

CS1010S

Tutorial 1: Introduction to Python

Nicholas Russell Saerang (russellsaerang@u.nus.edu)

Table of contents



1

Admin

Introduction, tutorial,
topics, expectations

2

Tools

Python, helpful tools

3

Lecture Recap

Recall what you
have learn

4

Tutorial 1

Introduction to
Python

Admin

Introduction, tutorial,
topics, expectations



Introduction

Nicholas **Russell** Saerang

Year 4 DSA, Minor in CS

ex-Head TA of CS1010S

Took CS1010S in 20/21 Sem 1

Email: russellsaerang@u.nus.edu



**Your Turn! Introduce your
Name, Major and Year**

Administrative Details



"**Tutorial** & Recitation" for 5% Grades

- 1 Tutorial Slot
- 1 Tutors to 14 students ratio

T06 - Mondays 1300-1400 @ BIZ2-0201

Use telegram group chat to help one another and chit chat!

Ask question on Coursemology Forum /
Ask me after tutorial class

Expectations



- 1) Prep and attempt your tutorial before class
- 2) Do not plagiarise
- 3) Participate in class (Raise question, answer question, share idea, ...)
- 4) Resolve issues as much as you can before asking me

Tools

Python, helpful tools

Python

- 1) IDLE - the default editor
- 2) Shell - the interpreter
- 3) Coursemology - Missions, trainings etc.
- 4) Documentation - Contains all information about the language
<https://docs.python.org/3/library/index.html>
- 5) Python coding style guide: PEP8
<https://www.python.org/dev/peps/pep-0008/>

Where to find help?



1. Coursemology forums
<https://coursemology.org/courses/2692/forums>
2. Python tutorial
<https://docs.python.org/3/tutorial/index.html>
3. Python Tutor
<https://pythontutor.com/>
4. StackOverflow
<https://stackoverflow.com/>
5. Telegram (mainly admin)
Groups & Me

Tutorial 1

Lecture Recap



Primitive Data Type

Integer (int)

Float (float)

String (str)

Boolean (bool)

NoneType (None)

```
>>> 1 / 3 # 0.3333333333333333
```

****By default, Python can handle up to 16dp**

```
>>> "a" + "  b" # "a  b"
```

****Spacing matters in Python string!**

```
T rue, False # Note the capital letter
```

Type conversion

Integer (***int***)

Float (***float***)

String (***str***)

Boolean (***bool***)

```
my_int = 1  
my_string = str(my_int) # "1"
```

Using `type(...)` to check the data type

```
type(my_int) # <class 'int'>
```

```
type(my_string) # <class 'str'>
```

Indexing

my_string[index]

```
word = "abcdefg"
```

```
p = word[0] # "a"
```

```
x = word[-1] # "g"
```

word	a	b	c	d	e	f	g
+ve	0	1	2	3	4	5	6
-ve	-7	-6	-5	-4	-3	-2	-1

Arithmetic Operators

`+ - * / ** // %` *# Assumed knowledge*

Relational Operators

`> < >= <= == !=` *# Assumed knowledge*

Logical Operators

and not or *# Assumed knowledge*

If you are not confident with this, approach me after class

String operation

1) Concatenation

string = "A" + "B" # "AB"

2) String comparison

"C" *in* *string* # False

"C" *>=* *string* # True

"B" *not in* *string* # False

= vs == (Don't be confused)

`x = 1` # Variable assignment

`x == 2` # Logical comparison



Box-and-pointer diagram
(You will learn more in the future)

