

# Python Programming

## Data Types



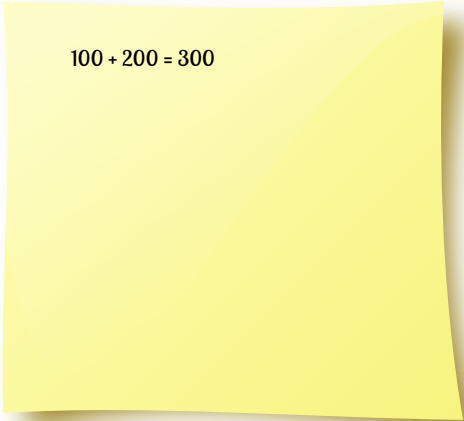
# Introduction

## Problem

What is the sum of two numbers?

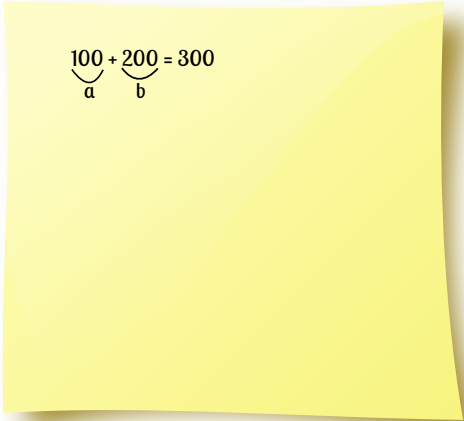
### Problem

What is the sum of two numbers?


$$100 + 200 = 300$$

## Problem

What is the sum of two numbers?

A yellow sticky note is centered on the slide. It contains a mathematical equation:  $\underbrace{100}_a + \underbrace{200}_b = 300$ . The numbers 100 and 200 are underlined with curly braces, and the letters 'a' and 'b' are written below the braces respectively.
$$\underbrace{100}_a + \underbrace{200}_b = 300$$

## Problem

What is the sum of two numbers?

$$\underbrace{100}_a + \underbrace{200}_b = 300$$

---

$$\begin{aligned}a &= 100 \\b &= 200 \\s &= a + b\end{aligned}$$

## Problem

What is the sum of two numbers?

$$\underbrace{100}_a + \underbrace{200}_b = 300$$

---

$a = 100$

$b = 200$

$s = a + b$

---

$a$  = from the user

$b$  = from the user

$s = a + b$

return " $s$ " (print?)

## Problem

What is the sum of two numbers?

$$\underbrace{100}_a + \underbrace{200}_b = 300$$

---

$a = 100$

$b = 200$

$s = a + b$

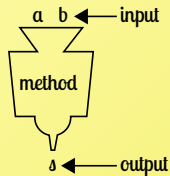
---

$a$  = from the user

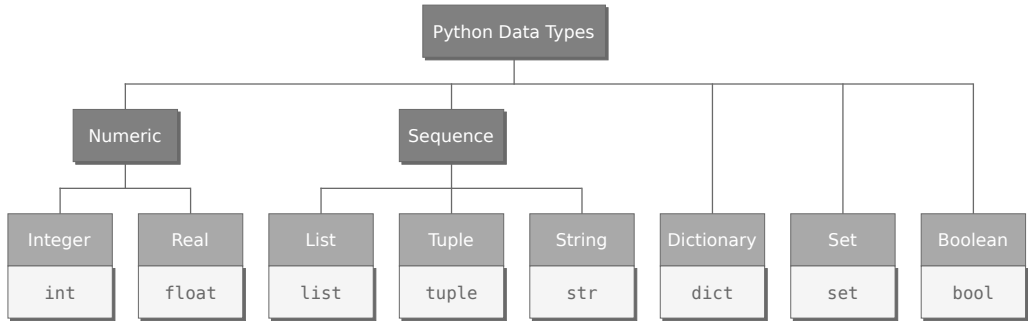
$b$  = from the user

$s = a + b$

return " $s$ " (print?)



