Leiden University
Medical Center

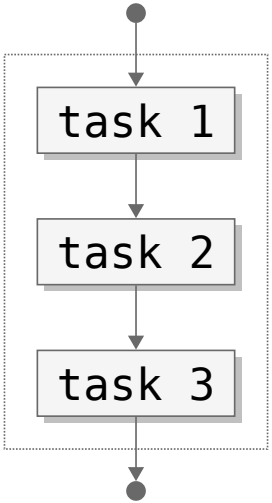
Python Programming

Flow Control

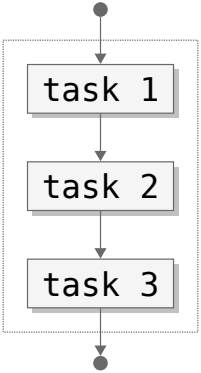
Mihai Lefter

Outline

Sequential Execution



Sequential Execution



sum.py

```
1  a = 100
2  b = 200
3  print(a+b)
```

terminal

```
$ python sum.py
300
```

Intermezzo - User input

Performed with the `input([prompt])` built-in function:

- If the `prompt` argument is present, it is written to the standard output.
- The user input is then read as a string.

Intermezzo - User input

Performed with the `input([prompt])` built-in function:

- If the `prompt` argument is present, it is written to the standard output.
- The user input is then read as a string.

sum.py

```
1 a = input('a = ')
2 b = input('b = ')
3 print(a+b)
```

Intermezzo - User input

Performed with the `input([prompt])` built-in function:

- If the `prompt` argument is present, it is written to the standard output.
- The user input is then read as a string.

sum.py

```
1 a = input('a = ')
2 b = input('b = ')
3 print(a+b)
```

terminal

```
$ python sum.py
a = 100
b = 200
100200
```

Intermezzo - User input

Performed with the `input([prompt])` built-in function:

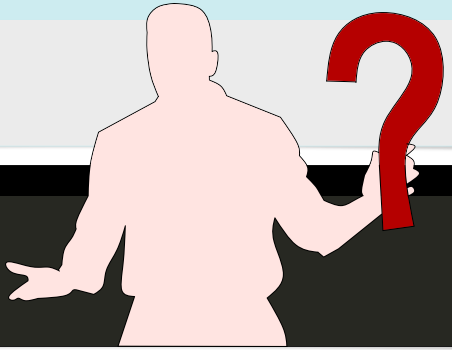
- If the `prompt` argument is present, it is written to the standard output.
- The user input is then read as a string.

sum.py

```
1 a = input('a = ')
2 b = input('b = ')
3 print(a+b)
```

terminal

```
$ python sum.py
a = 100
b = 200
100200
```



Intermezzo - User input

Performed with the `input([prompt])` built-in function:

- If the `prompt` argument is present, it is written to the standard output.
- The user input is then read as a `string`.

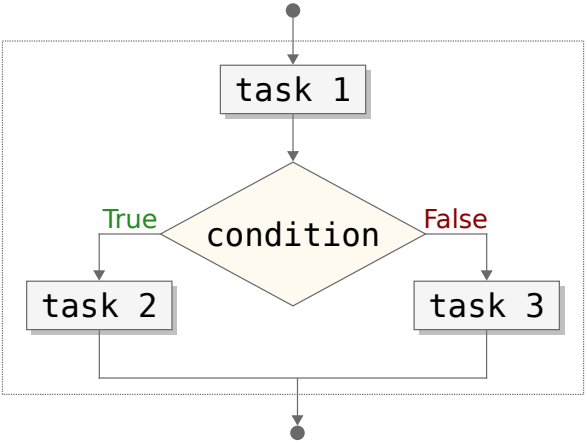
sum.py

```
1 a = int(input('a = '))
2 b = int(input('b = '))
3 print(a+b)
```

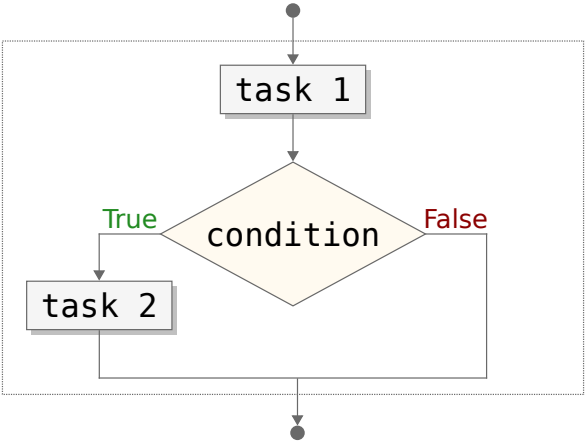
terminal

```
$ python sum.py
a = 100
b = 200
300
```

