Ghent University
Faculty of Sciences

Python Programming

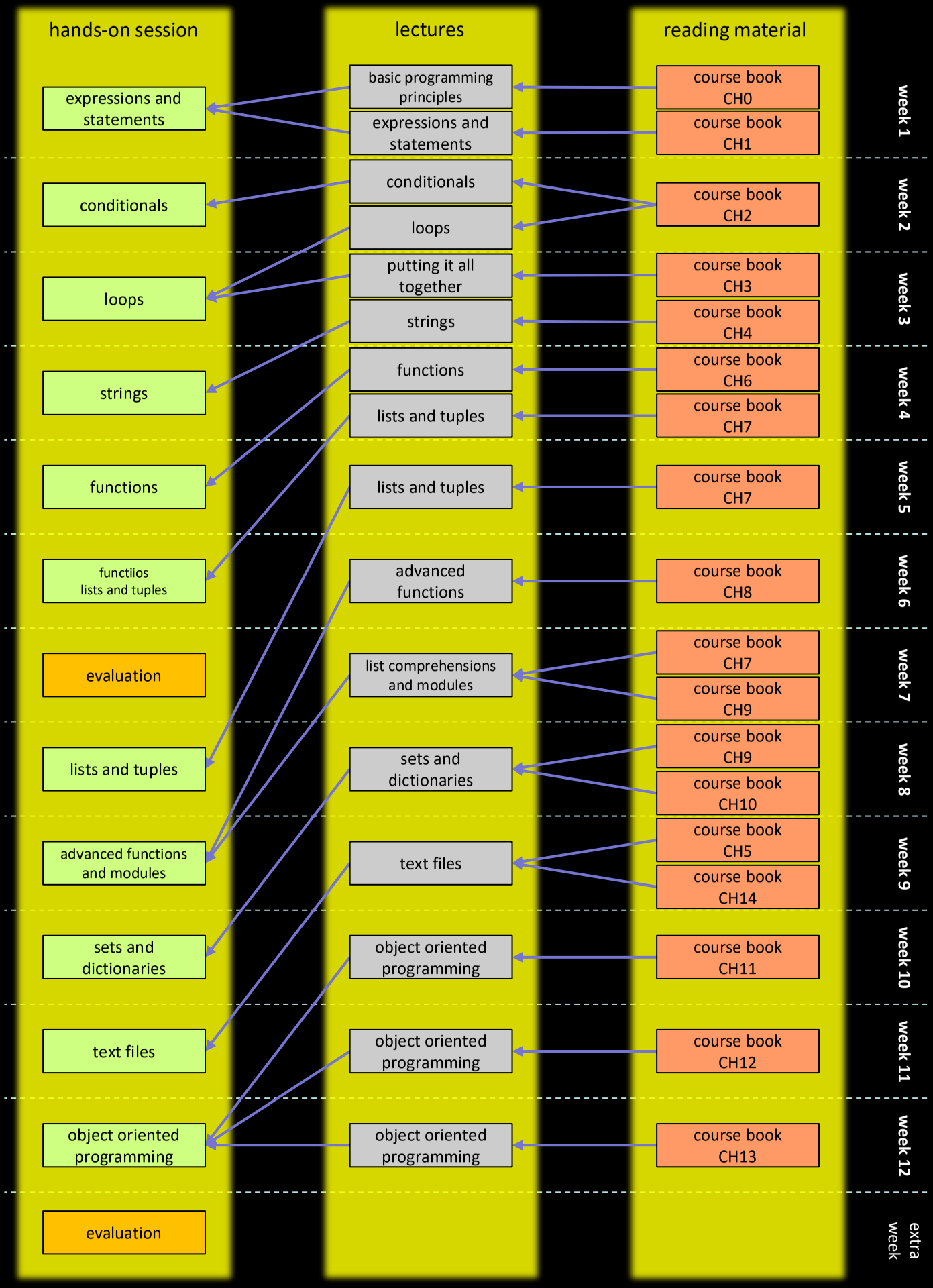
advanced functions

Prof. Dr. Peter Dawyndt

peter.dawyndt@ugent.be

@dawyndt

UNIVERSITEIT GENT


hands-on session
lectures
reading material
week 1
week 2
week 3
week 4
week 5
week 6
week 7
week 8
week 9
week 10
week 11
week 12
extra week
expressions and statements
conditionals
loops
strings
functions
functiios lists and tuples
evaluation
lists and tuples
advanced functions and modules
sets and dictionaries
text files
object oriented programming
evaluation
basic programming principles
expressions and statements
conditionals
loops
putting it all together
strings
functions
lists and tuples
lists and tuples
advanced functions
list comprehensions and modules
sets and dictionaries
text files
object oriented programming
object oriented programming
object oriented programming
course book CH0
course book CH1
course book CH2
course book CH3
course book CH4
course book CH6
course book CH7
course book CH7
course book CH8
course book CH7
course book CH9
course book CH9
course book CH10
course book CH5
course book CH14
course book CH11
course book CH12
course book CH13

Default params / keyword args

Default parameters

- function can specify default value for some of its parameters
  - just "assign" some value to the parameter in the definition
  - used when no value is explicitely passed to this parameters

```
>>> def keeper(name, quest='Holy Grail', color='Blue'):
...     print('What is your name?')
...     print(f'My name is {name}.')
...     print('What is your quest?')
...     print(f'To seek the {quest}.')
...     print('What is your favorite color?')
...     print(f'{color}.')

>>> keeper('Sir Launcelot of Camelot')
```

UNIVERSITEIT GENT

Default parameters

- function can specify default value for some of its parameters
  - just "assign" some value to the parameter in the definition
  - used when no value is explicitely passed to this parameters

```
>>> def keeper(name, quest='Holy Grail', color='Blue'):
...     print('What is your name?')
...     print(f'My name is {name}.')
...     print('What is your quest?')
...     print(f'To seek the {quest}.')
...     print('What is your favorite color?')
...     print(f'{color}.')

>>> keeper('Sir Launcelot of Camelot')
What is your name?
My name is Sir Launcelot of Camelot.
What is your quest?
To seek the Holy Grail.
What is your favorite color?
Blue.
```

UNIVERSITEIT GENT

Default parameters

- function can specify default value for some of its parameters
  - just "assign" some value to the parameter in the definition
  - used when no value is explicitely passed to this parameters

```
>>> def keeper(name, quest='Holy Grail', color='Blue'):
...     print('What is your name?')
...     print(f'My name is {name}.')
...     print('What is your quest?')
...     print(f'To seek the {quest}.')
...     print('What is your favorite color?')
...     print(f'{color}.')

>>> keeper('Arthur, King of the Britons', 'Holy Hand Grenade')
What is your name?
My name is Arthur, King of the Britons.
What is your quest?
To seek the Holy Hand Grenade.
What is your favorite color?
Blue.
```

UNIVERSITEIT GENT

Default parameters

- function can specify default value for some of its parameters
  - just "assign" some value to the parameter in the definition
  - used when no value is explicitely passed to this parameters

```
>>> def keeper(name, quest='Holy Grail', color='Blue'):
...     print('What is your name?')
...     print(f'My name is {name}.')
...     print('What is your quest?')
...     print(f'To seek the {quest}.')
...     print('What is your favorite color?')
...     print(f'{color}.')

>>> keeper('Sir Galahad', 'Holy Grail', 'Blue. No yellow')
What is your name?
My name is Sir Galahad.
What is your quest?
To seek the Holy Grail.
What is your favorite color?
Blue. No yellow.
```

UNIVERSITEIT GENT

Keyword arguments

- function can specify default value for some of its parameters
  - just "assign" some value to the parameter in the definition
  - used when no value is explicitely passed to this parameters
  - specified argument takes precedence over default value
  - *positional arguments*
    - arguments actually passed when function is called are matched up left to right to the parameters of the function
    - all parameters with default values (*optional parameters*) should come *after* parameters without them (*mandatory parameters*)
      - matching arguments to parameters would otherwise be ambiguous
      - an argument must be passed for each obligatory parameter
  - *keyword arguments*
    - useful with functions having lots of optional parameters
    - makes function calls easier to read

UNIVERSITEIT GENT

Keyword arguments

- function can specify default value for some of its parameters
  - *keyword arguments*
    - useful with functions having lots of optional parameters

```
>>> def keeper(name, quest='Holy Grail', color='Blue'):
...     print('What is your name?')
...     print(f'My name is {name}.')
...     print('What is your quest?')
...     print(f'To seek the {quest}.')
...     print('What is your favorite color?')
...     print(f'{color}.')

>>> keeper('Sir Galahad', color='Blue. No yellow')
```


  - *keyword arguments*
    - useful with functions having lots of optional parameters

UNIVERSITEIT GENT

Keyword arguments

- function can specify default value for some of its parameters
  - *keyword arguments*
    - useful with functions having lots of optional parameters