# Introduction





## Arithmetic Operators

Operator	Name	Example
<code>+</code>	Addition	<code>a + c</code>
<code>-</code>	Subtraction	<code>a - c</code>
<code>*</code>	Multiplication	<code>a * c</code>
<code>/</code>	Division	<code>a / c</code>
<code>//</code>	Floor division	<code>a // c</code>
<code>%</code>	Modulus	<code>a % c</code>
<code>**</code>	Exponentiation	<code>a ** c</code>

# Numeric Types

Introduction

**Numeric Types**

Sequence Types

Dictionaries

Sets

Booleans

Mutable vs Immutable

Equivalence

Casting

Hands on!

# Numeric Types

## Integer numbers

```
>>> 1
1
>>> 1 + 1
2
>>> 0
0
>>> 0 - 1
-1
>>> type(1)
<class 'int'>
```

## Numeric Types

### Real / floating point numbers

```
>>> 1.0
1.0
>>> 1 + 1.0
2.0
>>> 0
0
>>> 0 - 1.0
-1.0
>>> type(1.0)
<class 'float'>
```

# Numeric Types

## Problem

$$\underbrace{100}_a + \underbrace{200}_b = 300$$

$a = 100$

$b = 200$

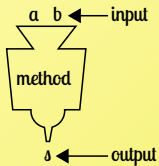
$s = a + b$

$a$  = from the user

$b$  = from the user

$s = a + b$

return " $s$ " (print?)



```
>>> a = 100
>>> b = 200
>>> s = a + b
>>> s
300
```

# Sequence Types

Introduction

Numeric Types

**Sequence Types**

Dictionaries

Sets

Booleans

Mutable vs Immutable

Equivalence

Casting

Hands on!

## Sequence Types

### Lists

Mutable sequences of values.

```
>>> integers = [2, 5, 2, 3, 7]
>>> type(integers)
<class 'list'>
```

Lists can be heterogeneous!

```
>>> a = 'xyz'
>>> [3, 'abc', 1.3e20, [a, a, 2]]
[3, 'abc', 1.3e+20, ['xyz', 'xyz', 2]]
```

### Tuples

Immutable sequences of values.

```
>>> t = 'white', 77, 1.5
>>> type(t)
<class 'tuple'>

>>> color, width, scale = t
>>> width
77
```



### Strings

Immutable sequences of characters.

```
>>> 'a string can be written in single quotes'  
'a string can be written in single quotes'
```

Strings can also be written with double quotes, or over multiple lines with triple-quotes.

```
>>> "this makes it easier to use the ' character"  
"this makes it easier to use the ' character"
```

```
>>> """A multiline string.  
... You see? I continued after a blank line."""  
'A multiline string.\nYou see? I continued after a blank line.'
```

### Strings

But do not mix them!

```
>>> 'a string can not be written with mixed quotes"  
File "<stdin>", line 1  
    'a string can not be written with mixed quotes"  
    ^  
SyntaxError: unterminated string literal (detected at line 1)
```

## Sequence Types

### Strings

A common operation is formatting strings using argument substitutions.

```
>>> '{} times {} equals {:.2f}'.format('pi', 2, 6.283185307179586)
'pi times 2 equals 6.28'
```

Accessing arguments by position or name is more readable.

```
>>> '{1} times {0} equals {2:.2f}'.format('pi', 2, 6.283185307179586)
'2 times pi equals 6.28'

>>> '{number} times {amount} equals {result:.2f}'.format(number='pi', amount=2, result=6.283185307179586)
'pi times 2 equals 6.28'
```

## Sequence Types

### Common sequence operations

All sequence types support: concatenation, membership, indexing, and slicing.

```
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]

>>> 'hay' in 'haystack'
True

>>> 'needle' in 'haystack'
False

>>> 'abcdefghijkl'[3]
'd'
```

### Slicing

Slice `s` from `i` to `j` with `s[i:j]`.

```
>>> 'abcdefghijkl'[4:8]
'efgh'
>>> 'abcdefghijkl'[:3]
'abc'
```

We can also define the step `k` with `s[i:j:k]`.

```
>>> 'abcdefghijkl'[7:3:-1]
'hgfe'
```