`a = 1`

In English: Variable `a` is assigned an integer value `1`

`a = 10`

In English: Variable `a` is reassigned to another integer value `10`

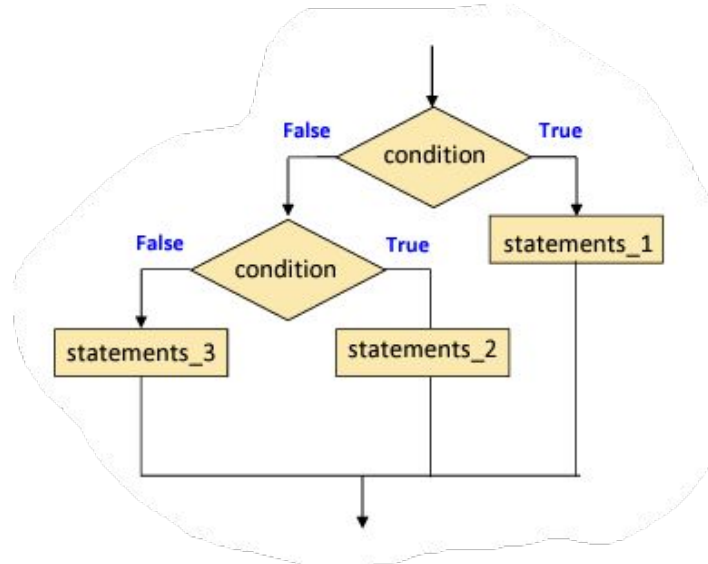
Conditional Statement

if <cond_1>:
 <statements_1>

elif <cond_2>:
 <statements_2>

else:
 <statements_n>

Draw this control flow



Function

def <name>(<parameters>):

 <body>

return <output>

Define function *add_one* which take one argument, *x*
return $x + 1$ when ***x* is positive**, else return *x*

```
def add_one(x):  
    if x > 0:  
        return x + 1  
    else:  
        return x
```

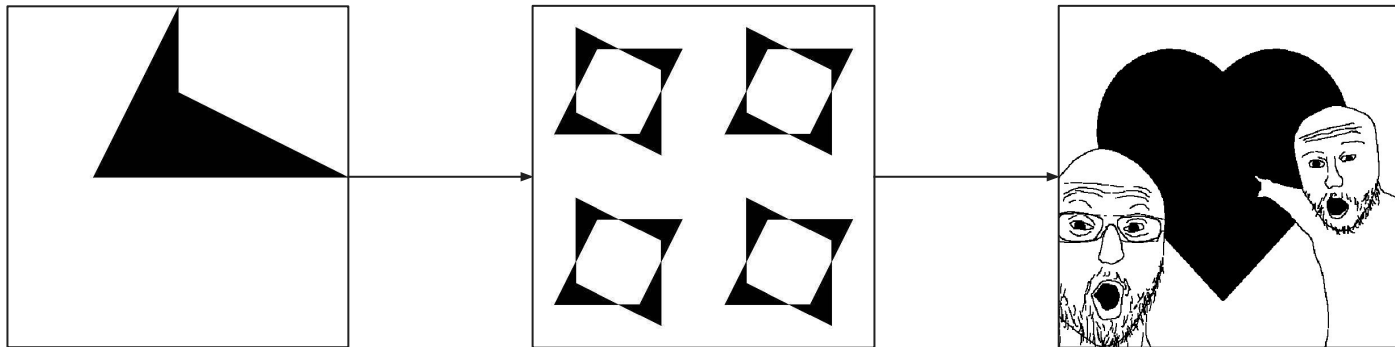
Runes

What are those!?

What are runes?

Something you just learnt in lecture this week!

- Abstract Environment! "Picture Language"



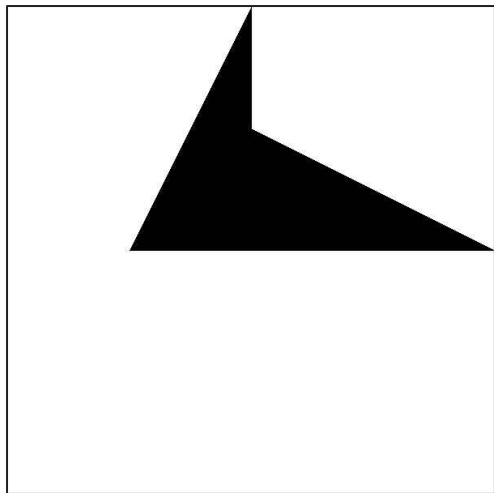
Basics:

- Don't need to know how the rune functions work
- Just need to know how to use them as they are described
- Build a complex image!