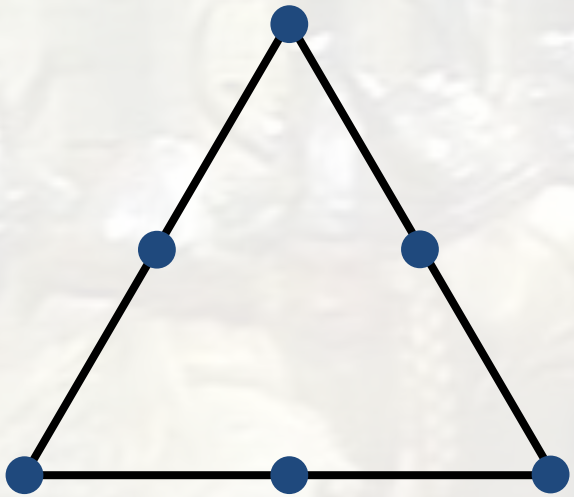
```
>>> def keeper(name, quest='Holy Grail', color='Blue'):
...     print('What is your name?')
...     print(f'My name is {name}.')
...     print('What is your quest?')
...     print(f'To seek the {quest}.')
...     print('What is your favorite color?')
...     print(f'{color}.')

>>> keeper('Sir Galahad', color='Blue. No yellow')
What is your name?
My name is Sir Galahad.
What is your quest?
To seek the Holy Grail.
What is your favorite color?
Blue. No yellow.
```

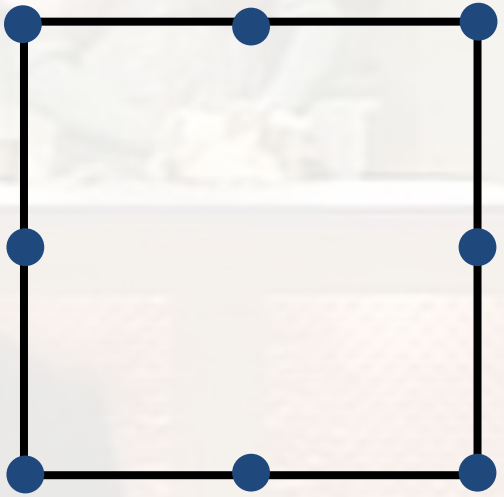
UNIVERSITEIT GENT

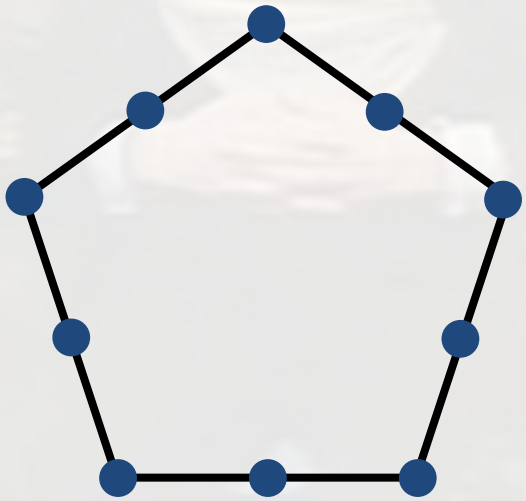
Pub game


1
6
12
SUBMITTED BY P. DAWYNDT
10
2
there are three
kinds of people:
8
5
those who can count,
and those who can't
4
11
Hoegaarden
7
9
3

