syntax

Document Link

1. No need to use semicolon for ending statement like other language.

```
# define main function to print out something
def main():
    i = 1
    max = 10
    while (i < max):
        print(i)
        i = i + 1

# call function main
main()
```

So Python officially doesn't support ++ or --

2. Divide line use backslash

Python uses a newline character to separate statements. It places each statement on one line.
However, a long statement can span multiple lines by using the backslash (\) character.
The following example illustrates how to use the backslash (\) character to continue a statement in the second line:

```
if (a == True) and (b == False) and \
   (c == True):
    print("Continuation of statements")
```

3. Identifier

Python identifiers are case-sensitive. For example, the **counter** and **Counter** are different identifiers.

4. **Keywords**

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

5. String literals

Python uses single quotes ('), double quotes ("), triple single quotes (''') and triple-double quotes (""") to denote a string literal.

The string literal needs to be surrounded with the same type of quotes. For example, if you use a single quote to start a string literal, you need to use the same single quote to end it.
The following shows some examples of string literals:

```
s = 'This is a string'
print(s)
s = "Another string using double quotes"
print(s)
s = ''' string can span
      multiple line '''
print(s)
s = “”” string can span use triple double code
      multiple line “””
print(s)
```

Summary

A Python statement ends with a newline character.

Python uses spaces and indentation to organize its code structure.

Identifiers are names that identify variables, functions, modules, classes, etc..

Comments describe why the code works. They are ignored by the Python interpreter.

Use the single quote, double-quotes, triple-quotes, or triple double-quotes to denote a string literal.

operator

Arithmetic Operators :

Operator	Description	a	b	Example	Result
+	Addition	5	2	a + b	7
-	Subtraction	5	2	a - b	3
*	Multiplication	5	2	a * b	10
/	Division (Floating)	5	2	a / b	2.5
//	Floor Division	5	2	a // b	2
%	Modulus (Remainder)	5	2	a % b	1
**	Exponentiation	5	2	a ** b	25

Assignment operator :

Operator	Description	a	b	Example	Equivalent To	Result
+=	Add and Assign	5	2	a += b	a = a + b	7
-=	Subtract and Assign	5	2	a -= b	a = a - b	3
*=	Multiply and Assign	5	2	a *= b	a = a * b	10
/=	Divide and Assign	5	2	a /= b	a = a / b	2.5
//=	Floor Divide and Assign	5	2	a //= b	a = a // b	2
%=	Modulus and Assign	5	2	a %= b	a = a % b	1
**=	Exponentiate and Assign	5	2	a **= b	a = a ** b	25

Comparison operators :

Python has six comparison operators, which are as follows:

- Less than (`<`)
- Less than or equal to (`<=`)
- Greater than (`>`)
- Greater than or equal to (`>=`)
- Equal to (`==`)

- Not equal to (`!=`)

Logical Operator :

Python has three logical operators:

- `and`
- `or`
- `not`

a and b

a	b	a and b
True	True	True
True	False	False
False	False	False
False	True	False

a or b

a	b	a or b

True	True	True
True	False	True
False	True	True
False	False	False

not a

a	**not a**
True	False
False	True

controlflow

If … else :

```
age = input('Enter your age:')
if int(age) >= 18:
    print("You're eligible to vote.")
```

Ternary operator :

In **Python**, that exact ? : syntax **does not exist**.

```
ticket_price = 20 if age >= 18 else 5
```

For loop

```
for index in range(5):
    print(index)
```

There is **no for index of range(5) syntax**. Python don't support for index of range() syntax.

Summary

- Use the for loop statement to run a code block a fixed number of times.
- Use the range(start, stop, step) to customize the loop.

