programming language ⇒ give instructions to machines (source code) ex. JS, PY, JAR

↓ (translator)

interpreter (usually what PY use) ⇒ line by line

compiler ⇒ all in once

cpython (C) | Jpython (Java) | PyPy | Iron python (.net)

py ⇒ interpreter ⇒ bit code ⇒ cpython VM ⇒ machine code

↘ what we are downloading

print('....') ⇒ can be single or double quote

name = input('what is your name')

print('hello' + name)

Terms: data types (int ...) | Actions (print ...) ⇒ functions

Integrated Development Environment

Terminal | Code Editors | IDEs | Notebooks

or glot.io

repl.it & run python online

2008

python 2 vs. python 3

↓

does not support python 2

Guido & creator

created 1991

python usually slower than Java, C++ ex..

↑ but more productive

why are so many languages ⇒ all about trade offs

Fundamental Data Types

int

print(type(2+4)) ⇒ <class 'int'>

print(type(2/4)) ⇒ <class 'float'>

print(type(0)) ⇒ <class 'int'>

print(type(20+1.1)) ⇒ <class 'float'>

print(2**2) ⇒ $2^2 = 8$

print(3//4) ⇒ a int rounded down ⇒ 0

print(5%4) ⇒ reminder

float ⇒ a number with decimal ⇒ takes up more space in memory than int

10.56 — 10 and 56 stored in different locations

math functions

print(round(3.1)) ⇒ 3

print(abs(-20)) ⇒ 20

operator precedence

() ⇒ ** ⇒ * or / ⇒ + or -

complex ⇒ another data type for numbers ⇒ only for high level math

bin(5) ⇒ return the binary version of the number

print(bin(5)) ⇒ 0b101 ← converted to binary

print(int('0b101', 2)) ⇒ 5 ← converted to Int

binary | base 2

str

long_string = ''' # with three ' can go multi lines

WOW

O O

'''

print(long_string)

classes ⇒ custom types

Specialized Data Types

other type

None, complex

developers foundamentals

// do not read the dictionary

know what exists and google

focus on the things we are actually using

when commenting ⇒ simple, clean, only when needed

variables

iq = 100 # binding

print(iq)

snake_case

start with lowercase or underscore — usually signify a private variable

case sensitive

Do not overwrite the key words

constants

PI = 3.14 ⇒ capitalized name

dunder

__ ⇒ two underscore & do not use

assign multi variables

a, b, c = 1, 2, 3

expressions vs. statements

iq = 100 ⇒ another statement

user_age = iq / 5 ⇒ expression

↓ statement

formatted strings

```
name = 'John'
age = 55

# print('hi' + name + '. You are ' + str(age) + ' years old')
print(f'hi {name}. You are {age} years old') # python 3
print('hi {0}. You are {1} years old'.format(name, age)) # python 2
print('hi {new_name}. You are {age} years old'.format(new_name='sally', age=100))
```

augmented assignment operator

```
some_value = 5
some_value += 2   # -=, *=, /=
```

↓ augmented assignment operator

Escape sequence

```
weather = "It\'s \" kind o\" sunny"
```

⇒ whatever comes after \ will be recognized as string

\t ⇒ add a tab

\n ⇒ add a new line

String indexes #17 -5:48

```
selfish = '01234567'
```

include ⇒ not include

#[start : stop : stepover]

```
print(selfish[0:8]) ⇒ '01234567'
print(selfish[1:]) ⇒ '1234567'
print(selfish[-1]) ⇒ '7'
print(selfish[::-1]) ⇒ '76543210'
```

immutability

strings are immutable

```
selfish[0] = '8'   ⇐ does not work
selfish = selfish + '8'   ⇐ works
```

got the dot → .format()

owned by something

print()

built in functions + methods

```
print(len('01234')) ⇒ return the length '5'
.upper()
.capitalize()
.find('be') ⇒ how many times 'be' exist in a string
.replace('be', 'me')

quote = 'to be or not to be'
print(quote.replace('be', 'me'))
```

not change the original one

string are immutable

```
print(quote) ⇒ 'to be or not to be'
```

exercise

```
birth_year = input('what year were you born?')
age = 2019 - int(birth_year)
print(f'your age is: {age}')
```

exercise 2

```
pass_length = len(pass)
hidden_pass = '*' * pass_length
```

Booleans

```
print(boo(0)) ⇒ False
print(boo('True')) ⇒ True
```

lists → a form of Array, collection of data types

ordered sequence of objects that can be any type

```
li = [1, 25, 'a', True]
```

list slicing → similar to string slicing, but mutable

```
li = ['A', 'B', 'C', 'D']
li[1] = 'F'
print(li) ⇒ ['A', 'F', 'C', 'D']
print(li[0:2]) ⇒ ['A', 'F'] ⇒ not changing the original one
print(li) ⇒ ['A', 'F', 'C', 'D']
```

if want to copy the value

```
li2 = li[:]   ⇐ make it a value type
li2 = li      ⇐ will make it a reference type,
                changes to li2 will influence li
```

matrix a way to describe multi deminision lists

```
matrix = [
    [1, 0, 1],
    [0, 1, 0],
    [1, 0, 1]
]
print(matrix[0][1]) ⇒ '0'
```

tuple

set

dict

List methods

```
basket = [1, 2, 3, 4, 5]

#adding
new_list = basket.append(100) # not return the result, change the original one
print(basket) => [1, 2, 3, 4, 5, 100]
print(new_list) => None

#insert
new_list = basket.insert(4, 100)  # not return the value, insert before index 4.
print(basket) => [1, 2, 3, 4, 100, 5]
print(new_list) => None

#extend
new_list = basket.extend([100, 101]) # not return the value
print(basket) => [1, 2, 3, 4, 5, 100, 101]
print(new_list) => None

#removing
basket.pop() => [1, 2, 3, 4] => remove the last one
basket.pop(0) => [2, 3, 4] => remove the 0 index => return the value removed

basket.remove(4) => [2, 3] => remove the "4" value => not return the value removed

#clean
new_list = basket.clear() # not return the value
print(new_list) #None

#index
basket = ['a', 'b', 'c', 'd', 'e']
print(basket.index('d')) => 3 & the index of 'd'
print(basket.index('d', 0, 3)) => error
print(basket.index('d', 0, 4)) => 3 => search 'd' start 0 and before 4)

print('d' in basket) => true

print('i' in 'hi my name') => true
print(basket.count('d')) => 1

#sort          -> method
basket.sort() => not return things

sorted(basket => return a new array, not change the original one
↓
function

new_basket = basket.copy()  => similar to  new_basket = basket[:]

#reverse
basket.sort()                          -> return a new list, not change the original one
basket.reverse()  => basket[::-1]
↓
not return new one, change the original one
```

#common usage

print(list(range(1, 100))) ⇒ a list from 1 to 99
- 1: start
- 100: befor this number

print(list(range(100))) ⇒ a list from 0 to 99

#.join()

sentence = '!'

new_sentence = sentence.join(['hi', 'my', 'name']) → return a string, not change the original one

print(new_sentence) ⇒ hi! my! name

''.join([])

#31