Python Fundamentals

Getting Started

Robert Smallshire
@robsmallshire
rob@sixty-north.com

Presenter

Austin Bingham
@austin_bingham
austin@sixty-north.com

pluralsight
hardcore developer training

python™

python™

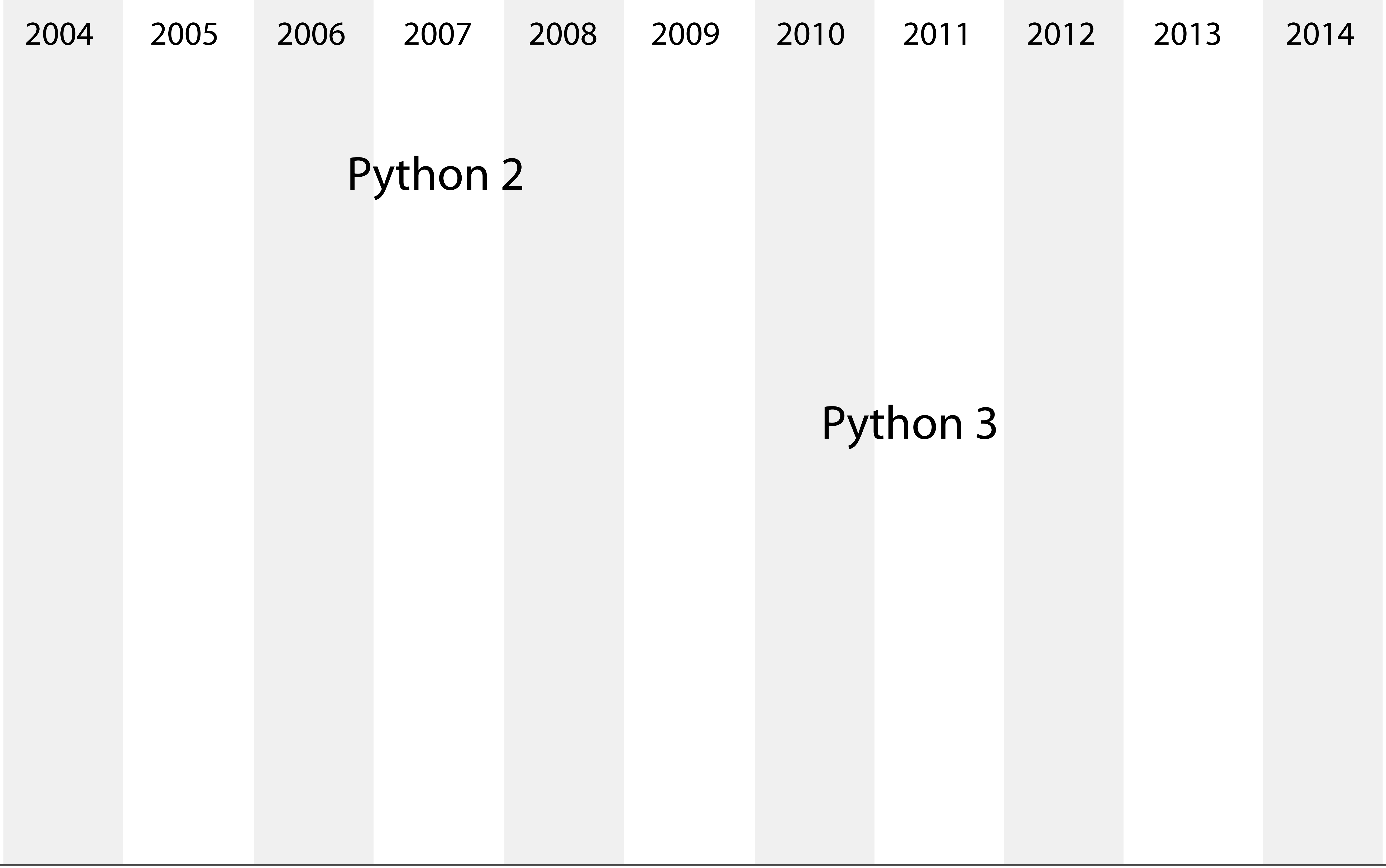