While loop

```
while num1 > num2 :

    print("Number is = ", num1)

    num2 += 1
```

Break

```
for index in range(0, 10):

    print(index)
    if index == 3:
        Break
```

Continue

```
for index in range(10):
    if index % 2:
        continue
    print(index)
```

pass

```
if condition:
    pass
for i in range(1,100):
    pass
while condition:
    pass
def fn():
    pass
class Stream:
    pass
```

Summary

- Use the Python `pass` statement to create a placeholder for the code that you'll implement later.

function

Function

```
def sum():
    return 10 + 20

total = sum()
print(total)
```

Default parameter function

- Place default parameters after the non-default parameters.

```
def greet(name, message='Hi'):
    return f"{message} {name}"

greeting = greet('John')
print(greeting)
```

Parameterize function

```
def get_net_price(price, discount):
    return price * (1-discount)

net_price = get_net_price(100, 0.1)
print(f'{net_price: .2f}')
```

Summary

- A Python function is a reusable named block of code that performs a task or returns a value.
- Use the `def` keyword to define a new function. A function consists of function definition and body.

- A function can have zero or more parameters. If a function has one or more parameters, you need to pass the same number of arguments into it.
- A function can perform a job or return a value. Use the `return` statement to return a value from a function.

Keyword Arguments

The following calls the `get_net_price()` function and uses the default values for `tax` and `discount` parameters:

```
def get_net_price(price, tax_rate=0.07, discount=0.05):
    discounted_price = price * (1 - discount)
    net_price = discounted_price * (1 + tax_rate)
    return net_price

net_price = get_net_price(100)
print(f'{net_price: .2f}')
```

Suppose that you want to use the default value for the `tax` parameter but not `discount`. The following function call doesn't work correctly.

```
def get_net_price(price, tax_rate=0.07, discount=0.05):
    discounted_price = price * (1 - discount)
    net_price = discounted_price * (1 + tax_rate)
    return net_price

net_price = get_net_price(100, 0.06)
print(f'{net_price: .2f}')
```

… because Python will assign `100` to price and `0.1` to tax, not discount.

To fix this, you must use keyword arguments:

```
def get_net_price(price, tax_rate=0.07, discount=0.05):
    discounted_price = price * (1 - discount)
    net_price = discounted_price * (1 + tax_rate)
    return net_price

net_price = get_net_price(
    price=100,
    discount=0.06
)
print(f'{net_price: .2f}')
```

Recursive Functions

A recursive function is a function that calls itself until it doesn’t.

```
def fn():
    # ...
    fn()
    # ...

def count_down(start):
    print(start)
    next = start - 1
    if next > 0:
        count_down(next)

count_down(3)
```

```
# use for
def sum_hundred(n) :
    total = 0
    for index in range(n+1) :
        total += index

    print(total)

sum_hundred(100)

# recursive
def sum_recursive(n) :

    if n > 0 :
      return  n + sum_recursive(n-1)
    return 0

result =sum_recursive(100)

print(result)

# use ternary operator call recursive
def ternary_recursive_sum(n) :
    return n + ternary_recursive_sum(n-1) if n > 0 else 0

res = ternary_recursive_sum(100)
print(res)
```

Summary
A recursive function is a function that calls itself until it doesn't.
And a recursive function always has a condition that stops calling itself.

Lambda function

What are Python lambda expressions

Python lambda expressions allow you to define anonymous functions.

Anonymous functions are functions without names. The anonymous functions are useful when you need to use them once.

A lambda expression typically contains one or more arguments, but it can have **only one expression**.

General function example

```
def add(a, b):
    return a + b

print(add(5, 3))
```

Lambda function example

```
add = lambda a, b: a + b
print(add(5, 3))
```

Summary

- Use Python lambda expressions to create anonymous functions, which are functions without names.
- A lambda expression accepts one or more arguments, contains an expression, and returns the result of that expression.
- Use lambda expressions to pass anonymous functions to a function and return a function from another function.

list

Basic

Python

Youtube link

1. For getting input we use
2. input(‘write input value’’) method. [python], scanf() [for c] cin() for [c++] and new Scaner () for java
3. For comment we use = # for single line. For multiple line we use = … comment message … or “”” cmt message “””
4. Data type = int,float/double,char,string,bool,