

Python

Timing Algorithms

Functional Programming

Turing Machines

Complexity Classes

Python

About Python

January 1994 – Python 1.0.0 released.

Guido van Rossum – Designer/Author of Python.

Current versions – 3.5.1 and 2.7.11.

Interpreted – Python implementation must be present at runtime.

Off-side rule – Blocks identified by indentation, as opposed to curly braces.

Popularity – IEEE Spectrum ranks it as the fourth most popular language (July 2015).

Community – Python Enhancement Proposals, notably PEP 8: The Python Style Guide.

spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages

Guido van Rossum

- Started Python as a hobby.
- Worked for Google, half-time spent on Python.
- Now works at Dropbox.
- Benevolent dictator for life (BDFL).

Conditions

```
1  x = int(raw_input("Please enter an integer: "))
2  if x < 0:
3    x = 0
4    print 'Negative changed to zero'
5  elif x == 0:
6    print 'Zero'
7  elif x == 1:
8    print 'Single'
9  else:
10   print 'More'
```

docs.python.org/2/tutorial

Loops

```
# A for loop.
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
  print(i, a[i])
```

```
# A while loop.
a, b = 0, 1
while b < 1000:
  print(b)
  a, b = b, a+b
```

docs.python.org/3/tutorial

Functions

```
1  # write Fibonacci series up to n
2  def fib(n):
3      """Print a Fibonacci series up to n."""
4      a, b = 0, 1
5      while a < n:
6          print(a)
7          a, b = b, a+b
```

docs.python.org/3/tutorial

CPython

Reference implementation – Many different Python implementations exist.

Version 3 – Broke backwards compatibility (somewhat).

Unladen Swallow – Google attempt to fix some Python problems.

Modules – Lots of great Python modules available.

www.python.org

Lists

Lists in Python are usually written as comma-separated values between square brackets.

Types – elements of a list don't have to have the same types.

Slicing is possible, where we take a sublist of the list.

Assignment to slices is possible.

len() is a built-in function that returns the length of a list.

range() is a built-in function that returns a list of numbers. Note: it returns an *iterator*.

```
1 letters = ['a', 'b', 'c']
2 letters[1:] = ['c', 'd']
3 range(10) # [0,1,2,3,4,5,6,7,8,9]
```

Strings

Strings are a lot like lists in Python.

Assignment to slices is not allowed, however.

```
1  words = "This is a sentence."
2  words[8]          # a
3  words[5:7]        # is
4  words[:7]         # This is
5  words[10:]        # sentence.
6  words[17:9:-1]    # ecnetnes
7
8  len(words)        # 19
9  "One" + "Two"     # OneTwo
```

docs.python.org/3/tutorial

Functions

def is the keyword for defining a function.

Parameters can be given defaults, so that they are optional.

```
1  def axn(x, a=1, n=2):
2          return a*(x**n)     # ax^n
3
4  axn(3)          # 9
5  axn(3, 2)       # 18
6  axn(3, 2, 3)    # 54
7  axn(3, n=3)     # 27
```

docs.python.org/3/tutorial

List comprehensions

Comprehensions are quick ways of creating lists from other lists.

```
1 nos = range(5) # [0, 1, 2, 3, 4]
2 squares = [i*i for i in nos] # [0, 1, 4, 9, 16]
3 oddsqs = [i*i for i in nos if i % 2 == 1] # [1, 9]
```

docs.python.org/3/tutorial

map()

map() takes a function and a list.

New list – it returns a new generator, which is the original list with the function applied to each element.

```
1 map(len, words)
2 list(map(len, words))
```

docs.python.org/3/tutorial

Lambda functions

lambda functions are short, inline functions.

Nameless – lambda functions need not have a name.

```
1  lambda x: x + n
```

docs.python.org/3/tutorial

Without generators

```
1  # Build and return a list
2  def firstn(n):
3      num, nums = 0, []
4      while num < n:
5          nums.append(num)
6          num += 1
7      return nums
8
9  sum_of_first_n = sum(firstn(1000000))
```

https://wiki.python.org/moin/Generators

With generators

```
# yields items instead of returning a list
def firstn(n):
    num = 0
    while num < n:
        yield num
        num += 1

sum_of_first_n = sum(firstn(1000000))
```

https://wiki.python.org/moin/Generators

Permutations

Permutations are rearrangements of ordered collections of items.

Example: "abcd" is a rearrangement of "bacd".

Abstraction

We can consider permutations abstractly. For instance, if we have four items to rearrange, we can label the first item 1, the second 2, and so on. Then we can represent the various permutations, in terms of the numbers associated with the items. The permutations above could be represented by (1 2 3 4) and (2 1 3 4). We can see then that when we are considering permutations of n items, it doesn't matter what those items are, we just consider them as numbers.

Counting permutations

With four items, how many distinct permutations are there?

Consider having four placeholders where we can place the items:

?	?	?	?

When we place an item in the first box, we have four choices, then we are left with only three choice for the second, two choices for the third, and one choice (i.e. not a choice at all) for the last.
So for there are $4 \times 3 \times 2 \times 1 = 4!$ choices in total.

Counting anagrams

Repitition

What happens when we consider two of our items to be the same? For instance, what if we are looking for all distinct rearrangements of "aacd" as opposed to "abcd"? In that case we need to account for the rearrangements of those items by dividing by the factorial of the number of times each item is repeated: $\frac{4!}{2!}$.

If more than one item is repeated a number of times we just keep dividing by the factorials of the numbers of repititons. The distinct number of rearrangements of "aaabbcd" is $\frac{7!}{3!2!}$.

Exercise

Calculate the number of distinct rearragements of the word "Mississippi".

Timing Algorithms

Functional Programming

Turing Machines

Complexity Classes