Windows

ubuntu

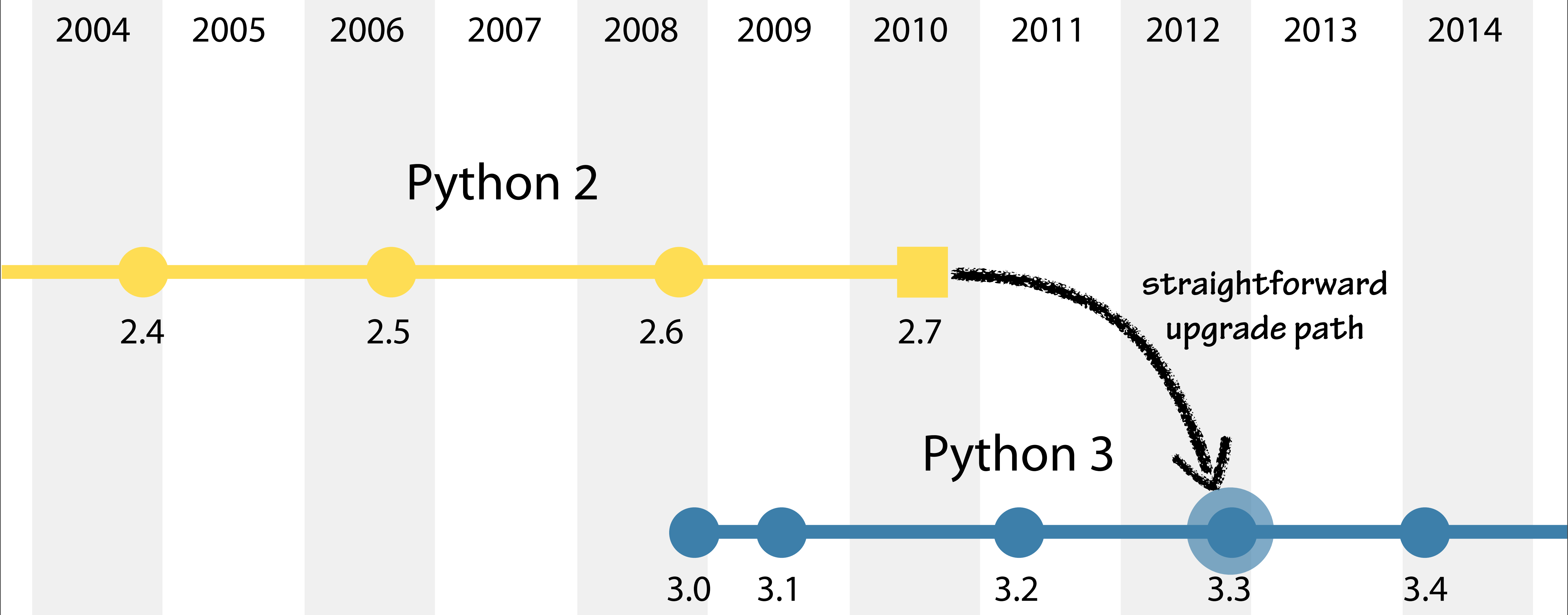
OS X

```
>>> if h > 50:
...     print("Greater than 50")
... elif h < 20:
...     print("Less than 20")
... else:
...     print("Between 20 and 50")
...
Between 20 and 50
```