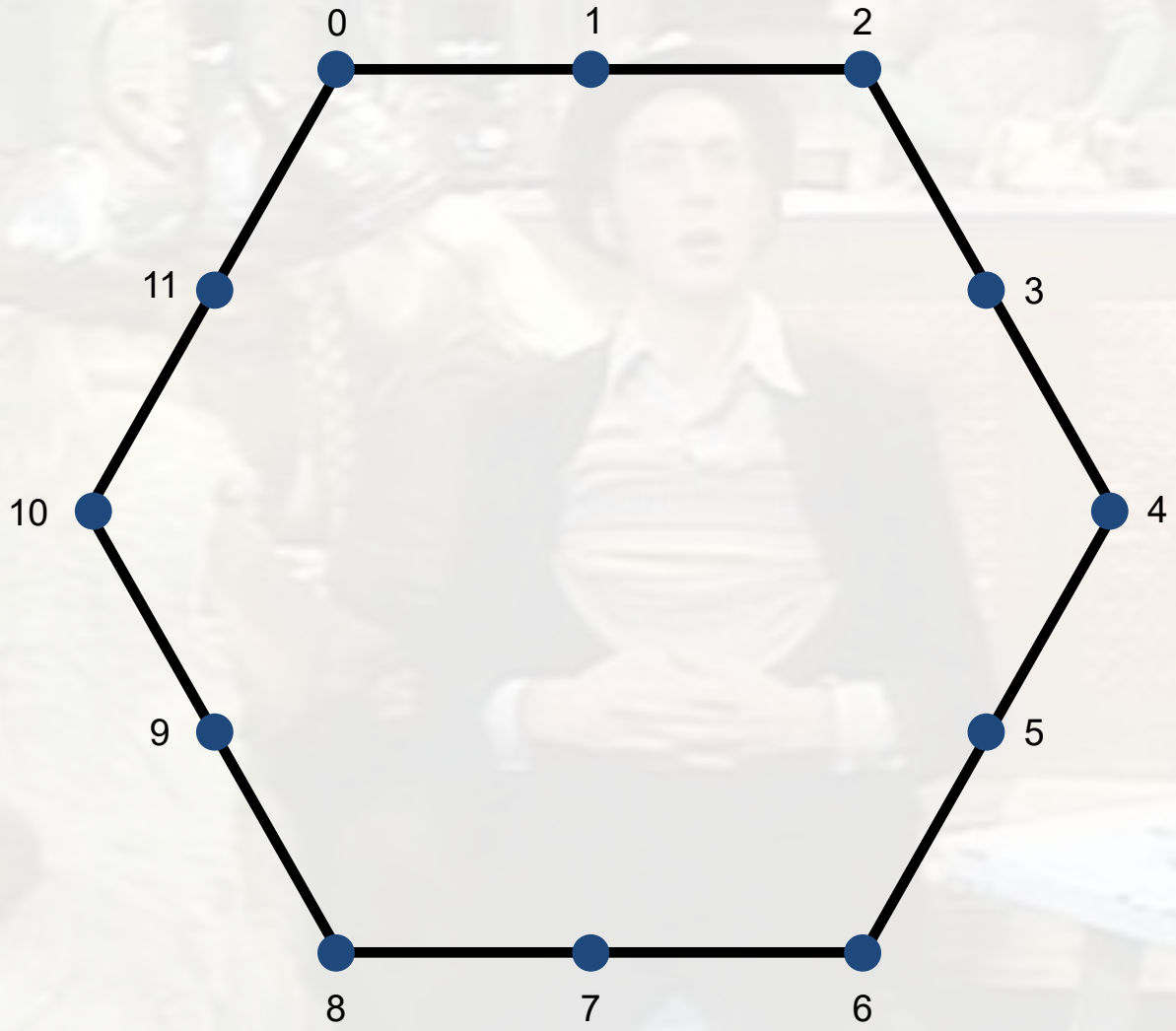
Pub game



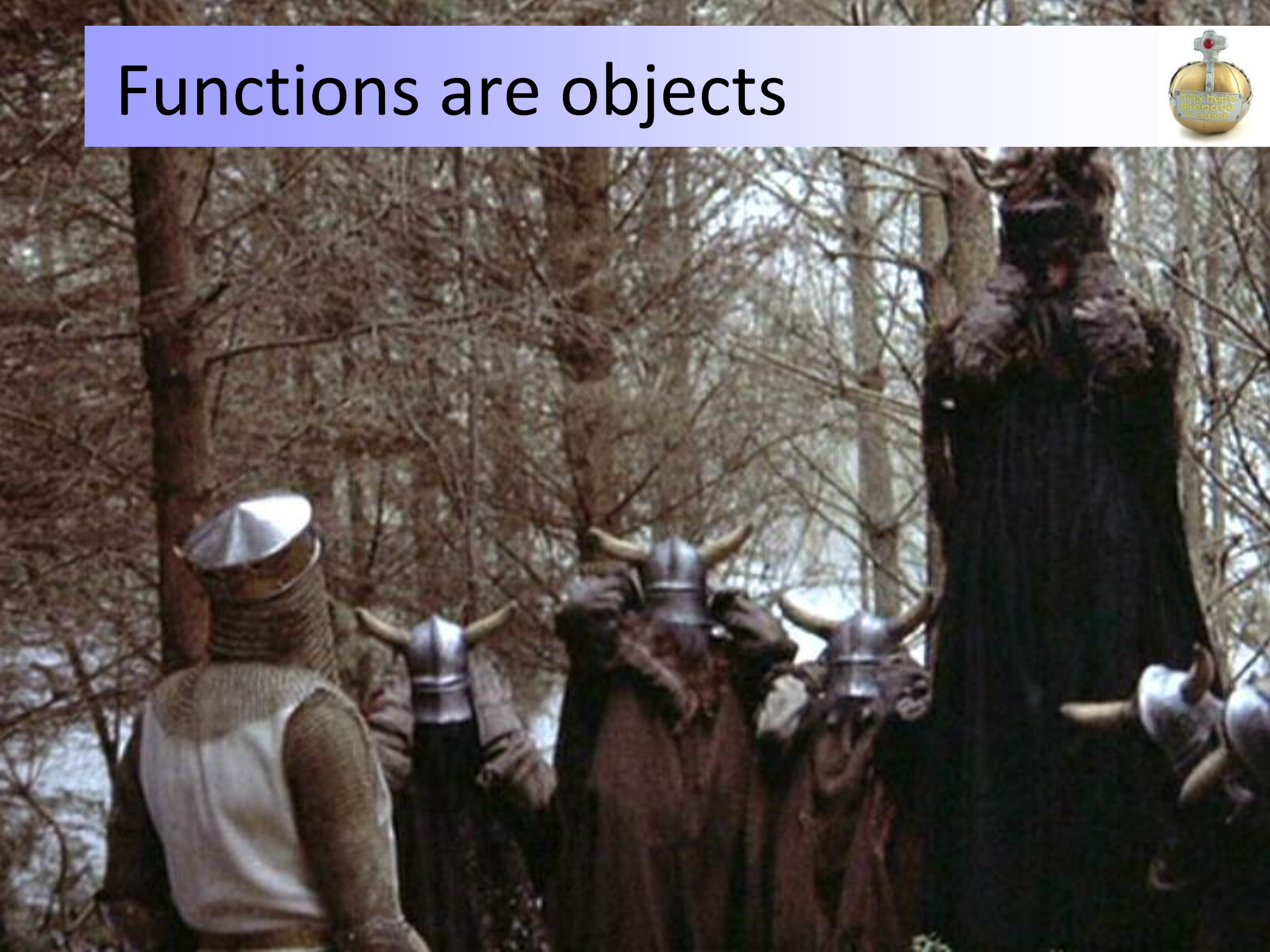




Pub game

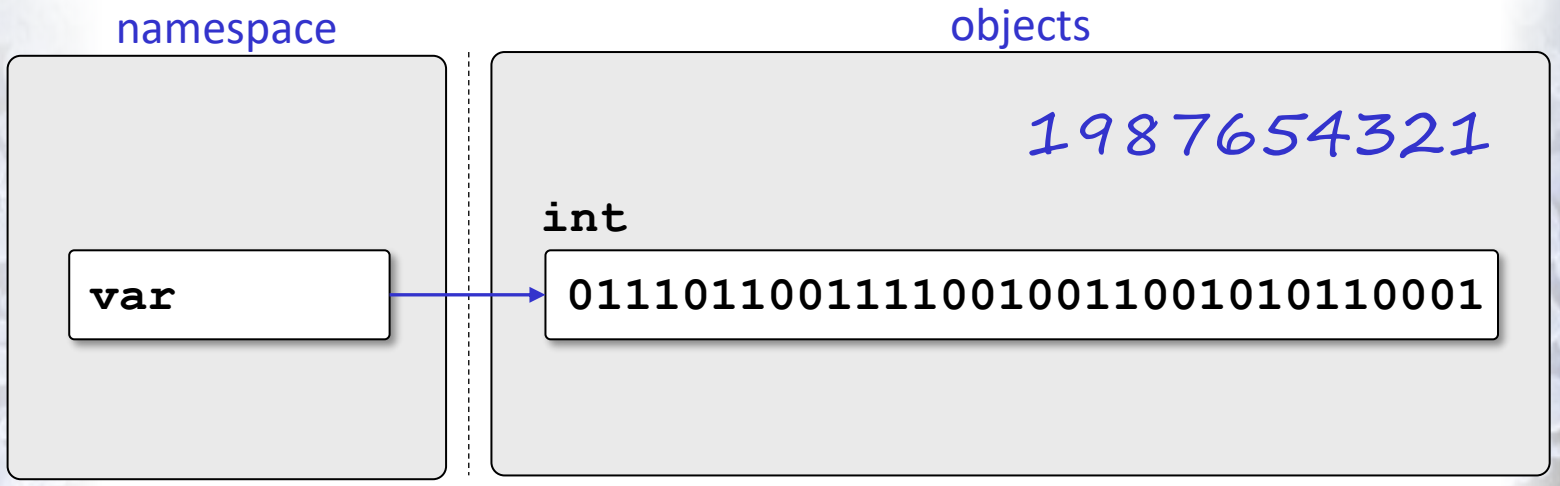


Functions are objects



Functions are objects

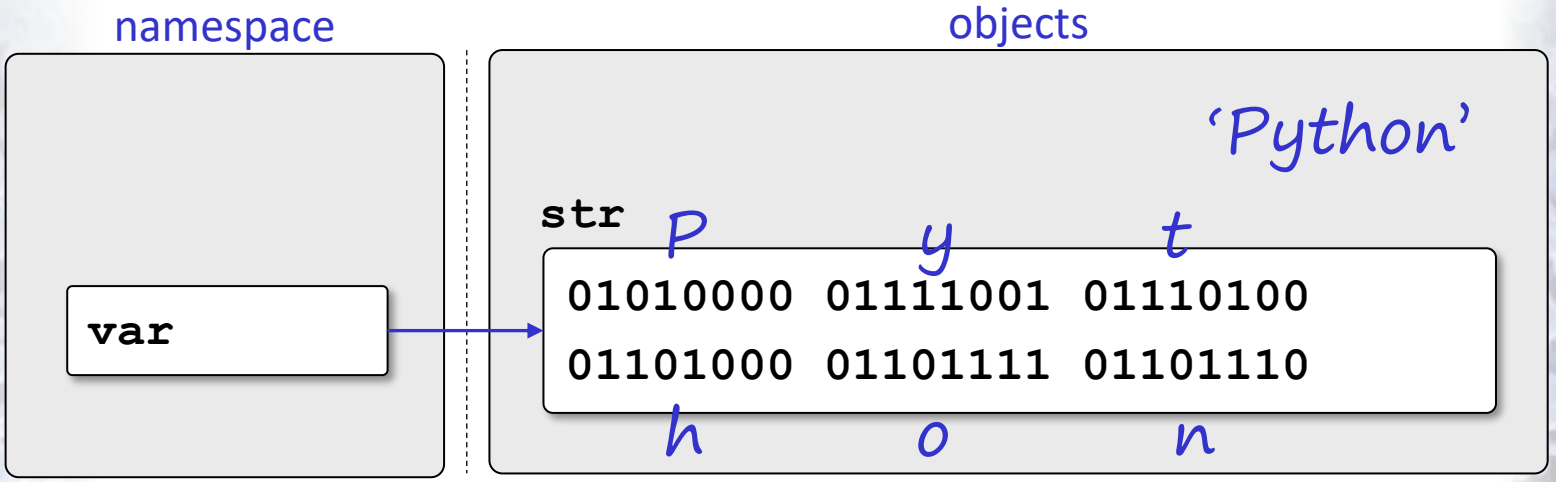
- an integer 32 bits of data …
  - … that variables can refer to



UNIVERSITEIT GENT

Functions are objects

- an integer 32 bits of data …
  - … that variables can refer to
- a string is a sequence of bytes representing characters …
  - … that variables can refer to
- a function is a sequence of bytes representing instructions …
  - … and yes, variables can refer to them too



Functions are objects

functions1.py

```
def perimeter(radius):
    from math import pi
    return 2 * pi * radius

radius = 2.87
print(perimeter(radius))
```

globals

namespace stack

objects

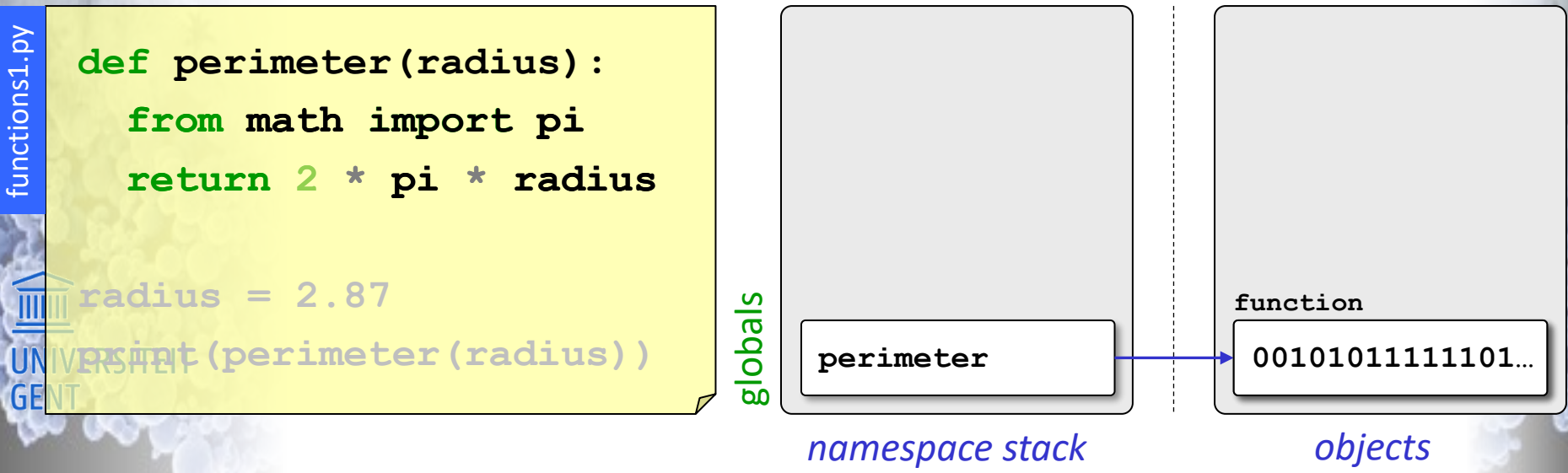
Functions are objects

- a function is just another object
  - happens to be an object that can be called
  - just as strings and lists are objects can be indexed
  - **`def`** is a shorthand for