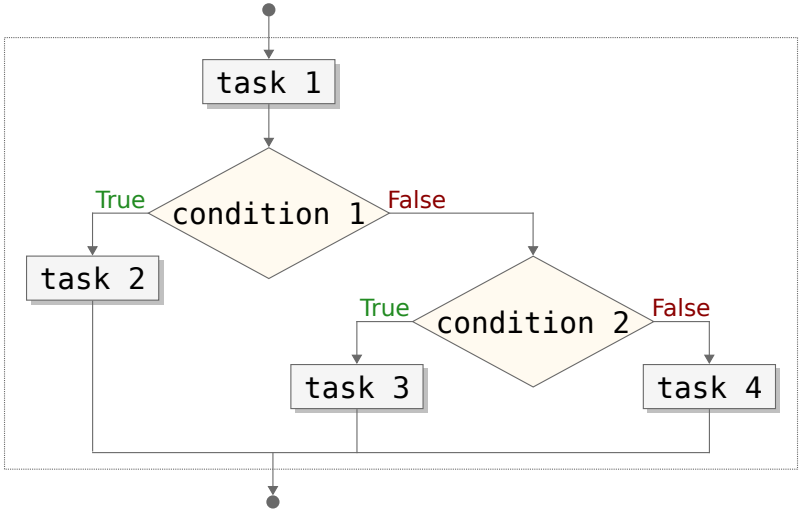
Conditionals



Conditionals



Conditionals



Truth Value Testing

- Built-in objects considered false:
  - constants defined to be false: `None` and `False`.

Truth Value Testing

- Built-in objects considered false:
  - constants defined to be false: `None` and `False`.
  - zero of any numeric type: `0`, `0.0`, `0j`, `Decimal(0)`, `Fraction(0, 1)`.

Truth Value Testing

- Built-in objects considered false:
  - constants defined to be false: `None` and `False`.
  - zero of any numeric type: `0`, `0.0`, `0j`, `Decimal(0)`, `Fraction(0, 1)`.
  - empty sequences and collections: `''`, `()`, `[]`, `{}`, `set()`, `range(0)`.

Truth Value Testing

- Built-in objects considered false:
  - constants defined to be false: `None` and `False`.
  - zero of any numeric type: `0`, `0.0`, `0j`, `Decimal(0)`, `Fraction(0, 1)`.
  - empty sequences and collections: `''`, `()`, `[]`, `{}`, `set()`, `range(0)`.
- For the moment, let's assume that any other object is considered true.

Truth Value Testing

- Built-in objects considered false:
  - constants defined to be false: `None` and `False`.
  - zero of any numeric type: `0`, `0.0`, `0j`, `Decimal(0)`, `Fraction(0, 1)`.
  - empty sequences and collections: `''`, `()`, `[]`, `{}`, `set()`, `range(0)`.
- For the moment, let's assume that any other object is considered true.

IPython

```
In [13]: bool(0)
Out[13]: False

In [14]: bool(1)
Out[14]: True

In [15]: bool([False])
Out[15]: True
```

Comparisons

Operation	Meaning	Example
`<`	strictly less than	`x < y`
`<=`	less than or equal	`x <= y`
`>`	strictly greater than	`x > y`
`>=`	greater than or equal	`x >= y`
`==`	equal	`x == y`
`!=`	not equal	`x != y`
`is`	object identity	`x is y`
`is not`	negated object identity	`x is not y`

Comparisons

IPython

```
In [16]: 3 < 4
Out[16]: True

In [17]: 3 <= 3.5
Out[17]: True

In [18]: 3 == 3.0
Out[18]: True

In [19]: 3 is 3.0
Out[19]: False

In [20]: 3 is 3
Out[20]: True
```

Boolean (Logical) Operations

Operation	Result	Notes
`x or y`	if `x` is false, then `y`, else `x`	(1)
`x and y`	if `x` is false, then `x`, else `y`	(2)
`not x`	if `x` is false, then `True`, else `False`	(3)

1. It evaluates `y` only if `x` is false.
2. It evaluates `y` only if `x` is true.
3. `not x == y` is interpreted as `not (x == y)` and `x == not y` is a syntax error.

Boolean (Logical) Operations

x	y	x or y	x and y
True	True	True	True
True	False	True	False
False	True	True	False
False	False	False	False

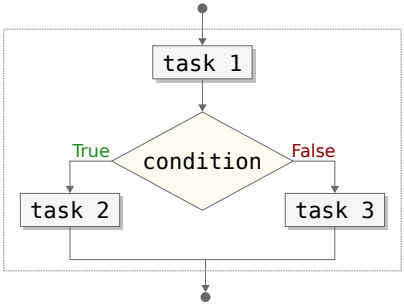
Boolean (Logical) Operations

IPython

```
In [21]: 3 < 4 and 5 <= 10
Out[21]: True

In [22]: 3 < 4 or 5 <= 10
Out[22]: True

In [23]: 3 < 4 and 5 > 10
Out[23]: False

In [24]: 3 < 4 and 5 > 10
Out[24]: True
```