Python Release Timeline



Python Release Timeline



Portable

python™

Platform Specific Installation

Windows

OS X

ubuntu

python™

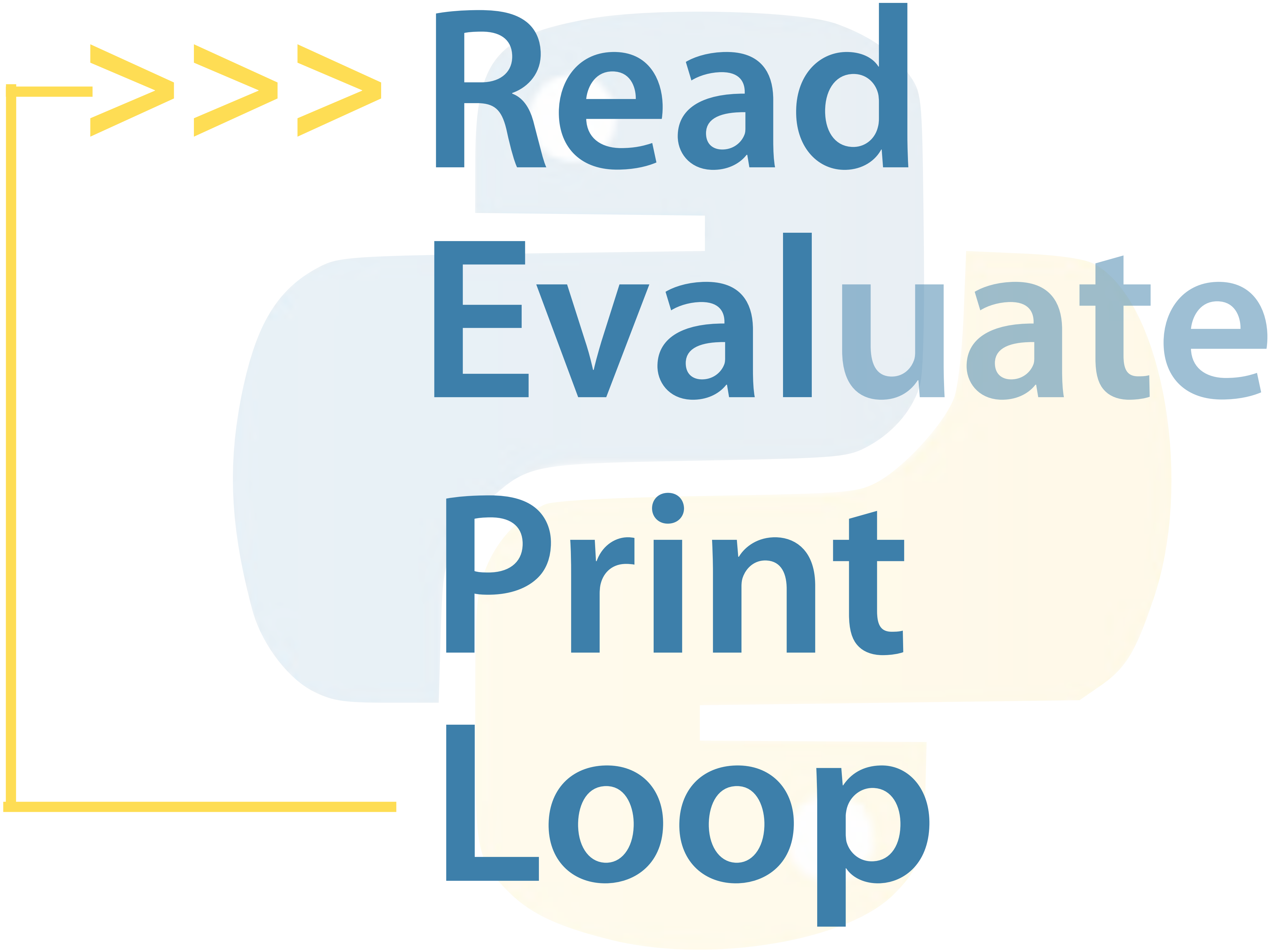
Windows

python™

ubuntu

python™

OSX

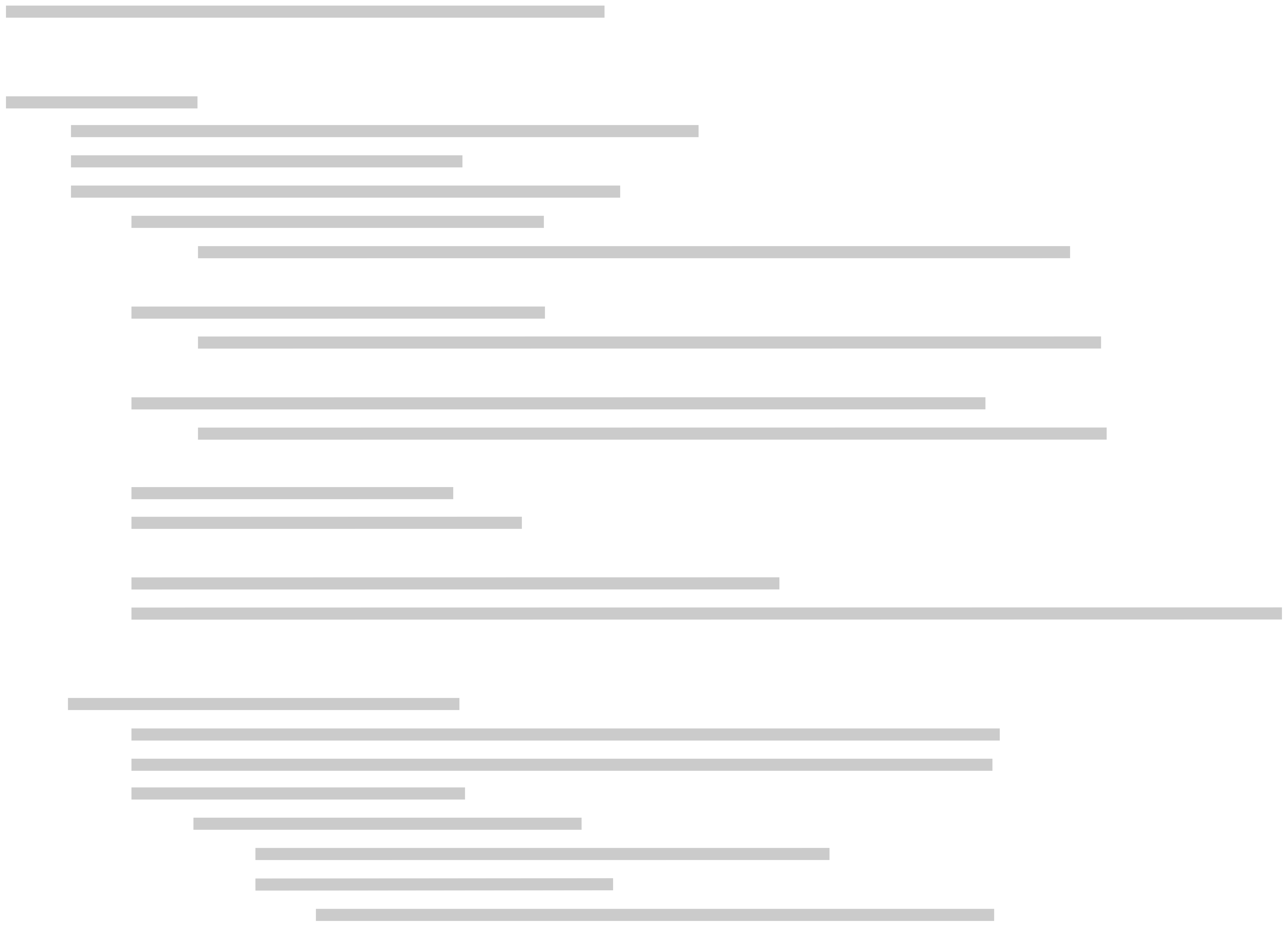

>>>
Read
Evaluate
Print
Loop

>>>REPL

Significant Indentation in Python

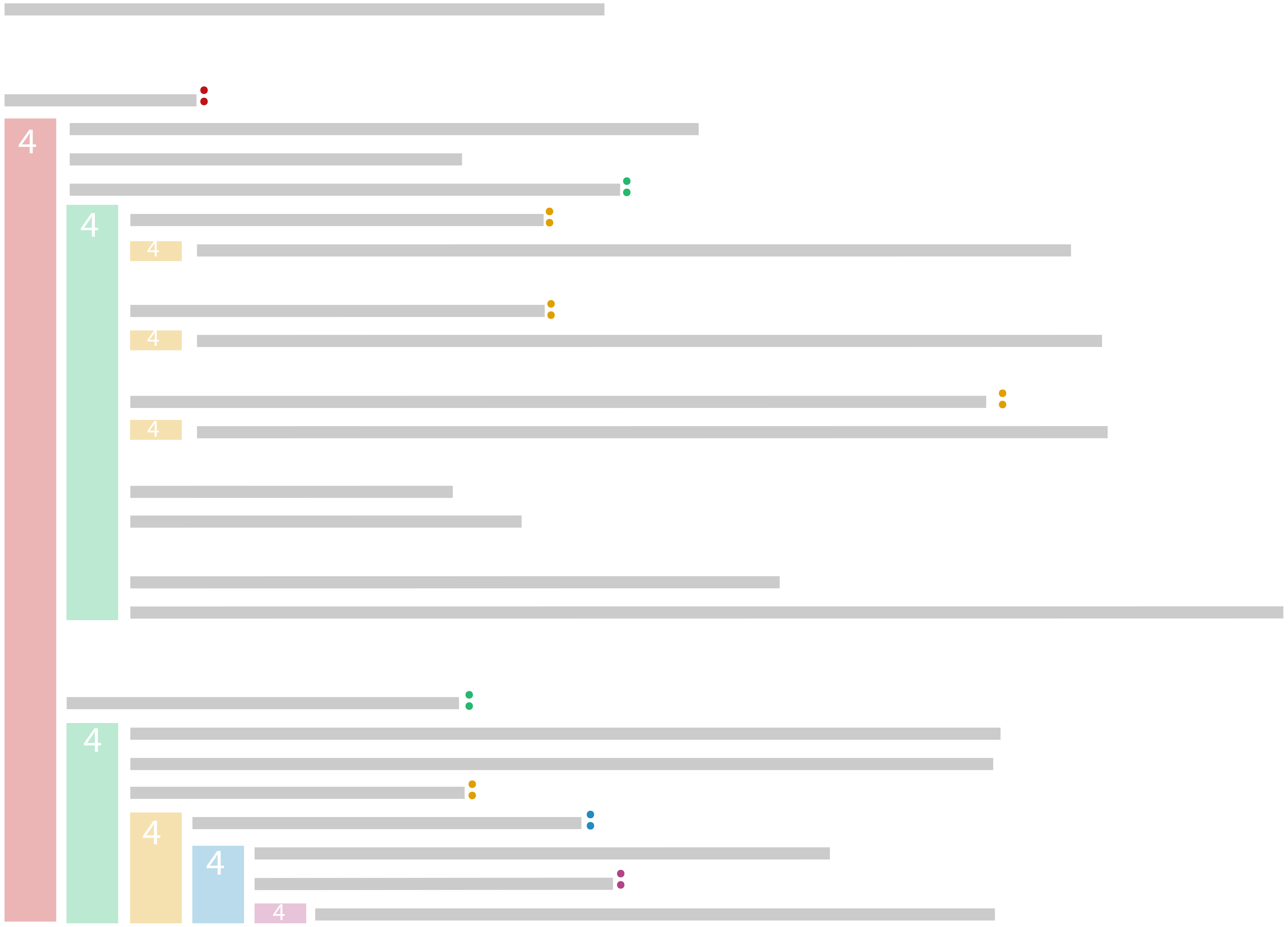
```
"""Class model for aircraft flights."""


class Flight:
    """A flight with a particular aircraft."""

    def __init__(self, number, aircraft):
        if not number[:2].isalpha():
            raise ValueError("No airline code in '{}'".format(number))

        if not number[:2].isupper():
            raise ValueError("Invalid airline code '{}'".format(number))

        if not (number[2:].isdigit() and int(number[2:]) <= 9999):
            raise ValueError("Invalid route number '{}'".format(number))

        self._number = number
        self._aircraft = aircraft

        rows, seats = self._aircraft.seating_plan()
        self._seating = [None] + [ {letter:None for letter in seats} for _ in rows ]




    def _passenger_seats(self):
        """An iterable series of passenger seating allocations."""
        row_numbers, seat_letters = self._aircraft.seating_plan()
        for row in row_numbers:
            for letter in seat_letters:
                passenger = self._seating[row][letter]
                if passenger is not None:
                    yield (passenger, "{}{}".format(row, letter))
```