"Create a function object, and assign it to a variable."

functions1.py

```
def perimeter(radius):
    from math import pi
    return 2 * pi * radius

radius = 2.87
print(perimeter(radius))
```



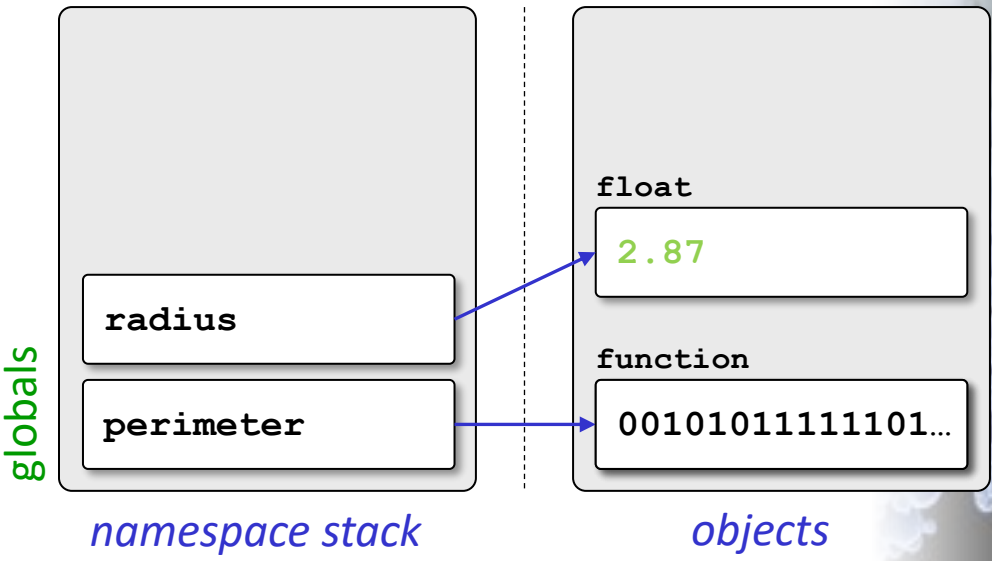
Functions are objects

- a function is just another object
  - happens to be an object that can be called
  - just as strings and lists are objects can be indexed
  - **`def`** is a shorthand for

"Create a function object, and assign it to a variable."

functions1.py

```
def perimeter(radius):
    from math import pi
    return 2 * pi * radius

radius = 2.87
print(perimeter(radius))
```



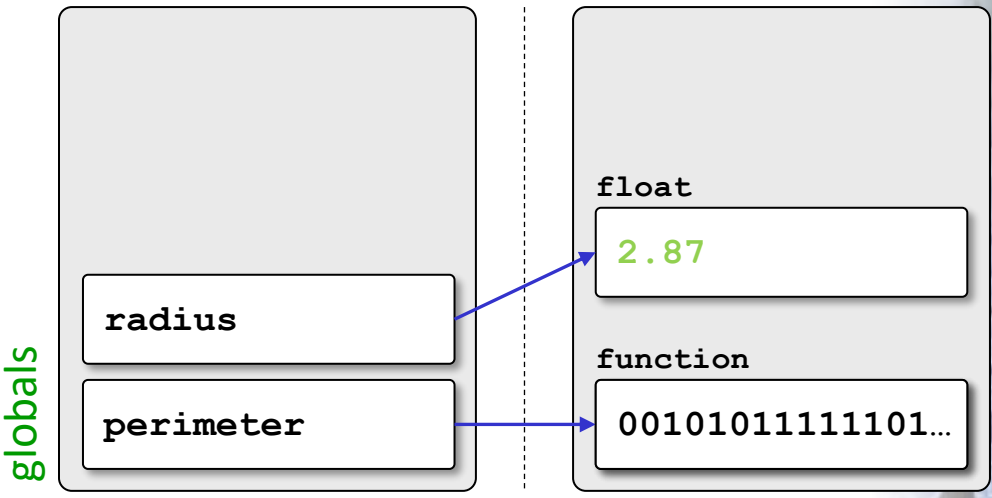
Functions are objects

- a function is just another object
  - happens to be an object that can be called
  - just as strings and lists are objects can be indexed
  - **`def`** is a shorthand for

"Create a function object, and assign it to a variable."

functions1.py

```
def perimeter(radius):
    from math import pi
    return 2 * pi * radius

radius = 2.87
print(perimeter(radius))
```



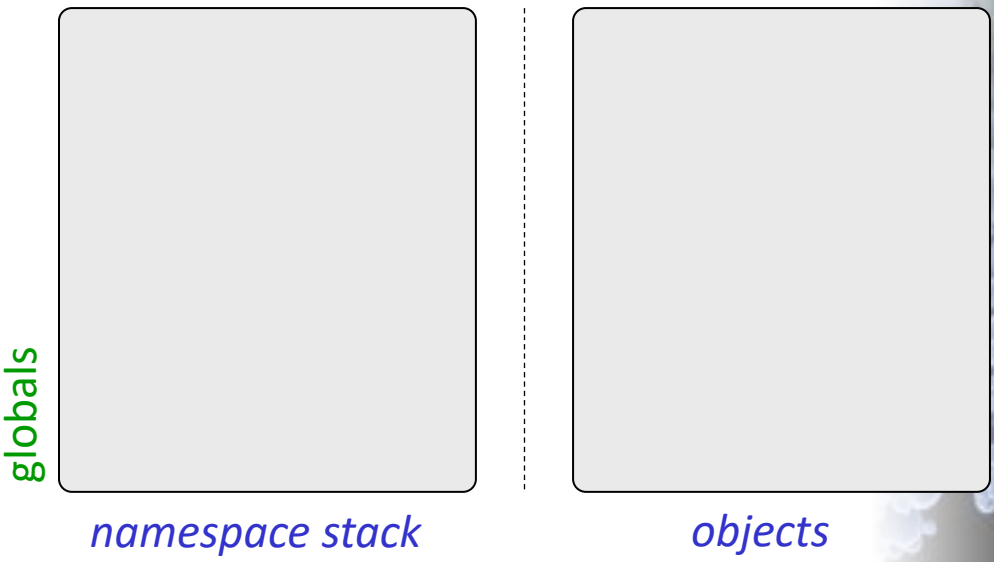
18.0327418316

Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)

functions2.py

```
def perimeter(radius):
    return 2 * 3.14 * radius

def perimeter(radius):
    from math import pi
    return 2 * pi * radius

print(perimeter(2.87))
```

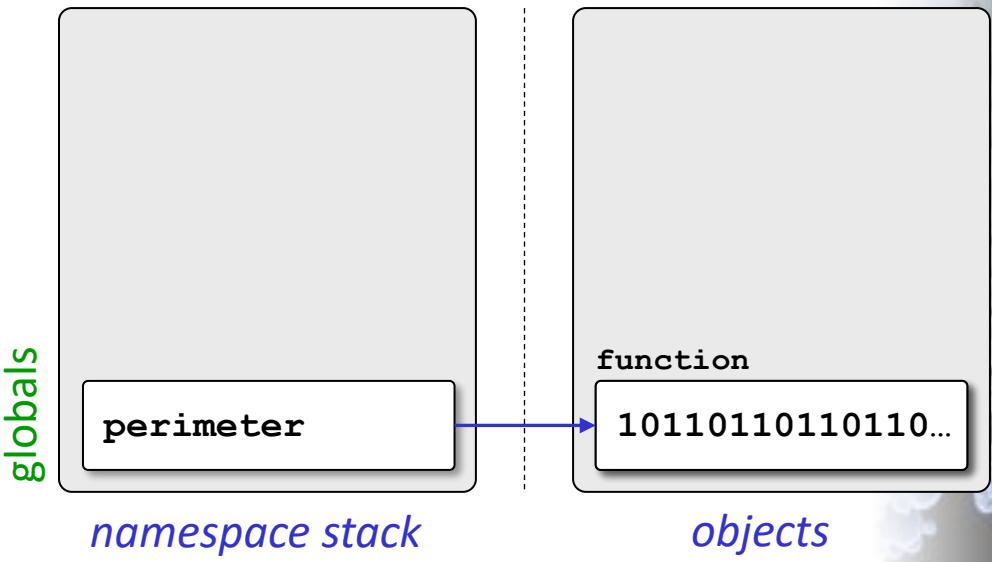


Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)

functions2.py

```
def perimeter(radius):
    return 2 * 3.14 * radius

def perimeter(radius):
    from math import pi
    return 2 * pi * radius

print(perimeter(2.87))
```