Rune Primitives VS Python

Runes

```
>>> show(nova_bb) # visually  
shows but does not return  
nova_bb
```



Python

```
>>> print("I am a CS1010S  
student")  
I am a CS1010S student
```

Rune Methods



Also defined in `runes.py` are the following functions as discussed in class:

- `stack`
- `stack_frac`
- `quarter_turn_right`
- `eighth_turn_left`
- `flip_horiz`
- `flip_vert`
- `turn_upside_down`
- `quarter_turn_left`
- `beside`
- `make_cross`
- `repeat_pattern`
- `stackn`

Tutorial 1

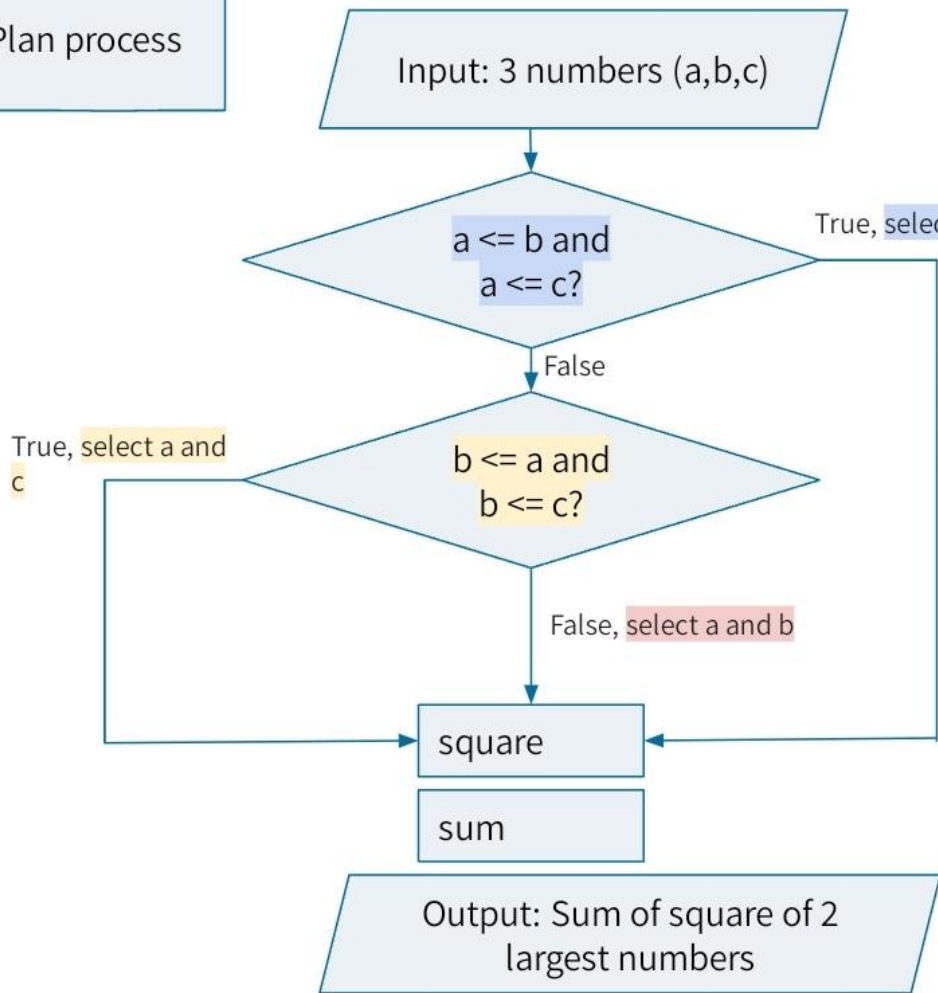
Introduction to Python



Question 1: Σ pair squares

Implement a function that takes three numbers as arguments and returns the sum of the squares of the two largest numbers. It is not guaranteed that all numbers are distinct. For example, if the numbers are 1, 3, 1, the answer is $3^2 + 1^2 = 10$. If the numbers are 7, 7, 2, the answer is $7^2 + 7^2 = 98$.