Conditionals

`if` statement



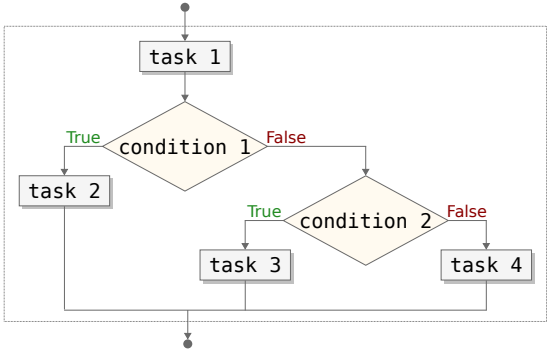
max.py

```
1  a = int(input('a = '))
2  b = int(input('b = '))
3
4  if a > b:
5      print(a)
6  else:
7      print(b)
```

terminal

```
$ python max.py
a = 100
b = 200
200
```

`if` statement



compare.py

```
1  a = int(input('a = '))
2  b = int(input('b = '))
3
4  if a > b:
5      print(a)
6  elif a == b:
7      print('equal')
8  else:
9      print(b)
```

terminal

```
$ python compare.py
a = 100
b = 100
equal
```

Indentation

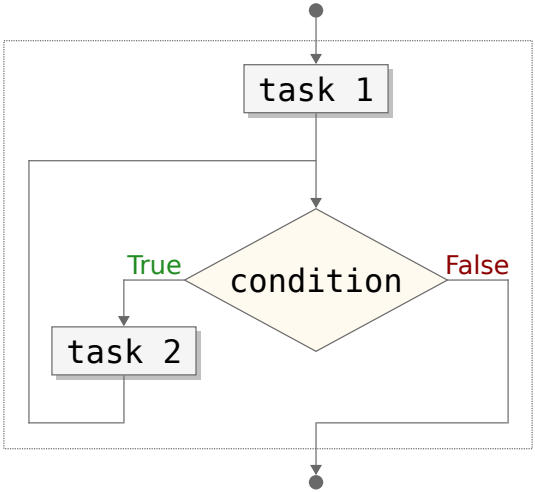
Python uses indentation to delimit blocks

- Instead of `begin ... end` or `{ ... }` in other languages.
- Always increase indentation by 4 spaces, never use tabs.
  - In any case, be consistent.

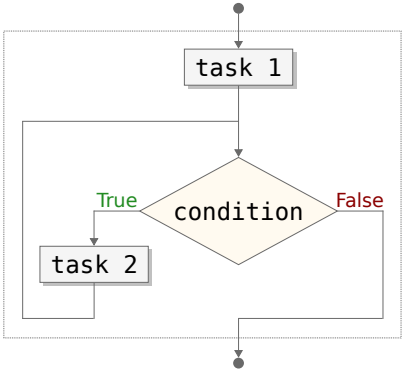
indentation_example.py

```
1 if False:
2     if False:
3         print('Why am I here?')
4     else:
5         while True:
6             print('When will it stop?')
7     print("And we're back to the first indentation level")
```

Loops



`while` statement



while_example.py

```
1 i = 0
2 while i < 5:
3     print(i)
4     i += 1
```

terminal

```
$ python while_example.py
0
1
2
3
4
```

`for` statement

Used to iterate over a sequence.

for_example.py

```
1 colors = ['red', 'white', 'blue', 'orange']
2
3 for color in colors:
4     print(color)
```

`for` statement

Used to iterate over a sequence.

for_example.py

```
1  colors = ['red', 'white', 'blue', 'orange']
2
3  for color in colors:
4      print(color)
```

terminal

```
$ python for_example.py
red
white
blue
orange
```

Python anti-patterns

These are common for programmers coming from other languages.

unpythonic.py

```
1  i = 0
2  while i < len(colors):
3      print(colors[i])
4      i += 1
5
6  for i in range(len(colors)):
7      print(colors[i])
```

We call them unpythonic.

Additionals

iteration.py

```
# Iteration with values and indices:
for i, color in enumerate(['red', 'yellow', 'blue']):
    print(i, '->', color)

# Taking two sequences together:
for city, population in zip(['Delft', 'Leiden'], [101030, 121562]):
    print(city, '->', population)

# Iterating over a dictionary yields keys:
for key in {'a': 33, 'b': 17, 'c': 18}:
    print(key)

# Iterating over a file yields lines:
for line in open('data/short_file.txt'):
    print(line)
```

The pass statement

If you ever need a statement syntactically but don't want to do anything, use `pass`.

comments_example.py

```
1 while False:
2     # This is never executed anyway.
3     pass
```

Comments

Comments are prepended by # and completely ignored.

comments_example.py

```
1 # Create the list.
2 l = []
3
4 # Add 42 to this list.
5 l.append(42)
```

Hands on!

1. Write a program that prints those numbers which are divisible by 13 and multiple of 5, between 10 and 1313 (both included).