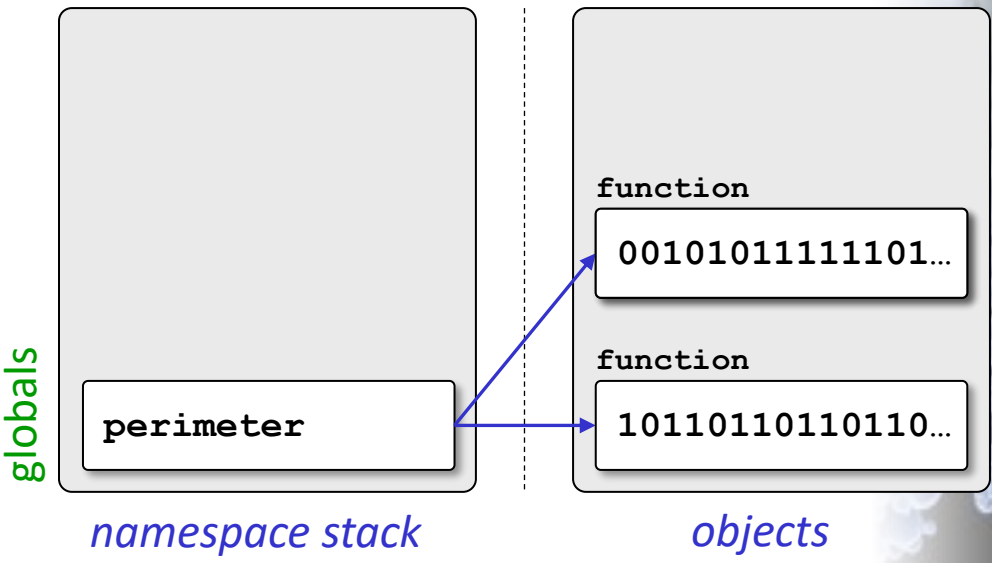


Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)

functions2.py

```
def perimeter(radius):
    return 2 * 3.14 * radius

def perimeter(radius):
    from math import pi
    return 2 * pi * radius

print(perimeter(2.87))
```

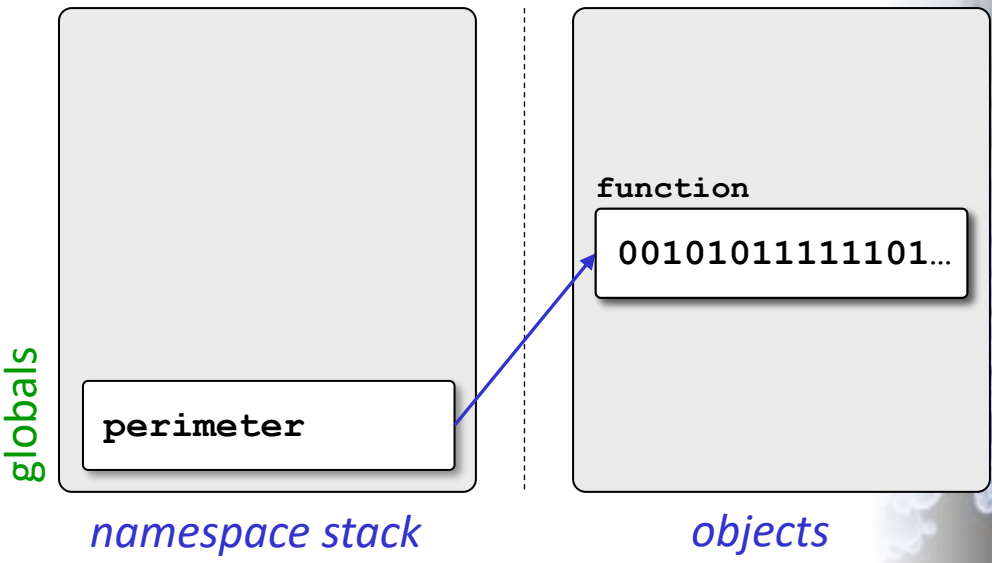


Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)

functions2.py

```
def perimeter(radius):
    return 2 * 3.14 * radius

def perimeter(radius):
    from math import pi
    return 2 * pi * radius

print(perimeter(2.87))
```

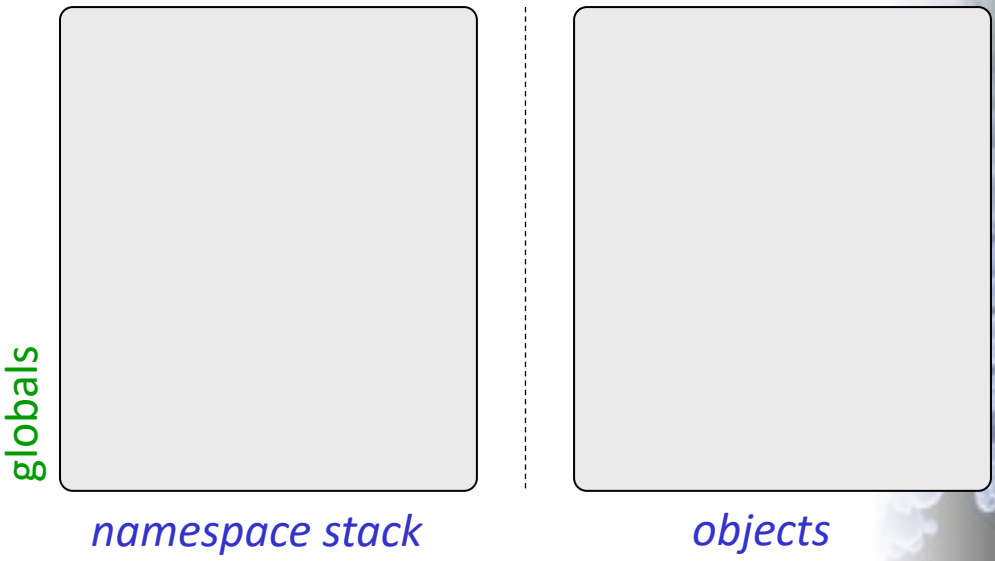


Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)
  - aliased (renamed)

functions3.py

```
def perimeter(radius):
    from math import pi
    return 2 * pi * radius

contour = perimeter
print(contour(2.87))
```

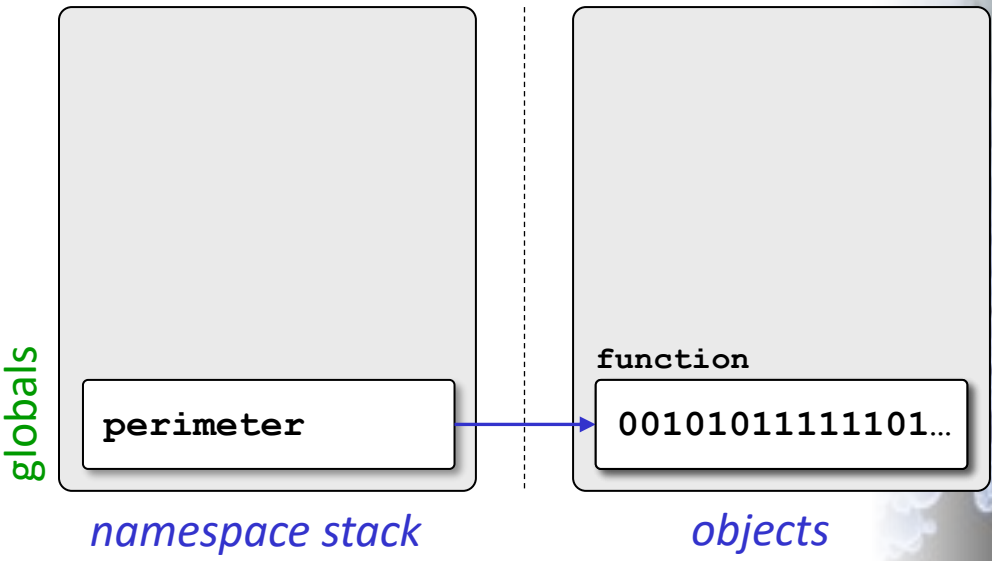


Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)
  - aliased (renamed)

functions3.py

```
def perimeter(radius):
    from math import pi
    return 2 * pi * radius

contour = perimeter
print(contour(2.87))
```

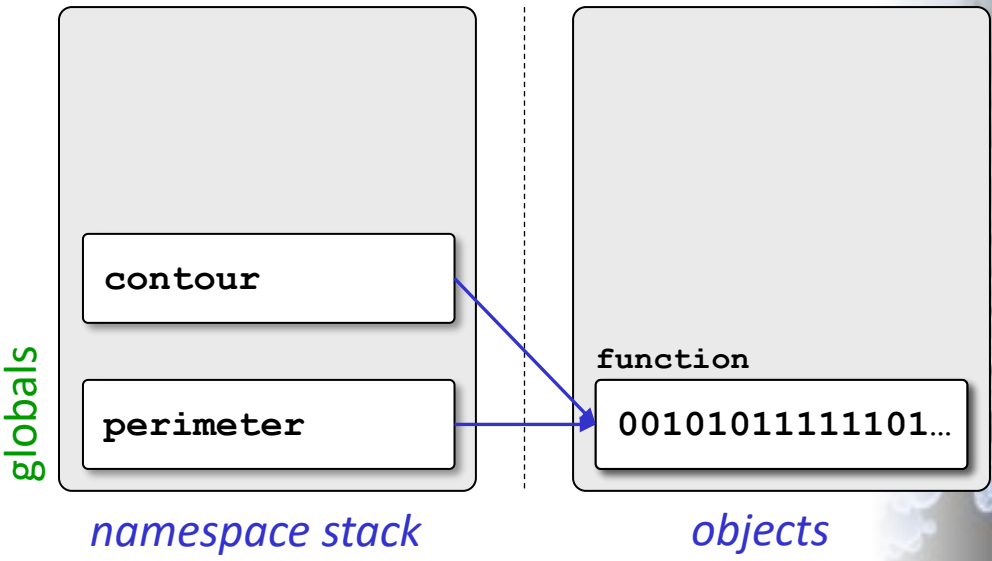


Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)
  - aliased (renamed)

functions3.py

```
def perimeter(radius):
    from math import pi
    return 2 * pi * radius

contour = perimeter
print(contour(2.87))
```