Step 2: Plan process



Step 3: Write code

```
def bigger_sum(a, b, c):  
    if a<=b and a<=c:  
        return b**2 + c**2  
    elif b<=a and b<=c:  
        return a**2 + c**2  
    else:  
        return a**2 + b**2
```

Question 1: Σ pair squares

Implement a function that takes three numbers as arguments and returns the sum of the squares of the two largest numbers. It is not guaranteed that all numbers are distinct. For example, if the numbers are 1, 3, 1, the answer is $3^2 + 1^2 = 10$. If the numbers are 7, 7, 2, the answer is $7^2 + 7^2 = 98$.

```
def sum_squares(x, y):
```

```
    return x**2 + y**2
```

```
def bigger_sum(a, b, c):
```

Question 1: Σ pair squares

```
def sum_squares(x, y):  
    return x**2 + y**2  
  
def bigger_sum(a, b, c):  
    if a <= b and a <= c:  
        return sum_squares(b, c)  
  
    elif b <= a and b <= c:  
        return sum_squares(a, c)  
  
    else:  
        return sum_squares(a, b)
```

Question 1: Σ pair squares

```
def bigger_sum(a, b, c):  
    return a**2 + b**2 + c**2 - min(a,b,c)**2
```

min is a built-in function, you can use it to get the minimum of at least two values given as the function parameters.

Question 2: Magnitude

Implement a function `magnitude` that takes in the coordinates of two points on a plane, $(x1, y1)$ and $(x2, y2)$, as arguments and returns the magnitude of the vector between them.

```
def magnitude (x1 , y1 , x2 , y2 ) :  
    # returns the magnitude of the vector  
    # between the points (x1, y1) and (x2, y2)
```

```
>>> magnitude (2, 2, 5, 6)  
5.0
```

Question 2: Magnitude

```
from math import sqrt

def magnitude(x1, y1, x2, y2):
    return sqrt((y2-y1)**2 + (x2-x1)**2)

from math import *

def magnitude(x1, y1, x2, y2):
    return sqrt((y2-y1)**2 + (x2-x1)**2)
```

Question 2: Magnitude

```
import math
```

```
def magnitude(x1, y1, x2, y2):  
    return math.sqrt((y2-y1)**2 + (x2-x1)**2)
```

```
import math as m
```

```
def magnitude(x1, y1, x2, y2):  
    return m.sqrt((y2-y1)**2 + (x2-x1)**2)
```


Question 2: Magnitude

```
def magnitude(x1, y1, x2, y2):  
    return ((y2-y1)**2 + (x2-x1)**2)**0.5
```

Question 3: Leap years

Implement a function `is_leap_year` that takes one integer parameter and decides whether it corresponds to a leap year, i.e. the function `is_leap_year` returns `True` if the input parameter is true, and `False` otherwise. So which years are leap years? Well, according to Wikipedia:

In the Gregorian calendar, the current standard calendar in most of the world, most years that are integer multiples of 4 are leap years. In each leap year, the month of February has 29 days instead of 28. Adding an extra day to the calendar every four years compensates for the fact that a period of 365 days is shorter than a solar year by almost 6 hours. This calendar was first used in 1582.

Some exceptions to this rule are required since the duration of a solar year is slightly less than 365.25 days. Over a period of four centuries, the accumulated error of adding a leap day every four years amounts to about three extra days. The Gregorian Calendar therefore omits 3 leap days every 400 years, omitting February 29 in the 3 century years (integer multiples of 100) that are not also integer multiples of 400. For example, 1600 was a leap year, but 1700, 1800 and 1900 were not. Similarly, 2000 was a leap year, but 2100, 2200, and 2300 will not be. By this rule, the average number of days per year is $365 + 1/4 - 1/100 + 1/400 = 365.2425$.

Question 3: Leap years

```
def is_leap_year(year):  
    # leap years are integer multiples of 4  
    # except years that are multiples of 100  
    # but not years that are multiples of 400  
    pass
```