Significant Indentation in Python



Significant Indentation in Python

Four spaces per level of indentation



Significant Whitespace

1. Requires **readable** code

2. No **clutter**

3. Human and computer can’t get **out of sync**

Significant Whitespace Rules

1. Prefer **four spaces**

2. **Never** mix spaces and tabs

3. Be **consistent** on consecutive lines

4. Only deviate to **improve** readability

Programming as Guido ~~intended~~ it

indented

Moment of Zen

Readability Counts

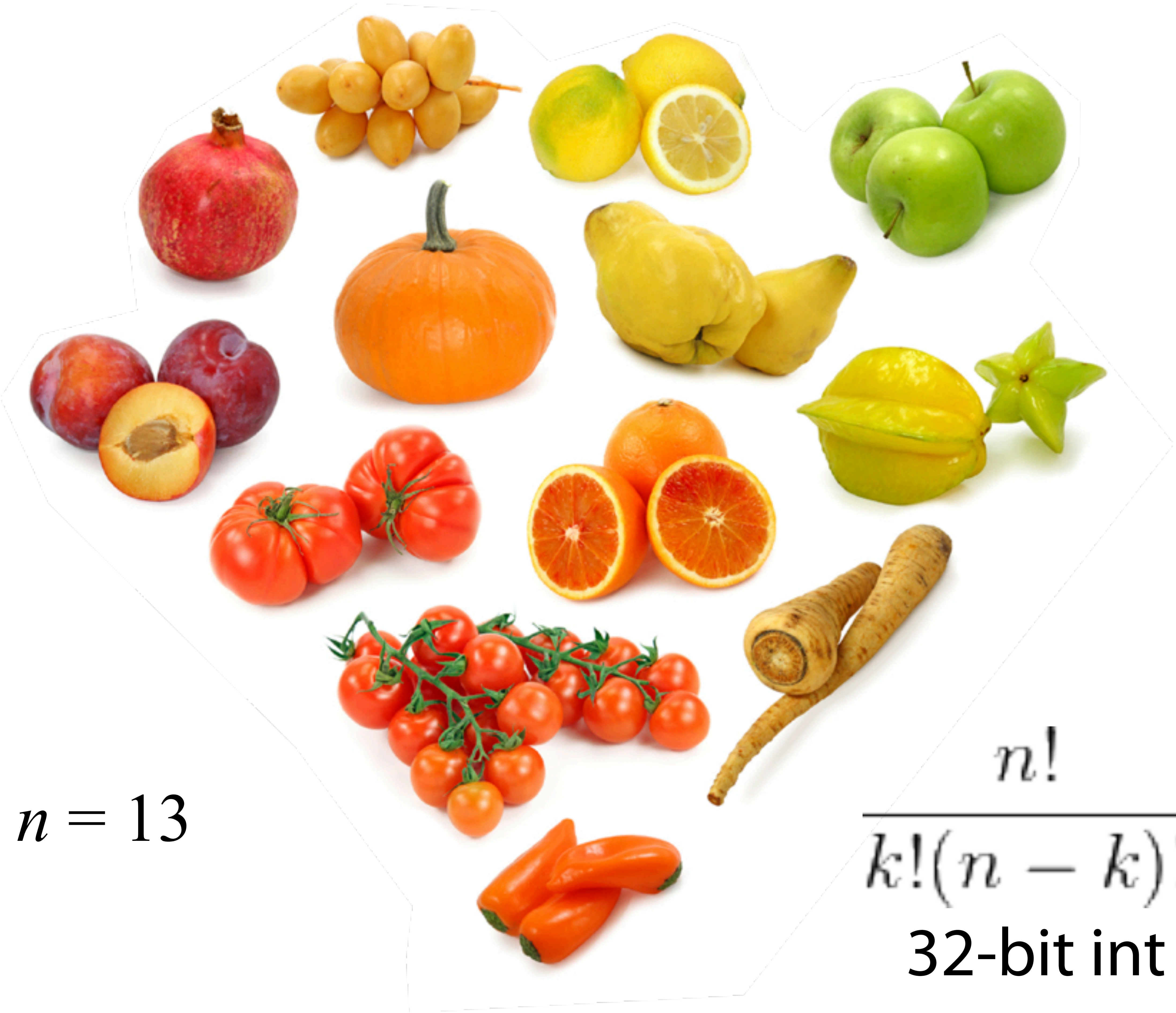
Clarity Matters
So readability makes
For valuable code