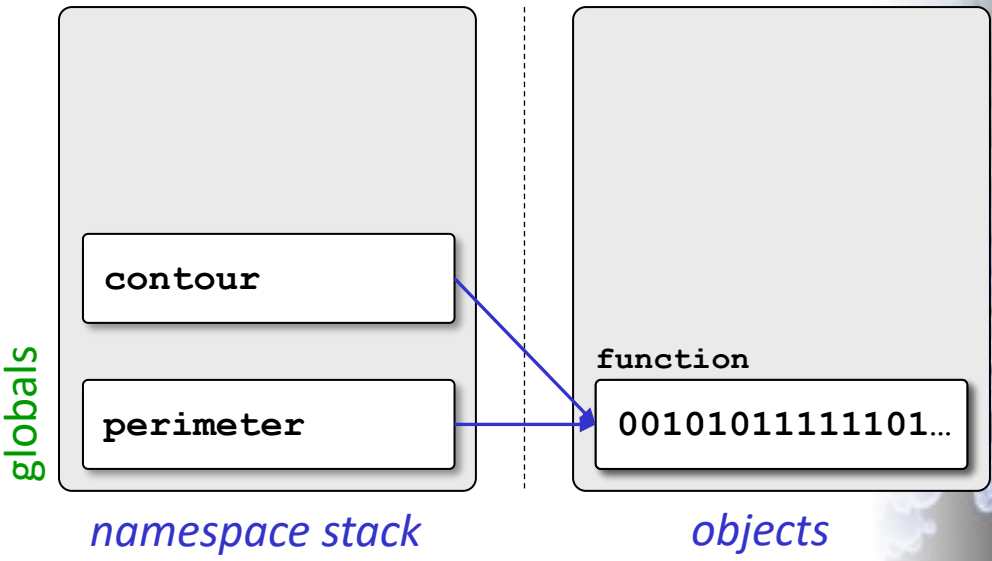


Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)
  - aliased (renamed)

functions3.py

```
def perimeter(radius):
    from math import pi
    return 2 * pi * radius

contour = perimeter
print(contour(2.87))
```



Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)
  - aliased (renamed)
  - passed as arguments to other functions

functions4.py

```
from math import pi

def perimeter(radius):
    return 2 * pi * radius

def area(radius):
    return pi * radius ** 2

def compute(function, value):
    return function(value)

print(compute(perimeter, 2.87))
print(compute(area, 2.87))
```

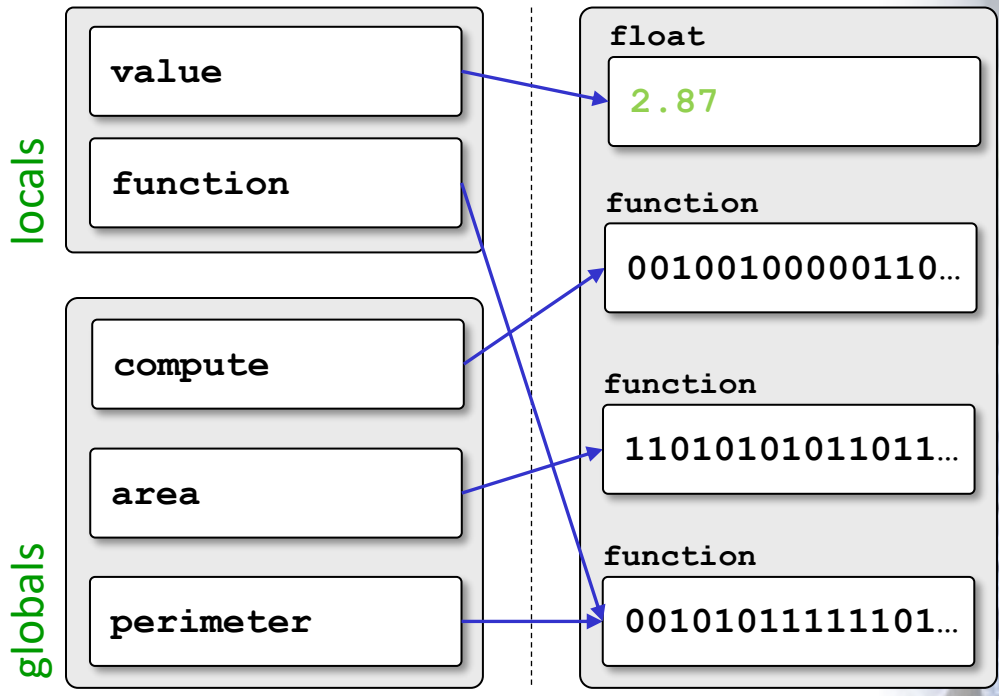
Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)
  - aliased (renamed)
  - passed as arguments to other functions

functions4.py

```
from math import pi

def perimeter(radius):
    return 2 * pi * radius

def area(radius):
    return pi * radius ** 2

def compute(function, value):
    return function(value)

print(compute(perimeter, 2.87))
print(compute(area, 2.87))
```

18.0327418316



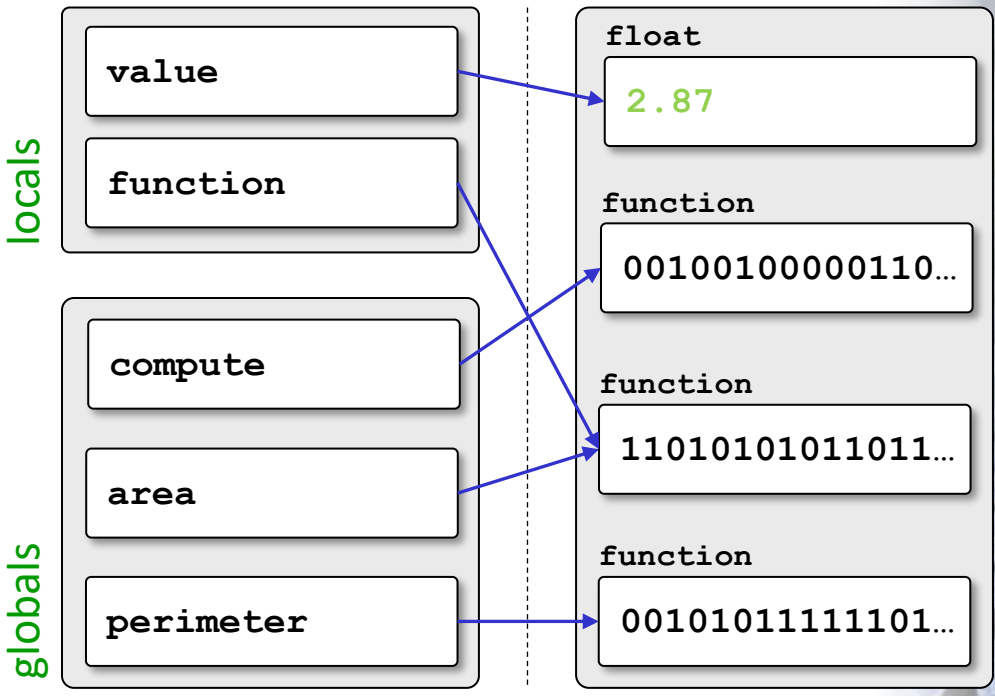
Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)
  - aliased (renamed)
  - passed as arguments to other functions

functions4.py

```
from math import pi

def perimeter(radius):
    return 2 * pi * radius

def area(radius):
    return pi * radius ** 2

def compute(function, value):
    return function(value)

print(compute(perimeter, 2.87))
print(compute(area, 2.87))
```

```
18.0327418316
25.8769845284
```