### Several helpful builtins

```
>>> len('attacgataggcatccgt')
```

```
18
```

```
>>> max([17, 86, 34, 51])
```

```
86
```

```
>>> sum([17, 86, 34, 51])
```

```
188
```

```
>>> ('atg', 22, True, 'atg').count('atg')
```

```
2
```

### More with lists

We can replace, add, remove, reverse and sort items in-place.

```
>>> l = [1, 2, 3, 4]
>>> l[3] = 7
>>> l.append(1)
>>> l[1:3] = [3, 2]
>>> l.sort()
>>> l.reverse()
>>> l
[7, 3, 2, 1, 1]
```

### Additional useful built-ins

```
>>> list('abcdefghijk')
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k']

>>> range(5, 16)
range(5, 16)

>>> list(range(5, 16))
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

>>> zip(['red', 'white', 'blue'], range(3))
<zip object at 0x7f00199c3a40>

>>> list(zip(['red', 'white', 'blue'], range(3)))
[('red', 0), ('white', 1), ('blue', 2)]
```



Introduction

Numeric Types

Sequence Types

**Dictionaries**

Sets

Booleans

Mutable vs Immutable

Equivalence

Casting

Hands on!

### Unordered map of hashable values to arbitrary objects

```
>>> d = {'a': 27, 'b': 18, 'c': 12}
>>> type(d)
<class 'dict'>

>>> d['e'] = 17
>>> 'e' in d
True

>>> d.update({'a': 18, 'f': 2})
>>> d
{'a': 18, 'b': 18, 'c': 12, 'e': 17, 'f': 2}
```

### Accessing dictionary content

```
>>> d['b']
18

>>> d.keys()
dict_keys(['a', 'b', 'c', 'e', 'f'])

>>> list(d.keys())
['a', 'b', 'c', 'e', 'f']

>>> list(d.values())
[18, 18, 12, 17, 2]

>>> list(d.items())
[('a', 18), ('b', 18), ('c', 12), ('e', 17), ('f', 2)]
```

# Sets

Introduction

Numeric Types

Sequence Types

Dictionaries

**Sets**

Booleans

Mutable vs Immutable

Equivalence

Casting

Hands on!

**Mutable unordered collections of hashable values without duplication**

```
>>> x = {12, 28, 21, 17}
```

```
>>> type(x)
```

```
<class 'set'>
```

```
>>> x.add(12)
```

```
>>> x
```

```
{17, 28, 12, 21}
```

```
>>> x.discard(21)
```

```
>>> x
```

```
{17, 28, 12}
```

## Sets are not indexable

Sets are unordered collections, and therefore without index.

```
>>> x[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
```

In contrast to lists ...

```
>>> list(x)[0]
17
```

## Operations

We can test for membership and apply many common set operations such as union and intersect.

```
>>> 17 in {12, 28, 21, 17}
```

```
True
```

```
>>> {12, 28, 21, 17} | {12, 18, 11}
```

```
{17, 18, 21, 11, 28, 12}
```

```
>>> {12, 28, 21, 17} & {12, 18, 11}
```

```
{12}
```

## Operations

```
>>> s1 = {12, 28, 21, 17}
```

```
>>> s2 = {28, 32, 71, 12}
```

```
>>> s1 - s2  
{17, 21}
```

```
>>> {1,3}.issubset({1,2,3})  
True
```



# Booleans

Introduction

Numeric Types

Sequence Types

Dictionaries

Sets

**Booleans**

Mutable vs Immutable

Equivalence

Casting

Hands on!

## Booleans

The two boolean values are written `False` and `True`.

```
>>> True or False
```

```
True
```

```
>>> True and False
```

```
False
```

```
>>> not False
```

```
True
```

## Comparisons

Comparisons can be done on all objects and return a boolean value.

```
>>> 1 < 2
```

```
True
```

```
>>> 1 == 2
```

```
False
```

```
>>> "Left" == "Right"
```

```
False
```

```
>>> "Right" == "Right"
```

```
True
```

