Python, The Joyful Parts

presented on the first pymug meeting on the 10th of March 2019

- by AR. J.

Python is pure delight

- cool syntax
- superpowers
- no main function to begin with

C++

```cpp
#include <iostream>
using namespace std;


int main() {
    // code here

}
```

Java

```java
public class MyJavaProgram {

    public static void main(String []args) {
        // code here

    }
}
```

Python

```
```

Popularity

- Most wanted language
- `3rd` most popular
  if you remove html, css, sql and bash
  (1st js, 2nd Java)
- confirmed `3rd` by being 3rd loved

source: SO survey 2018

Print yeah, but feature packed! [1]

- normal

```
print(1)
```

- many

```
print(1, 2, 3, 4)
# 1 2 3 4
```

- different data types with no casting

```
print(1, 'a', [1, 2, 3])
```

Print yeah, but feature packed! [2]

- cool control

```
print(1, 2, end=' ')
print(3, 4)
# 1 2 3 4
```

- even nicer

```
print(1, 2, 3, 4, sep='+')
# 1+2+3+4
```

Print yeah, but feature packed! [3]

std output next door

- normal

```
print(1, 2, file=sys.stdout)
```

- redirect to file

```
print(1, 2, file=open('log.txt', 'w+'))
```

- since stream, IO concepts integrated

```
print(1, 2, file=open('log.txt', 'w+'), flush=True)
```

Variables, candy enough to be mentionned here [1]

- multiple assignment

```
x, y, z = 1, 2, 3
```

- where py stands out

```
a, *x = 0, 1, 2, 3
# a -> 0, x -> [1, 2, 3]
```

- middle

```
a, *x, b = 0, 1, 2, 3
# x -> [1, 2]
```

Variables, candy enough to be mentionned here [2] - variable swapping

- C++

```cpp
#include <iostream>
using namespace std;
int main()
{
    int a = 5, b = 10, temp;
    temp = a;
    a = b;
    b = temp;
    return 0;
}
```

- python

```python
a, b = 4, 5
a, b = b, a
```

Variables, candy enough to be mentionned here [3]

- multiline string in variable without \n

```
x = """
<div>
    <a href="#">python.org</a>
</div>
"""
```

- same assignment

```
x = y = z = 0
```

Printing own file content, a two-liner

- the 2 lines below prints the file content

```
with open(__file__) as f:
    print(f.read())
```

loooops [1]

- iterating over elements

```
fruits = ['apple', 'orange', 'apple']
for fruit in fruits:
        print(fruit)
# apple
# orange
# apple
```

- range

```
for i in range(5):
        print(i, end=' ')
# 0 1 2 3 4
```

loooops [2]

- more control

```
for i in range(5, 15, 2):
        print(i, end=' ')
# 5 7 9 11 13
```

- indexing: same spirit

```
x = 'abcdefghijklmnopqrstuvwxyz'
print(x[5:15:2])
# fhjln
```

A fact on bool

- True equals one

```
if True == 1:
        print('ok')
# ok
```

- we can add

```
x = True + False
# 1
```

- mix

```
x = True + 1 + 2 + 3
# 7
```

lists and strings [1]

- reverse list or ... string

```
'abc'[::-1]
# cba
```

- easy palindrome check

```
def p(word):
    if word == word[::-1]:
        return True
    return False
```

lists and strings [2]

- populating lists

```
x = [i for i in range(5)]
# [0, 1, 2, 3, 4]
```

- even numbers generation

```
x = [i for i in range(10) if i%2 == 0]
# [0, 2, 4, 6, 8]
```

lists and strings [3]

useful in-builts

- sum

```
sum([1, 2, 3])
# 6
```

- max

```
max([1, 3, 5, 2, 9, 12])
# 12
```

- count

```
x = [1, 2, 3, 1, 4, 5].count(1)
# 2
```

lists and strings [4]

- custom checks

```
def check(x):
    if x > 15:
        return True
    return False

x = [check(i) for i in range(20)]
# [False, ..., True, True]
```

- getting count at the same time

```
x = sum([check(i) for i in range(20)])
# 4
```

lists and strings [5]

- but simple checks are easy

```
x = 5
print(0 < x)
# True
```

- previous example

```
x = sum([i > 15 for i in range(20)])
```

- also in between

```
x = sum([ 0 > i > 15 for i in range(20)])
```

lists and strings [5]

- list to string

```
x = ['a', 'b', 'c']
string = '-'.join(x)
# a-b-c
```

- convert integer list to string

```
# [str(i) for i in x]

x = [1, 2, 3]
string = '-'.join([str(i) for i in x])
# 1-2-3
```