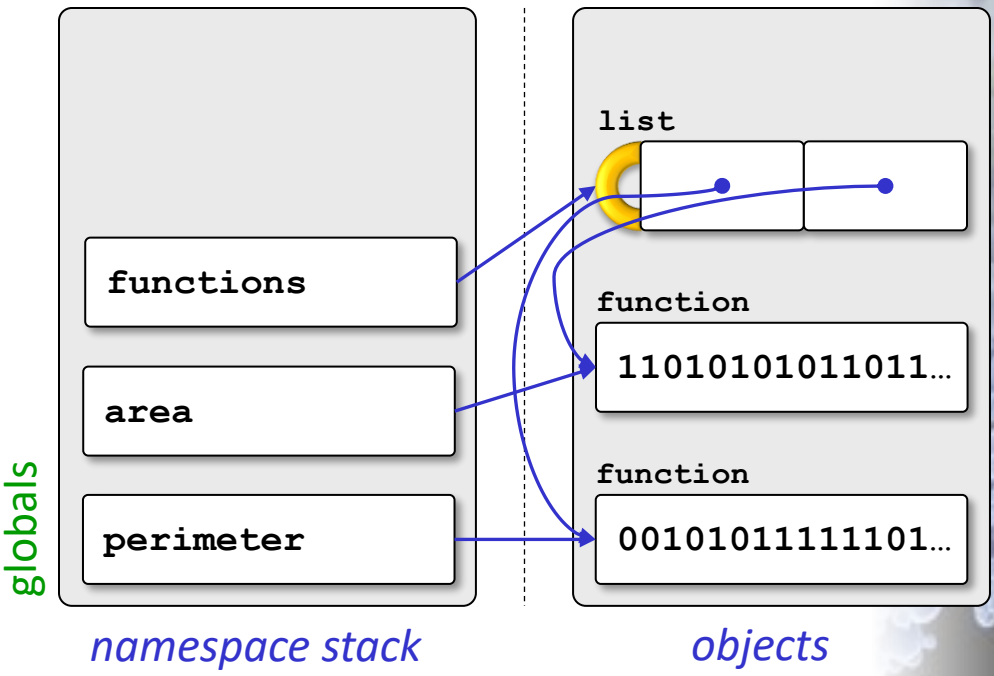


Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)
  - aliased (renamed)
  - passed as arguments to other functions
  - stored as elements in lists (or other container types)

functions5.py

```
from math import pi

def perimeter(radius):
    return 2 * pi * radius

def area(radius):
    return pi * radius ** 2

functions = [perimeter, area]
for function in functions:
    print(function(2.87))
```



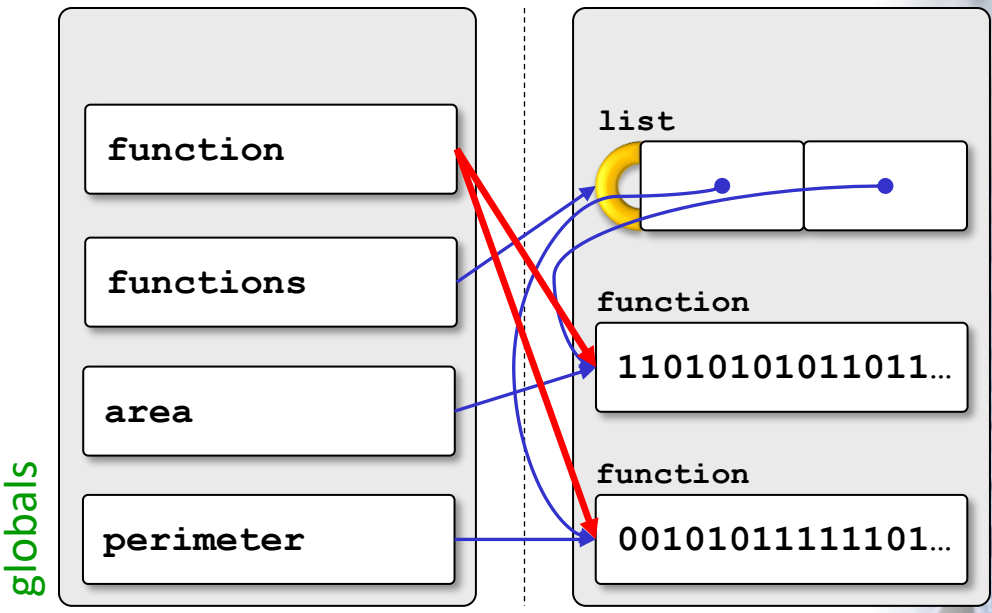
Functions are objects

- this means that functions can be
  - redefined (just variables can be reassigned a new value)
  - aliased (renamed)
  - passed as arguments to other functions
  - stored as elements in lists (or other container types)

functions5.py

```
from math import pi

def perimeter(radius):
    return 2 * pi * radius

def area(radius):
    return pi * radius ** 2

functions = [perimeter, area]
for function in functions:
    print(function(2.87))
```

```
18.0327418316
25.8769845284
```



Functions are objects

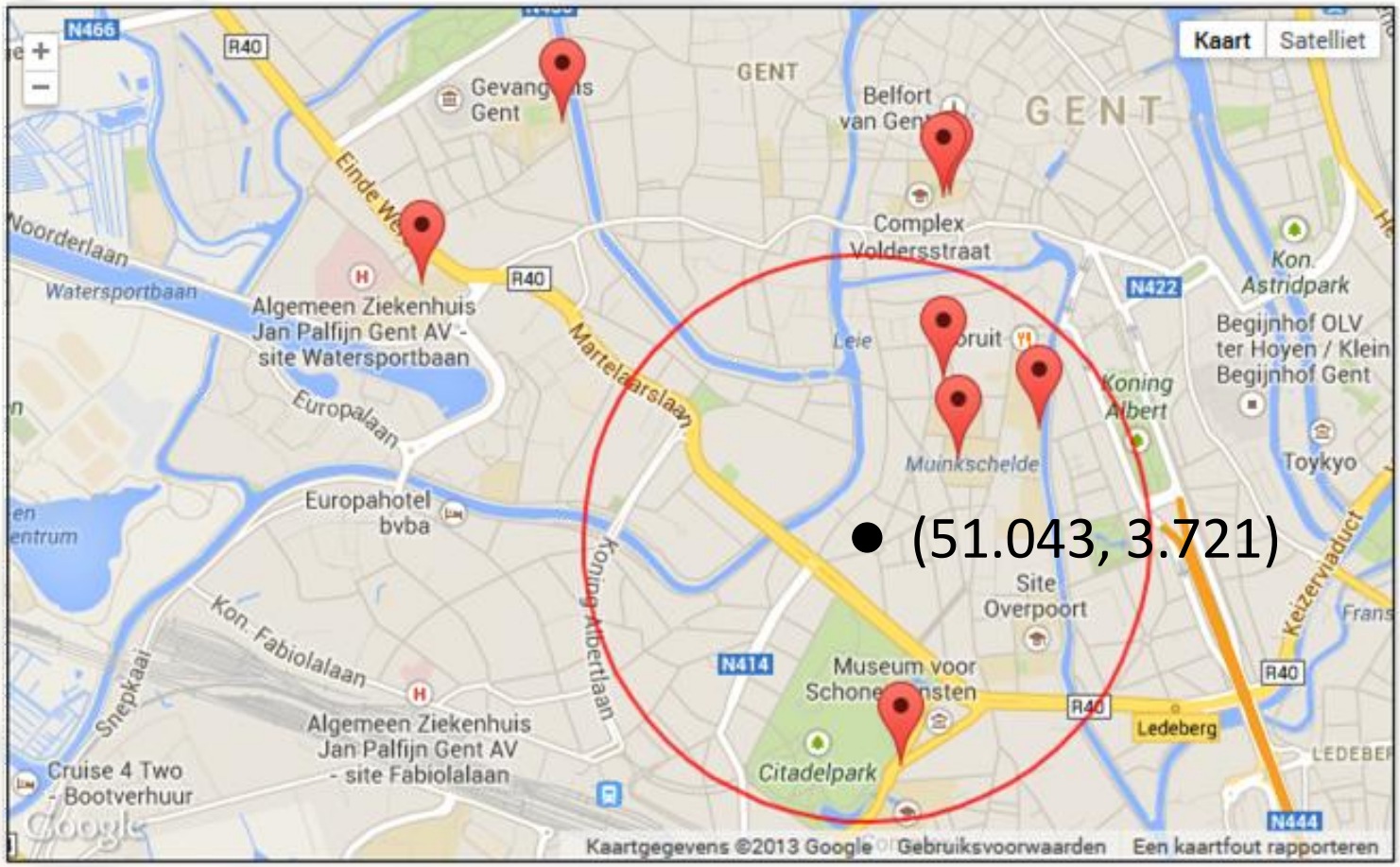
- function object also has properties and methods

```
>>> def perimeter(radius):
...     """Calculates perimeter of circle with given radius."""
...     return 2 * radius * pi
...
>>> dir(perimeter)
['__annotations__', '__call__', '__class__', '__closure__', '__code__', '__defaults__',
'__delattr__', '__dict__', '__doc__', '__eq__', '__format__', '__ge__', '__get__',
'__getattribute__', '__globals__', '__gt__', '__hash__', '__init__', '__kwdefaults__',
'__le__', '__lt__', '__module__', '__name__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__']
>>> perimeter.__name__
'perimeter'
>>> perimeter.__doc__
'Calculates perimeter of circle with given radius.'
>>> help(perimeter)
Help on function perimeter in module __main__:

perimeter(radius)
    Calculate perimeter of circle with given radius.
```