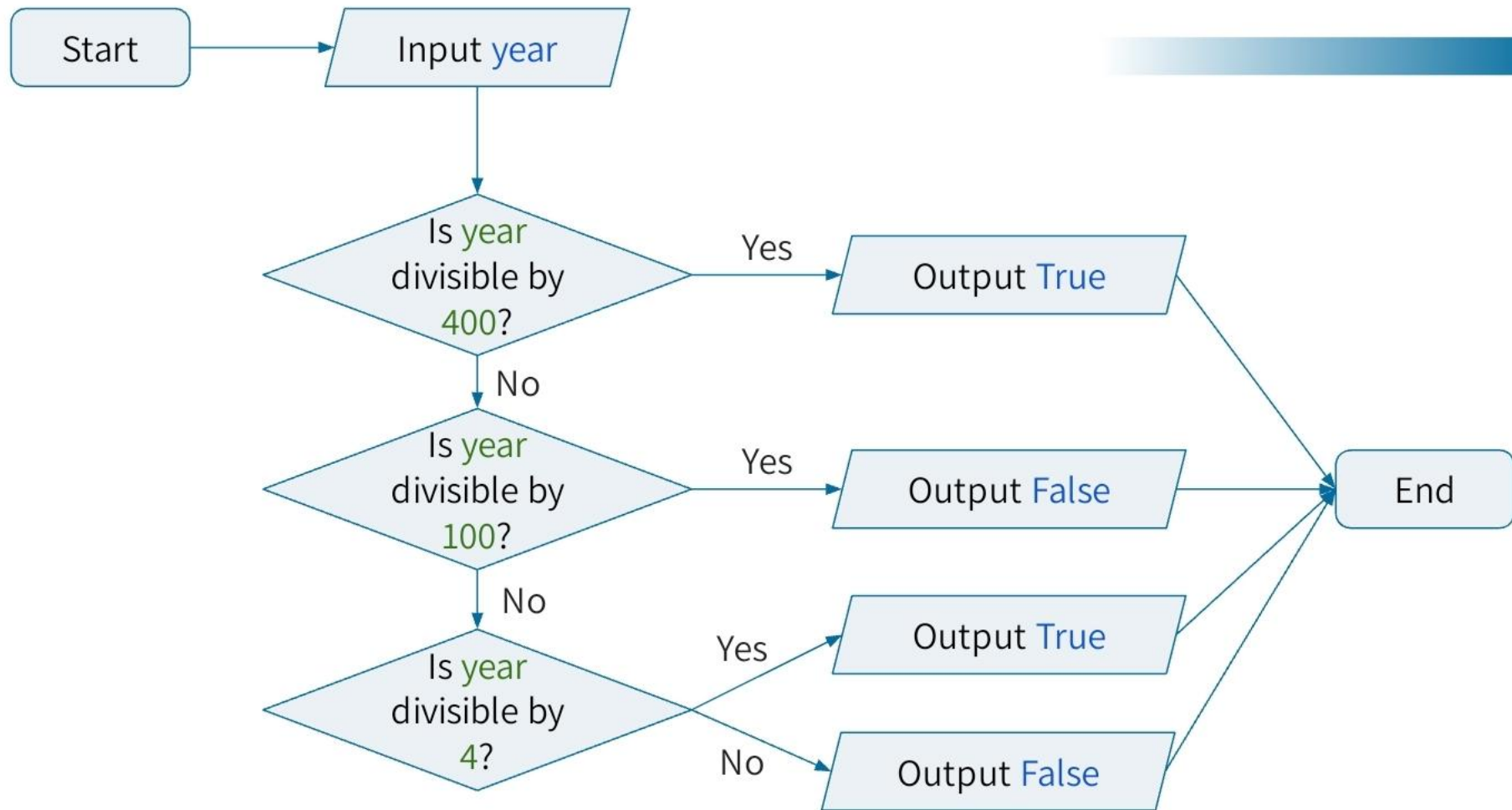
Question 3: Leap years - incorrect

```
def is_leap_year(year):  
    # leap years are integer multiples of 4  
    if year % 4 == 0: return True  
  
    # except years that are multiples of 100  
  
    if year % 100 == 0: return False  
  
    # but not years that are multiples of 400  
  
    if year % 400 == 0: return True  
  
    return False
```

Question 3: Leap years - incorrect

```
def is_leap_year(year):  
    # leap years are integer multiples of 4  
    if year % 4 == 0: return True  
  
    # except years that are multiples of 100  
  
    if year % 100 == 0: return False  
  
    # but not years that are multiples of 400  
  
    if year % 400 == 0: return True  
  
    return False
```

is_leap_year(1900) # True	is_leap_year(1996) # True
is_leap_year(1999) # False	is_leap_year(2000) # True



Question 3: Leap years

```
def is_leap_year(year):  
    # but not years that are multiples of 400  
    if year % 400 == 0: return True  
  
    # except years that are multiples of 100  
    if year % 100 == 0: return False  
  
    # leap years are integer multiples of 4  
    if year % 4 == 0: return True  
  
    return False
```