Python Standard Library

`import module_name`

python™

$$\frac{n!}{k!(n-k)!}$$



$n = 13$

$$\frac{n!}{k!(n-k)!}$$

32-bit int

Scalar types and values

python Scalar types and values

int	42	arbitrary precision integer
float	4.2	64-bit floating point numbers
NoneType	None	the null object
bool bool	True False	boolean logical values

int

unlimited precision signed integer

python™

float

IEEE-754 double precision (64-bit)

53 bits of binary precision

15 to 16 bits of decimal precision

None

The sole value of `NoneType`.

Often used to represent the absence of a value.

Not displayed by the REPL.

`bool`

Boolean logical value.

Either `True` or `False`.

python Relational Operators

==	value equality / equivalence
!=	value inequality / inequivalence
<	less-than
>	greater-than
<=	less-than or equal to
>=	greater-than or equal to

python Conditional Statements

```
if expr:
    print(”expr is True”)
```

expr is converted to bool as if by the bool() constructor

```
if h > 50:
    print(”Greater than 50”)
else:
    if h < 20:
        print(”Less than 20”)
    else:
        print(”Between 20 and 50”)
```

```
if h > 50:
    print("Greater than 50")
else:
    if h < 20:
        print("Less than 20")
    else:
        print("Between 20 and 50")
```

```
if h > 50:
    print("Greater than 50")
else:
    if h < 20:
        print("Less than 20")
    else:
        print("Between 20 and 50")
```

Python provides the `elif` keyword to eliminate the need for nested `if` ... `else` structures in many cases.

```python
if h > 50:
    print("Greater than 50")
elif h < 20:
    print("Less than 20")
else:
    print("Between 20 and 50")
```

Python provides the `elif` keyword to eliminate the need for nested `if` ... `else` structures in many cases.

Flat is better than nested

```
if h > 50:
    print("Greater than 50")
elif h < 20:
    print("Less than 20")
else:
    print("Between 20 and 50")
```

Python provides the `elif` keyword to eliminate the need for nested `if` ... `else` structures in many cases.

python while loops

```
while expr:
    print("loop while expr is True")
```

expr is converted to bool as if by the bool() constructor

python breaking out

```
while True:
    if expr:
        break
print("Go here on break")
```

The `break` keyword terminates the innermost loop, transferring execution to the first statement after the loop

python Getting Started – Summary

- **Obtaining and installing Python 3**
  - Windows
  - Ubuntu Linux
  - Mac OS X
- **Read-Eval-Print-Loop or REPL**
- **Simple arithmetic with + - * / % and //**
- **Assigning objects to named variables with the = operator**
- `print()`
- **Exiting the REPL**
  - Ctrl-Z on Windows
  - Ctrl-D on Linux and Mac.
- **Significant indentation - usually four spaces**
- **Python Enhancement Proposals**
  - PEP 8 – Python Style Guide
  - PEP 20 – The Zen of Python

Getting Started – Summary

- **Importing Python Standard Library modules:**
  - `import module`
  - `from module import function`
  - `from module import function as alias`
- **Finding and browsing** `help()`
- **Scalar built-in types**
  - `int` `float` `None` `bool`
  - conversions between types
- **Relational operators == != < > <= >= for equivalence and ordering**
- **Conditional statements with** `if` **...** `elif` **...** `else`
- `while` **loops**
- **Interrupting execution with Ctrl-C to create a** `KeyboardInterrupt` **exception**
- **Breaking out of loops with** `break`
- **Augmented assignment operators for modifying objects in-place**
- **Requesting text from the user with** `input()`