# Mutable vs Immutable

Introduction

Numeric Types

Sequence Types

Dictionaries

Sets

Booleans

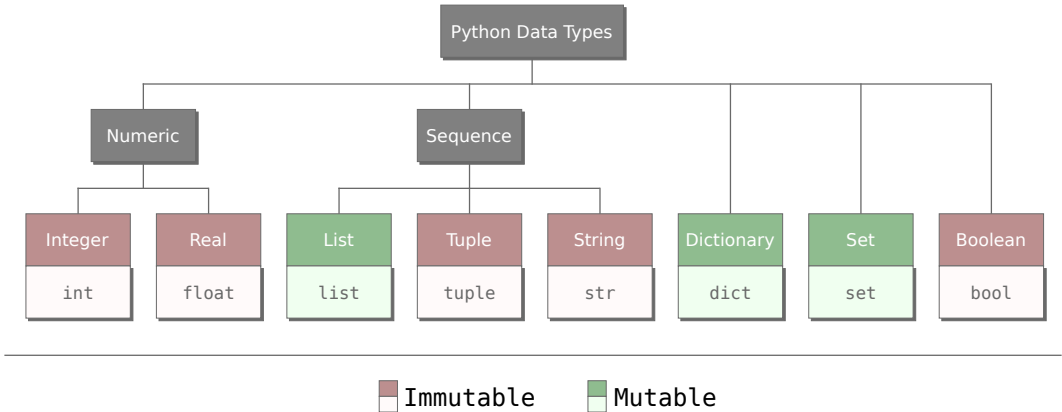
**Mutable vs Immutable**

Equivalence

Casting

Hands on!

# Mutable vs Immutable



## Mutable vs Immutable

### Mutable data types

Objects whose value can change are said to be mutable.

```
>>> a = [1,2,3,4]
```

```
>>> a.append(5)
```

```
>>> a[2] = 6
```

```
>>> a
```

```
[1, 2, 6, 4, 5]
```

```
>>> b = {1,2}
```

```
>>> b.add(3)
```

```
>>> b
```

```
{1, 2, 3}
```

## Mutable vs Immutable

### Immutable data types

Objects whose value is unchangeable once they are created are called immutable.

```
>>> (1,2,3,4)[2] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

NB. In Python 3.8, range is also a data type and immutable.

# Equivalence

Introduction

Numeric Types

Sequence Types

Dictionaries

Sets

Booleans

Mutable vs Immutable

**Equivalence**

Casting

Hands on!



### Value vs object

We have two equivalence relations: value equality (==) and object identity (is).

```
>>> a, b = [1, 2, 3], [1, 2, 3]
```

```
>>> a == b
```

```
True
```

```
>>> a is b
```

```
False
```

```
>>> a = 0
```

```
>>> b = 0
```

```
>>> a is b
```

```
True
```

### Changing the type of a value

Sometimes you might want to combine values of different types.

```
>>> x = 1
>>> name = 'John'
>>> name + x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

### Combining different types

```
>>> x = 1
>>> name = 'John'
>>> name + str(x)
'John1'
```

And further ...

```
>>> x = 1
>>> x
1
>>> str(x)
'1'
>>> int(str(x))
1
```

## Hands on!

1. Make a list `list1` with 10 integer elements.
  - a What is the sum of all the items in the `list1` list.
  - b Make a list `list2` from `list1` that does not include the 0th, 4th, and 5th elements.
  - c Sum only the elements from `list1` which are between the 2nd and the 6th elements.
2. Food:
  - a. Create a dictionary for food products called `prices` and put some values in it, e.g.,  
`'apples': 2, 'oranges': 1.5, 'pears': 3, ...`
  - b. Create a corresponding dictionary called `stock` and put the stock values in it, e.g.,  
`'apples': 0, 'oranges': 1, 'pears': 10, ...`
  - c. Add another entry in the `prices` dictionary with key `'bananas'` and value `13`.
  - d. Add another entry in the `stocks` dictionary with key `'bananas'` and value `11`.
  - e. What is the total money value for the `'bananas'` (`stock × price`)?
  - f. How many products are in the `stocks` dictionary?
  - g. Are the number of products in the `stocks` and `prices` dictionaries equal?
  - h. Are there the same products in the `stocks` and `prices` dictionaries?
  - i. What is the most expensive value in the `prices` dictionary?

## Acknowledgements

Mihai Lefter  
Martijn Vermaat  
Jeroen Laros  
Jonathan Vis