docstring

UNIVERSITEIT GENT

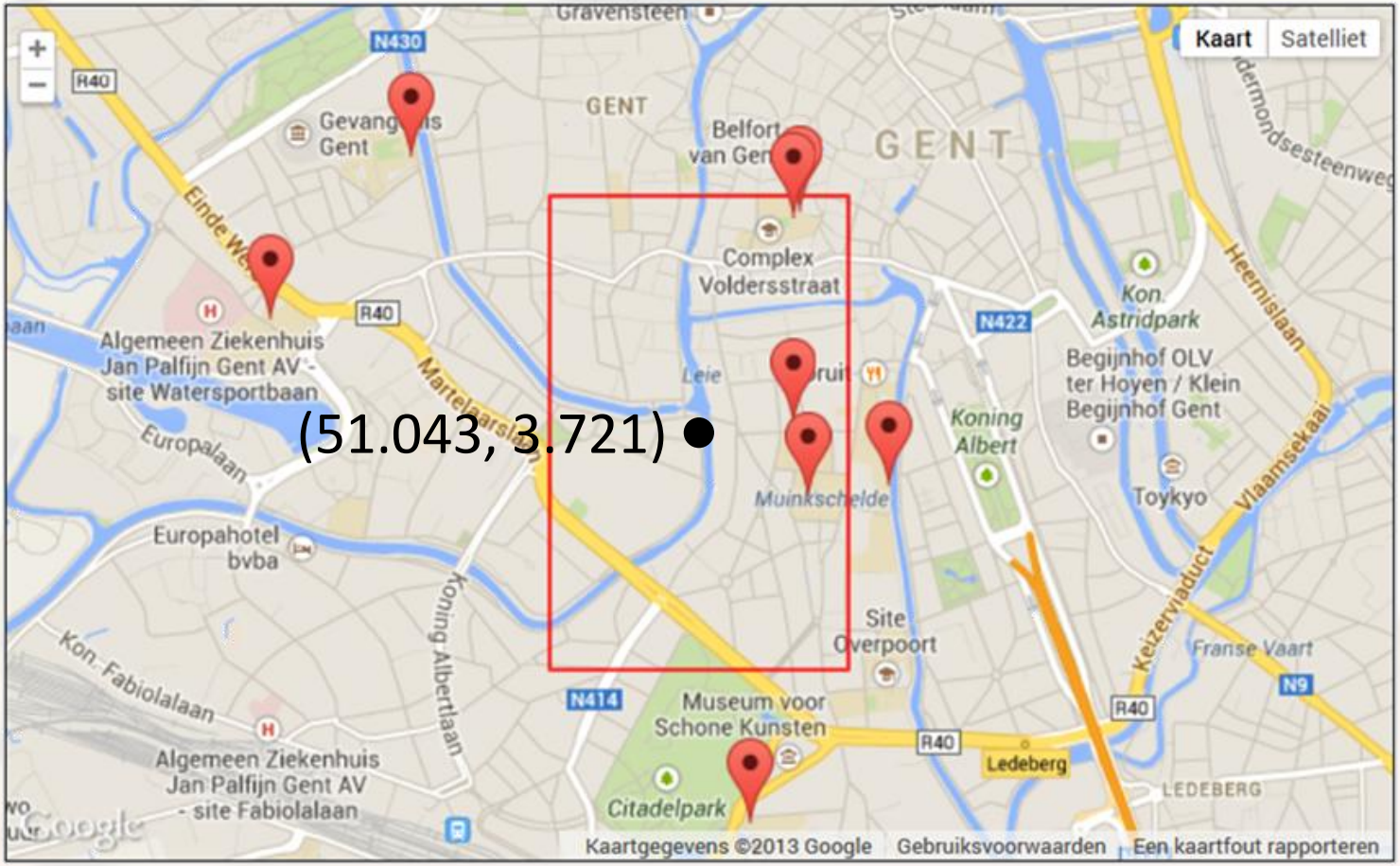
Location filter



- (51.043, 3.721)

Euclidean distance $$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

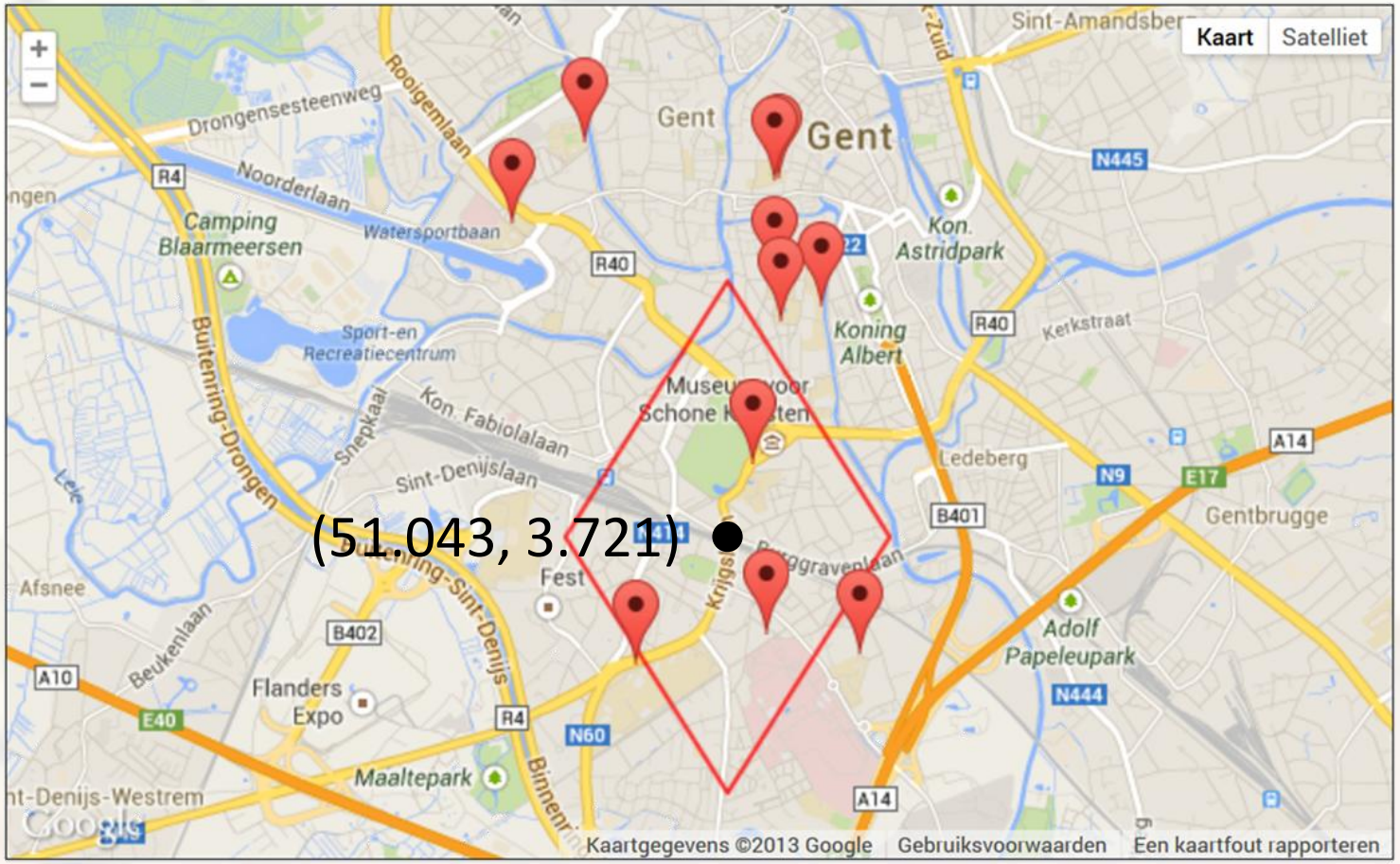
Location filter



(51.043, 3.721)

chessboard distance $d = \max(|x_1 - x_2|, |y_1 - y_2|)$

Location filter



Manhattan distance $$d = |x_1 - x_2| + |y_1 - y_2|$$

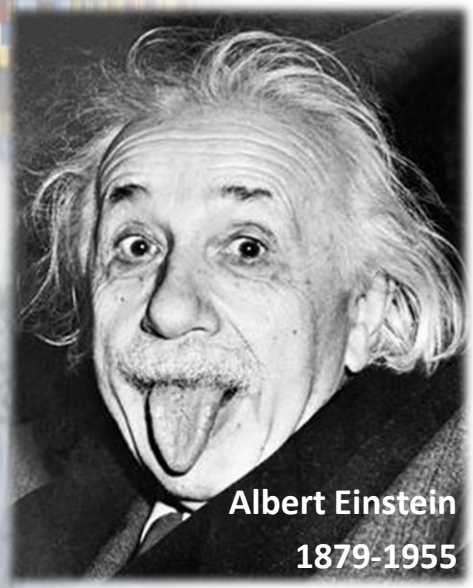
Homework (next lecture)

- course book
  - read chapter 9 (sets and dictionaries)
  - read chapter 10 (more program development)
- read problem description of classroom exercises
  - Simulating polymers
  - Cryptograms
  - Blood types

Questions or remarks?

UNIVERSITEIT GENT

The sky is the limit…



"Everything should be simple:
as simple as possible,
but no simpler."

— A. Einstein

UNIVERSITEIT GENT