Question 3: Leap years

```
def is_leap_year(year):  
    # but not years that are multiples of 400  
    if year % 400 == 0: return True  
  
    # except years that are multiples of 100  
    if year % 100 == 0: return False  
  
    # leap years are integer multiples of 4  
    if year % 4 == 0: return True  
  
    return False
```

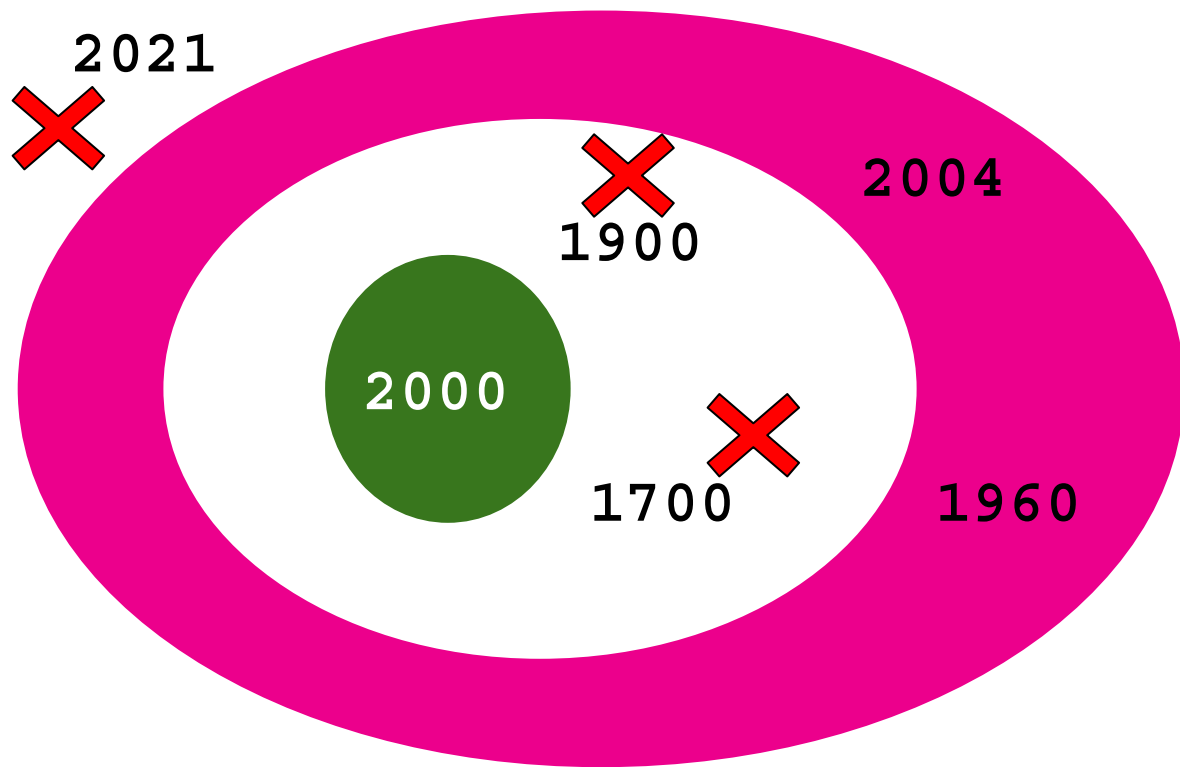
is_leap_year(1900) # False	is_leap_year(1996) # True
is_leap_year(1999) # False	is_leap_year(2000) # True

Question 3: Leap years

```
# short form, not recommended
def is_leap_year(year):
    return (year % 4 == 0 and not year % 100 == 0) \
        or (year % 400 == 0)
```

is_leap_year(1900) # False	is_leap_year(1996) # True
is_leap_year(1999) # False	is_leap_year(2000) # True

Question 3: Leap years - Venn diagram



Green:

`year % 400 == 0`

Red:

`year % 4 == 0 and`

`year % 100 != 0`

Combined:

`year % 400 == 0 or`

`(year % 4 == 0 and`


`year % 100 != 0)`

Extra Questions

```
x = 1
```

```
def foo(x):  
    return x + 1
```

```
print(foo(2))  
print(foo(x))  
print(x)
```



Extra Questions

```
x = 1
```

```
def add_one(x):  
    print (x + 1) # return None
```

```
print(add_one(2))
```

```
y = add_one(x)
```

```
print(add_one(y))
```

